

OpenLatch documentation

Install the client, define each agent's Autonomy Zone, and roll out across your fleet.

Install the client

How verdicts work →

Get started



Install the client and see your first verdict.

Concepts



Autonomy Zones, intent policies and verdicts.

Guides



One task per page, from authoring to Enforce.

Deploy



Roll the client out through MDM or config management.

Reference



Every command, error code and policy field.

Changelog



Every client release and platform change.

Overview

This path takes you from a fresh machine to your first decided action: you install the client next to an agent, sign in to the console, write one intent policy and watch it decide.

OpenLatch is the execution control layer for enterprise AI agents. The client runs next to each agent and decides on every covered action before it executes. The platform, at `app.openlatch.ai`, is where you define each agent's Autonomy Zone and read the record of what happened.

- 1

[Install the client](#) on a machine where a coding agent runs.
- 2

[Sign in to the console](#) with the method your organization uses.
- 3

[Write your first policy](#) and ratify it in Monitor.
- 4

Use the agent as usual, then open **Anomalies** in the console to see what the policy recorded.

Who does what

You are	Start with
A developer with a coding agent	Install the client
An administrator rolling the client out to many machines	Install the client , then the Deploy section
A security or compliance owner who writes policy	Write your first policy , then Intent policies

Before you start

- A macOS, Linux or Windows machine with at least one coding agent the client wires. [Control levels](#) lists them, plane by plane.
- An account in an OpenLatch organization at `app.openlatch.ai`.

To learn what OpenLatch is before you install anything, read [What is OpenLatch](#).

Install the client

Install the client on the machine where your coding agent runs, so that every covered action is decided before it executes. The client installs at user scope and needs no administrator rights. Pick the command that matches the machine.

On any machine that already has Node.js:

```
...  
  
npx @openlatch/client init
```

Node.js is needed by npx only. The client itself is a native binary and does not run on Node.js.

No runtime required:

```
...  
  
curl -LsSf https://openlatch.ai/install.sh | sh
```

No runtime required:

```
...  
  
powershell -ExecutionPolicy Bypass -c "irm https://openlatch.ai/install.ps1 | iex"
```

Each command installs the same client build from the npm registry, then runs `openlatch init`, which detects the coding agents on the machine, wires their hooks and starts the client.

Enroll the machine

- 1 On a machine where you sit at a terminal, `openlatch init` opens `app.openlatch.ai` in your browser when the machine has no credential yet. Sign in, and the client stores the credential in the operating system's keychain.

- 2 On a machine with no one at the terminal, create an API key on the **API Keys** page of your console account first, and set it as `OPENLATCH_API_KEY` before you run the install command.
- 3 Run `openlatch status` and check that each agent is listed with its control level.

 Note

The download is integrity-checked against the registry's own digest. That detects corruption and truncation, not a compromised registry.

Check the hooks

Run `openlatch doctor`. It diagnoses the configuration and the connection to the platform, and for anything that is not healthy it names the cause and the command that fixes it.

What the install changes

- It installs for your user only. No administrator rights are needed.
- With a script, it places two binaries, `openlatch` and `openlatch-hook`, in `~/ .local/bin` on macOS and Linux, or in `%LOCALAPPDATA%\openlatch\bin` on Windows, and adds that folder to your PATH. A terminal that was already open may need to be reopened once.
- With `npx`, nothing is added to your PATH: the client keeps its binaries under its state directory, so a bare `openlatch` may not be found. Run later commands through the same `npx` command, with the command you want in place of `init`, for example `status`.
- It compares a SHA-512 digest of the download with the digest the registry publishes for that release before it installs anything.

Exit status

For deployment tools that read the exit status of an install script:

Status	Meaning
0	Installed and set up. The result may still name one thing to look at
6	Installed, but setup did not finish. The result on screen says why
7	Installed and waiting for an AI agent: none is on the machine yet. Install one and run the script again
2-5	Nothing usable was installed. The script prints an 0L - 21xx code naming the reason

Supported platforms

Platform	Status
macOS, Apple silicon and Intel	Supported
Linux with glibc, x86_64 and arm64	Supported
Windows, x64	Supported
Linux with musl, such as Alpine	Not supported. The install script stops and names the reason
Windows on ARM64	Not supported. The install script stops and names the reason

Next, [write your first policy](#).

Write your first policy

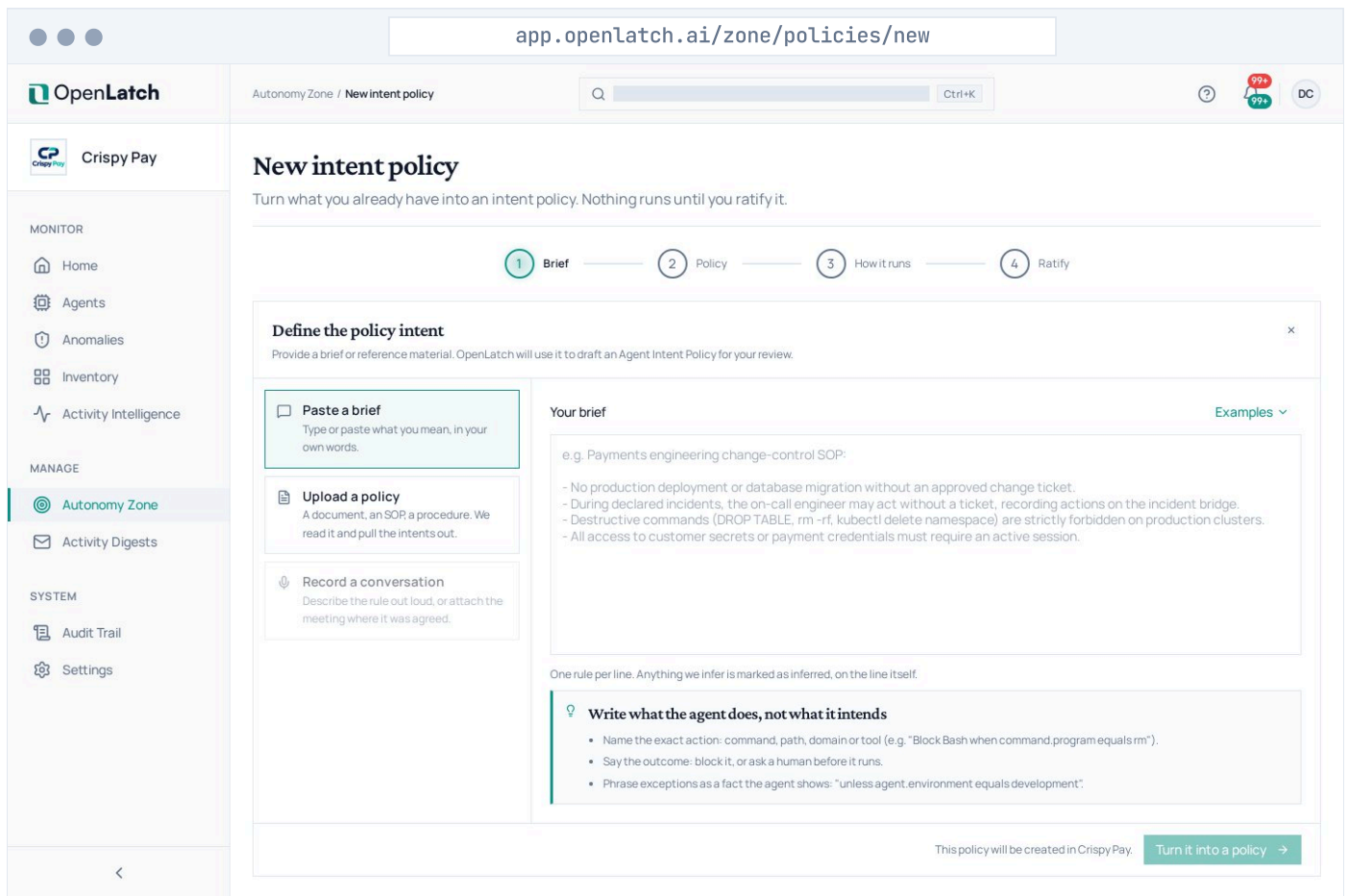
In this tutorial you turn one sentence into an Agent Intent Policy (AIP), ratify it in Monitor, and watch it record an anomaly without stopping the agent.

Before you start, [install the client](#) on a machine with a coding agent and [sign in to the console](#).

Write the policy

- 1 In the console, open **Autonomy Zone**, then the policies list, and choose **New intent policy**.
- 2 On the **Brief** step, paste one sentence that says what you want, or pick an example. This tutorial uses:

```
Don't let the agent fetch anything from pastebin.com.
```
- 3 On the **Policy** step, read the sentence OpenLatch generated from your brief, with the test cases shown beside it. Each case is a concrete situation and the verdict the policy would return. Answer Yes if the sentence says what you meant. Answer No and say what you meant in your own words, and OpenLatch regenerates the sentence and its cases.
- 4 On the **How it runs** step, choose the mode **Monitor**.
- 5 On the **Ratify** step, choose the zone that holds the policy, then run the replay. It applies the policy to your recent agent activity and shows what it would have decided. Ratify the policy.



The New intent policy wizard: Brief, Policy, How it runs, Ratify.

If the wizard says that authoring is unavailable, an administrator of your OpenLATCH site has not configured AI authoring yet.

See it record

- 1 Ratifying rebuilds the policy bundle. The client fetches it on its next poll; to fetch it now, run `openlatch system aip refresh` on the agent's machine.
- 2 Ask the agent to fetch a page from `pastebin.com`.
- 3 The policy runs in Monitor, so the action proceeds and an anomaly is recorded. Open **Anomalies** in the console to read it: the anomaly names the policy, its dimension and the zone it came from.

What you did

You wrote an intent policy in your own words, checked it against concrete cases, and ran it in Monitor, where it records and never stops the agent. When the anomalies show the policy is accurate, switch it to Enforce in its zone: the client then applies its verdict before the action runs. [Modes](#) explains the difference, and [Intent policies](#) explains what OpenLatch built from your sentence.

Sign in to the console

Sign in to the console at `app.openlatch.ai` to define Autonomy Zones, write intent policies and read what your agents did.

Sign in

- 1 Open `app.openlatch.ai`. The sign-in page offers the methods your organization allows.
- 2 Sign in with one of them: your organization's SAML single sign-on, a Google, Microsoft, GitHub, Apple or LinkedIn account, a magic link sent by email, a passkey, or an email address and password.
- 3 If your account has two-factor authentication turned on, enter the code from your authenticator app.

The first time you sign in without an invitation, the console walks you through creating your organization. When a colleague invited you, the invitation link takes you to your organization directly.

Link the client to your account

When `openlatch init` or `openlatch system auth login` needs a credential, it opens the console in your browser. Once you are signed in, the console creates an API key for the client and hands it back to the terminal, which stores it in the operating system's keychain. If the hand-back fails, the page shows the key so that you can copy it into the terminal.

The sign-in pages

Page	What it is for
/login	Sign in with any method your organization allows
/register · /signup	Create an account, from an invitation or on your own; /signup leads to /register
/forgot- password	Ask for a password reset link by email
/reset- password	Choose a new password from the reset link
/two-factor- verify	Enter the code from your authenticator app
/change- password	Choose a new password when your account must change it before it continues
/welcome	Create your organization after your first sign-in
/cli-auth	The page the client opens to receive its API key
/activate	A one-time setup screen a new OpenLatch site shows before its first organization exists. On <code>app.openlatch.ai</code> you never see it

What is OpenLatch

Read this page to understand what OpenLatch does before you install it.

OpenLatch is the execution control layer for enterprise AI agents. It lets you give agents more autonomy without losing control: you define the field each agent plays on, and every covered action the agent takes is decided before it executes.

The problem

AI agents do more than answer. They run commands, change files, call tools and spend money, often with broad permissions on the machine they run on. Instructions in a prompt shape what an agent intends, but they cannot guarantee what it does: a long context, a confused plan or an injected instruction can still lead to an action nobody wanted.

The approach

Principle	What it means
Inside the agent, not a proxy in front of it	The client plugs into the agent's own native hooks, so it sees each action while there is still time to change the outcome
Decided before impact	Every covered action reaches the policy decision point before it runs, and the verdict is enforced. A verdict is deterministic because an explicit policy produced it
Local-authoritative	The client decides on the machine, from a resident policy bundle. It keeps deciding with the network unplugged, and the agent never waits on the network for a verdict
Your intent, in your words	You write policy as a sentence. OpenLatch turns it into an Agent Intent Policy and shows you concrete test cases before you ratify it
Evidence after the fact	The client sends the record of each action and its verdict to the platform after the decision, for audit and review

The threat model is a misbehaving agent, not a malicious operator: OpenLatch controls what a sanctioned agent does through its own hooks. It is not an unbypassable sandbox.

The pieces

Piece	Where it runs	What it does
The client	Next to each agent, on a developer machine or wherever the agent executes	Wires the agent's hooks, holds the policy bundle, decides each covered action and records it
The platform	<code>app.openlatch.ai</code>	Holds the agent record, the Autonomy Zones and their intent policies, compiles and distributes policy bundles, and stores the evidence

Next steps

- [Install the client.](#)

- Read [How OpenLatch works](#) for the path of one action, end to end.
- See which agents are wired, plane by plane, in [Control levels](#).

FAQ

Find quick answers here, each with a link to the page that covers it in full.

Is OpenLatch free?

Pricing for the platform is being worked out with our design partners, and design-partner pilots are free.

Do I need Node.js?

Only for `npx @openlatch/client init`, because `npx` is part of Node.js. The client is a native binary and does not run on Node.js. On a machine without Node.js, run the script for your platform; it needs nothing beyond what the operating system ships:

```
curl -LsSf https://openlatch.ai/install.sh | sh
```

```
powershell -ExecutionPolicy Bypass -c "irm https://openlatch.ai/install.ps1 | iex"
```

See [Install the client](#).

Which agents does OpenLatch cover?

Claude Code, Codex CLI, Cursor and Cline are wired today. Coverage is stated per agent and per plane: [Control levels](#) shows which plane is Enforced, Monitored or Observed for each one.

What data does the client send to the platform?

After each decision, the client sends the record of the action and its verdict: what the agent tried to do, such as a command, a file path or a network destination, and what was decided. Credentials are redacted

on the machine before anything leaves it. See [Privacy and telemetry](#).

Can I use OpenLatch offline?

Yes. The client decides on the machine against its resident policy bundle, which keeps deciding with the network unplugged. Fetching new policy and sending records to the platform need a connection; deciding does not.

Does OpenLatch slow my agent down?

The client decides on the machine, from a policy bundle it holds in memory, so the agent never waits on the network for a verdict. Records go to the platform afterwards.

Can I host the platform myself?

Not today. The platform runs as a managed service at `app.openlatch.ai`.

How do I uninstall the client?



```
openlatch uninstall
```

This removes the agent hooks and stops the client. Add `--purge` to also delete the client's state, including the `~/ .openlatch` folder and the credential stored in your keychain.

Neither command removes the binaries: a purge is not an uninstall of the files the install placed. If you installed from a script, `openlatch` and `openlatch-hook` stay in `~/ .local/bin` or `%LOCALAPPDATA%\openlatch\bin`, and so does the `PATH` entry. Delete them by hand if you want the machine exactly as it was.

Where do I report a bug or ask for a feature?

Email us at hello@openlatch.ai.

Concepts

These pages explain the ideas you meet everywhere in OpenLatch: the Autonomy Zone, intent policies, verdicts, modes and anomalies. Read them to understand why the product behaves as it does; the guides tell you what to click.

OpenLatch runs a continuous control loop around each agent. Control exists to make it easier to say yes: the clearer the field an agent plays on, the more it can be trusted to do on its own.

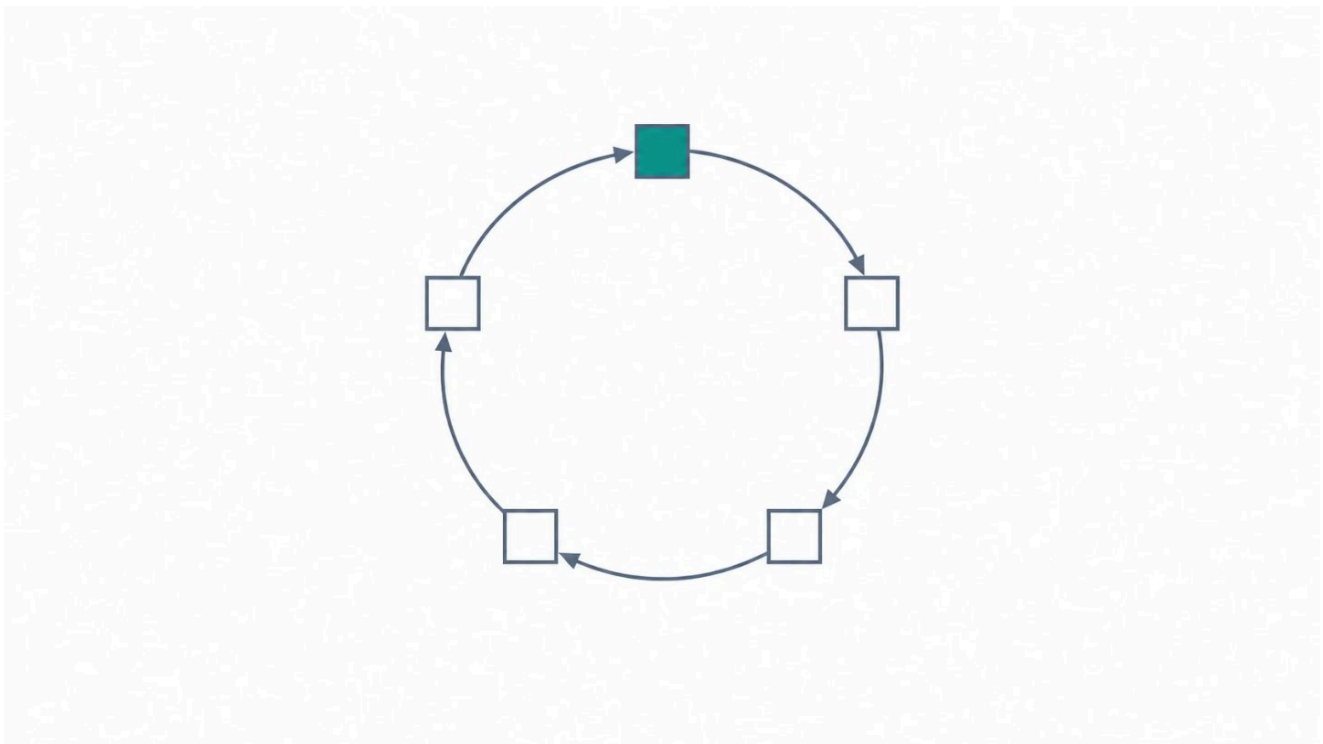


Figure 1. The control loop: Identify, Define, Detect, Act and Improve, around each agent.

Step	What happens	Read
Identify	The agent is connected to your enterprise context: its identity, owner, unit, type and environment, and the control level OpenLatch has over it	Control levels
Define	Each agent resolves to one effective Autonomy Zone, built from intent policies across five dimensions	Autonomy Zone · Dimensions · Intent policies
Detect	What the agent does is captured as an execution graph and checked against its zone. An action that violates an intent policy is an anomaly	Execution graph · Anomalies
Act	Before impact, each covered action gets one verdict: Allow, Ask, Block or Optimize	Verdicts · Modes
Improve	The record of every decision shows which policies are accurate, so you can move them from Monitor to Enforce. OpenLatch never widens an agent's authority on its own	Modes

For the path of a single action from the agent's hook to the verdict, read [How OpenLatch works](#).

Autonomy Zone

An agent's Autonomy Zone is the contextual operating envelope it acts within: what it is trying to achieve, the conditions under which it may act, and the consequences the enterprise accepts. Inside the zone, the agent is free to pursue its mission. Every covered action that would cross the boundary is decided before impact.



Figure 1. An Autonomy Zone is the operating envelope of one agent: what it may do, under which conditions, with which consequences accepted.

The agent brings the mission; the zone brings the rules. Think of a basketball court: the lines and the rules of the game are fixed, the player moves freely inside them, and the referee calls what crosses a line. The zone is the court, its intent policies are the rules, and the verdict is the call.

What a zone holds

A zone holds intent policies across five dimensions: Quality, Economics, Security, Safety and Compliance. [Dimensions](#) explains what each one covers. Each policy runs in Disabled, Monitor or Enforce mode in that zone, as [Modes](#) explains.

Zones follow your organization

Zones sit on your organization's own tree of units, usually synchronized from your directory. You do not maintain a second hierarchy.

Rule	What it means
A zone lives on a unit	Any unit may carry one zone, and most do not
Inheritance is never optional	A unit without a zone inherits the zone of the nearest unit above it that has one. Every zone inherits every zone above it
Add, never loosen	A lower zone can add policy, never remove what it inherits
One agent, one zone	Each agent is assigned to exactly one zone: by default the nearest zone at or above its owner's unit, or one a user sets by hand
No hidden targeting	A policy cannot apply to some agents in its zone and not others. If it should not reach an agent, move the agent or author the policy lower in the tree

The effective zone

What one agent actually operates under is its **effective zone**: every policy of its assigned zone and of every zone above it, grouped by dimension. Each policy names the zone it came from. When the same policy is held at several levels on an agent's path, it reaches the agent once, in the strictest of its modes.

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Agents / Agent / Autonomy Zone

Q

Ctrl+K

?

99+

99+

DC

GENERAL

Overview

Identity

Autonomy Zone

Anomalies

Execution Graph

Economics

CONFIGURATION

MCP

Skills

Rules

Hooks

ARTIFACTS

Domains

Files

Commands

Sessions

ci-runner-payments-01

Claude CodeLinux x64cClient v0.1.1828.171.37.168IDLE

Autonomy Zone

12 intent policies reach this agent from 2 Autonomy Zone layers – none authored on the agent.

AUTONOMY ZONE

Payments

inherits 2 layers: Crispy Pay

POLICIES IN FORCE

12

9 inherited · 3 in its Autonomy Zo...

ENFORCING

6 of 12

the other 6 record, and do not act

ANOMALIES - 14 DAYS

0

nothing flagged

2 Autonomy Zone layers apply here

Assigned to Payments, resolved top-down from the org tree. A layer below may tighten what it inherits.

Crispy Pay

+9151

Payments ASSIGNED HERE

+344

Effective intent policies

INTENT POLICY	ALSO IN	LAYER	MODE	ANOMALIES 14D
Economics 1 policy · 0 enforcing · 1 in Monitor				
AIP-007 A session stays under sixty tool runs	-	↑ Crispy Pay	MONITOR	0

An agent's Autonomy Zone tab: its zone and path, the policies in force, and the zone each one came from.

What a zone is not

Not

Because

A static list of guardrails

The same action can get a different verdict for a different agent, mission or environment

Instructions in the prompt

A prompt shapes behavior and can be forgotten. The zone is enforced at the agent's execution boundary, outside its context window

A permission list

Existing identities and permissions are inputs to the zone, which judges the action in context

An unbypassable sandbox

The threat model is a misbehaving agent, not a malicious operator

Dimensions

An Autonomy Zone is bounded by five dimensions. Each answers one question about an agent’s action, and each is populated with intent policies.

Dimension	The question it answers	Example intent policies
Quality	Is the work up to standard?	Code is tested before it ships. Outputs meet the accuracy and completeness bar. The mandated process is followed
Economics	Is the cost justified?	Budgets and token caps per period. No runaway loops or pointless retries. The model matches the task
Security	Can the agent touch this?	No credential leaves the corporate boundary. Only approved tools, MCP servers and skills. Data access stays within authorization
Safety	Is the consequence acceptable?	No production access without approval. No irreversible or destructive operation. Blast radius and customer impact stay within what the enterprise accepts
Compliance	Is it lawful and by the book?	Data-protection constraints for agents that serve customers in regulated markets. Internal policies and standards. Contractual obligations

A policy can span several dimensions

An intent policy is made of one or more intent atoms, and each atom belongs to a dimension. A policy carries the dimensions of all its atoms, so one policy can be about Security and Safety at once. The console groups an agent’s effective policies by dimension, and every anomaly names the dimension of the policy it violated.

What is not a dimension

- **The mission.** What an agent is for belongs to the agent, not to the zone. The agent brings the mission; the zone brings the rules.
- **Audit evidence.** Compliance rules run in the zone, before the action. The evidence of what happened, which policy applied, which verdict was returned and whether it was enforced is the record OpenLatch keeps of every decision, whatever the dimension.

Intent policies

An intent policy captures what you mean, rather than every surface an agent might use to do it. In OpenLatch you write it as an **Agent Intent Policy (AIP)**, and OpenLatch compiles it into deterministic rules the client applies before each covered action.

From your words to an AIP

You start from a brief: a sentence, a pasted policy document or a recorded conversation. OpenLatch splits it into **intent atoms**, one per rule, and generates two things from each atom: a sentence you can read, and concrete test cases.

Atom field	What it holds	Example
who	The agents the rule is about	any agent
does what	The action	deploy
to what	The target	a production namespace
only if	The condition that permits it	an approved change ticket is linked
unless	The exception	an incident bridge is active
class	Whether the atom can be enforced: enforceable, monitor-only or out of scope	enforceable

You never fill these fields in. OpenLatch extracts them from your brief, and you change them by editing the sentence.

You ratify the sentence

For the atom above, you read this sentence:

While no incident bridge is active, when an agent requests a deployment to production, the system shall require an approved change ticket.

Beside it, the test cases show what the policy would decide in concrete situations, for example a deployment with a ticket, one without a ticket, and one without a ticket during an incident. The cases are there so your answer is informed. You answer Yes or No on the sentence:

- **Yes** accepts the atom.
- **No** opens a note where you say what you meant in your own words. OpenLatch changes the fields, regenerates the sentence and its cases, and marks the cases whose verdict moved. The atom comes back for a fresh Yes or No, because an answer given on the old sentence does not cover the new one.

You never see or approve the compiled rules. You approve what you can read.

From an AIP to the client

Once its atoms are accepted, you ratify the AIP into zero, one or several zones, each in its own mode. The platform compiles the atoms into rules and builds one policy bundle per agent from its effective zone. The bundle is integrity-checked, and the client keeps it on the machine, so it keeps deciding with the network unplugged.

An atom's class matters at runtime: a `monitor-only` atom stays in Monitor even when its policy runs in Enforce, because it can be watched but not enforced.

The policy lifecycle

1. Write a brief or bring a source document.
2. Review the generated sentences and their cases, and accept them.
3. Replay the policy against your recent agent activity to see what it would have decided.
4. Ratify it in Disabled, Monitor or Enforce.
5. Run it in Monitor, read its anomalies, and move it to Enforce when it is accurate.

[Write your first policy](#) walks through these steps in the console.

Verdicts

Before a covered action executes, the client returns one of four verdicts: Allow, Ask, Block or Optimize. A verdict is deterministic because an explicit policy produced it.

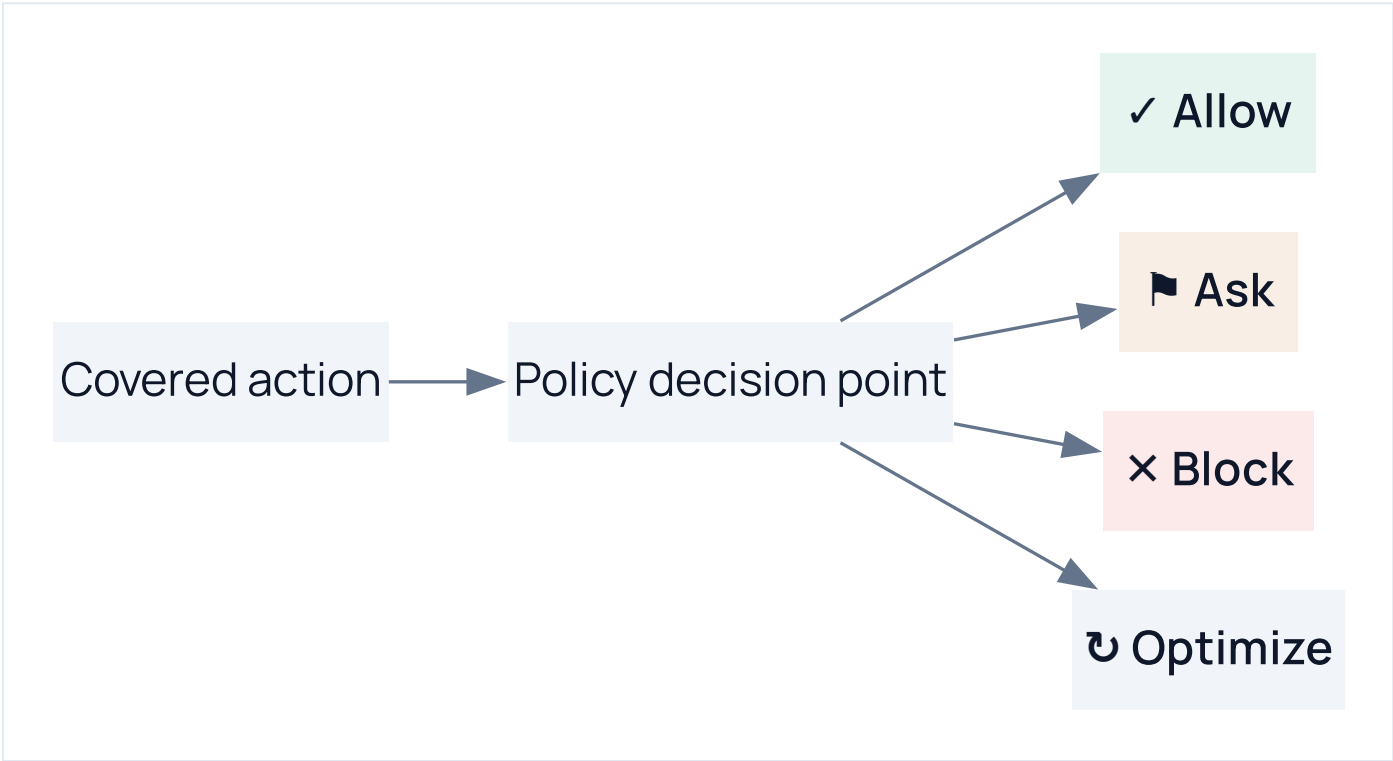


Figure 1. Every covered action reaches the policy decision point before it runs. The effective zone's intent policies produce one deterministic verdict.

Verdict	What happens
Allow	The action runs. In Monitor, it may run and be flagged
Ask	By default the action runs and is flagged for asynchronous review; it holds only where an atom opts in for an irreversible action
Block	The action does not run. The agent receives a native refusal that names the rule that produced it
Optimize	The path changes. In Enforce a lever acts; in Monitor it measures what it would have done

When several policies match one action, the most restrictive verdict wins, in this order: Block, then Ask, then Optimize, then Allow.

Ask

Ask lets the action run and flags it for review, so a person can decide whether the agent should do it again. It prevents recurrence, not the first occurrence. An atom can opt in to hold an irreversible action until it is answered; nothing else waits on a person.

How Ask reaches the agent depends on the agent. Claude Code receives it as its own native ask, carrying the rule’s reason. Codex CLI has no ask of its own, so it receives a message carrying the reason, and the call proceeds.

Optimize

Optimize changes the path of an action instead of stopping it. Each change is made by a lever, and what a lever does depends on the mode of the policy.

Lever	Status
loop_stop	Acts in Enforce; measures in Monitor
steer	Acts in Enforce; measures in Monitor
narrow_output	Acts in Enforce on Claude Code; measures in Monitor
effort_clamp	Measures by default, in Monitor and Enforce alike; acts in Enforce only where you opt in
context_edit	Measures by default, in Monitor and Enforce alike; acts in Enforce only where you opt in
prefix_guard	Measures by default, in Monitor and Enforce alike; acts in Enforce only where you opt in
substitute	This lever is not shipped, and in Monitor it measures only

In Monitor, a lever only measures what it would have done, and the record says so. In Enforce, it is applied.

One action, four verdicts

Illustrative scenario. The same action, `git push`, can earn each verdict depending on its context:

Context	Verdict
An agent pushes a feature branch of a repository its team owns	Allow
An agent pushes to a shared release branch	Ask: the push runs and is flagged for review
An agent force-pushes to the main branch of a production repository	Block
An agent retries the same failing push in a loop	Optimize: in Enforce, <code>loop_stop</code> ends the loop



Note

Ask prevents recurrence, not the first occurrence. On Codex CLI it surfaces as a message and the call proceeds.

Modes

Every intent policy runs in one of three modes in each zone that holds it. The mode decides whether a match is only recorded or whether its verdict is applied before the action runs.

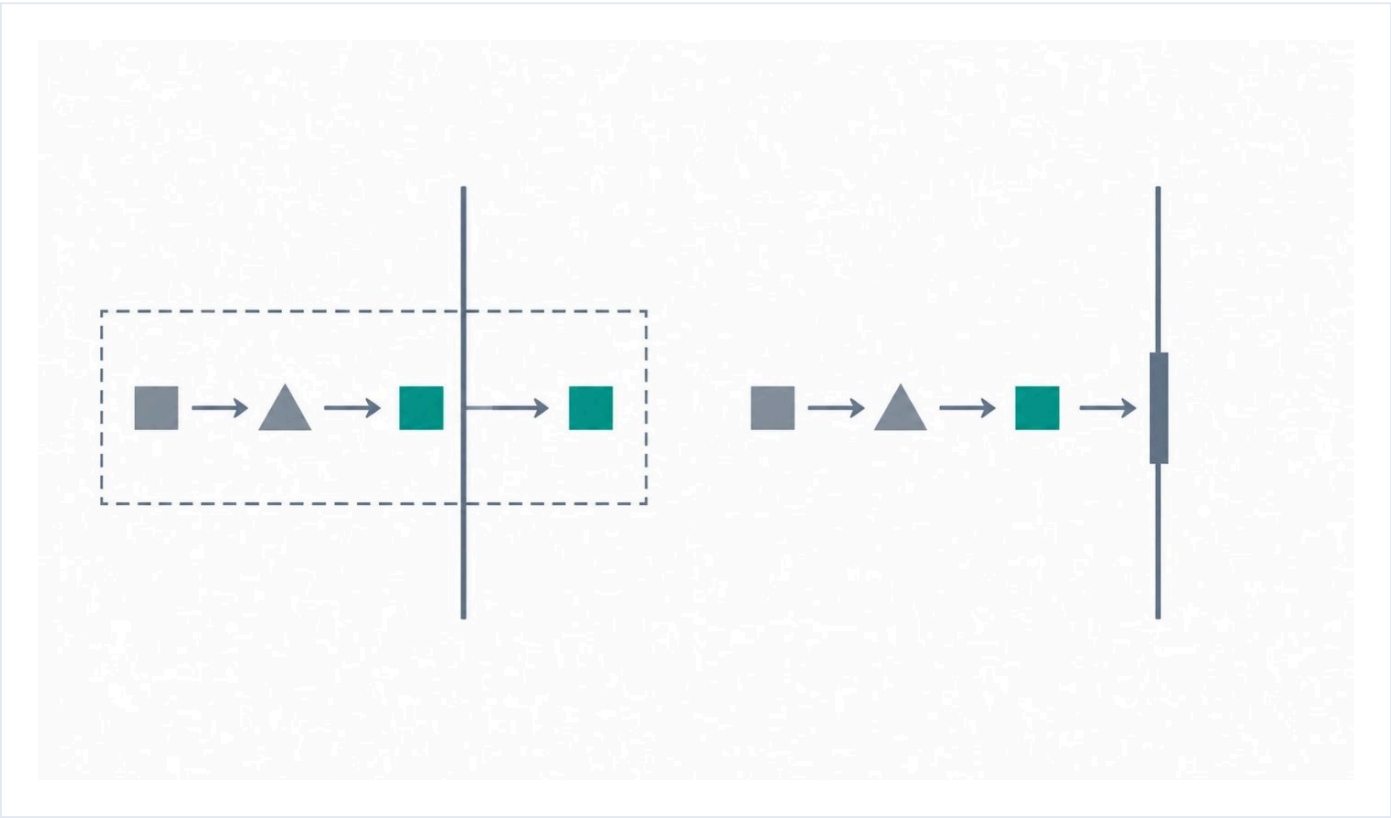


Figure 1. The same actions in Monitor and in Enforce: in Monitor they cross the boundary and are recorded; in Enforce the verdict applies first.

Mode	What it does
Disabled	The policy is kept, with its history, but nothing is evaluated
Monitor	A match is recorded and flagged, and the action proceeds
Enforce	The client applies the policy’s verdict before the action runs, and records whether it did

The mode belongs to the policy in a zone, never to a single atom. One exception is built into the atom itself: an atom whose class is `monitor-only` can be watched but not enforced, so it stays in Monitor even

when its policy runs in Enforce.

Earn Enforce on evidence

Start a new policy in Monitor. It records every action it would have decided, without changing what the agent does. Read its anomalies: when they show the policy says what you meant, switch it to Enforce. When they show it does not, edit it or disable it.

OpenLatch never widens an agent's authority on its own. Every move from Monitor to Enforce, and every new policy, is a choice someone in your organization makes.

One policy, several zones

The same policy can sit in several zones, each with its own mode, for example Monitor in one team and Enforce in another, with no copy to keep in step.

Rule	What it means
Strictest wins	When an agent's path holds the same policy more than once, it runs once, in the strictest mode: Enforce over Monitor over Disabled
A zone only tightens	A policy is switched on in a zone only if it does not loosen the zones above it. Holding it Disabled is always allowed
Lowering is never blocked	Moving a policy from Enforce to Monitor, or to Disabled, is always allowed
One version everywhere	Every zone runs the policy's current version, each in its own mode

Anomalies

An **anomaly** is an agent action that OpenLatch determines violates an intent policy of one of the dimensions of the agent's Autonomy Zone. It is the one thing OpenLatch flags.

Every anomaly names the policy it violated, that policy's dimension, and the zone the policy came from. It is explainable by construction: a rule compiled from a policy your organization ratified produced it, and nothing else did.

What an anomaly does in each mode

Policy mode	What happens
Disabled	Nothing is evaluated, so no anomaly is raised
Monitor	The anomaly is recorded and flagged, and the action proceeds
Enforce	The action receives its verdict before it executes, and the anomaly records it

Where you see them

The **Anomalies** page of the console lists every anomaly in your organization. Each agent also has an **Anomalies** tab with only its own.

app.openlatch.ai/anomalies

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Anomalies

Ctrl+K

?

99+

DC

Anomalies

Review actions that crossed an intent policy, then archive the evidence when triage is complete.

Export to CSV

Waiting on your answer

All

WAITING ON YOUR ANSWER

0

nothing waiting

ANOMALIES - 7 DAYS

26

across 10 agents

BLOCKED

50

ALLOWED AND FLAGGED

50

Nothing is waiting on your answer

Every flagged action has an answer against it, so the policy knows what to do next time.

The Anomalies page, listing the actions that violated an intent policy, with the policy, its dimension and the verdict.

What anomalies are not

Anomalies come only from intent policies. OpenLatch does not flag an action because it differs from what an agent usually does: if no ratified policy covers an action, it raises no anomaly. To flag something new, write the intent policy that says so.

Execution graph

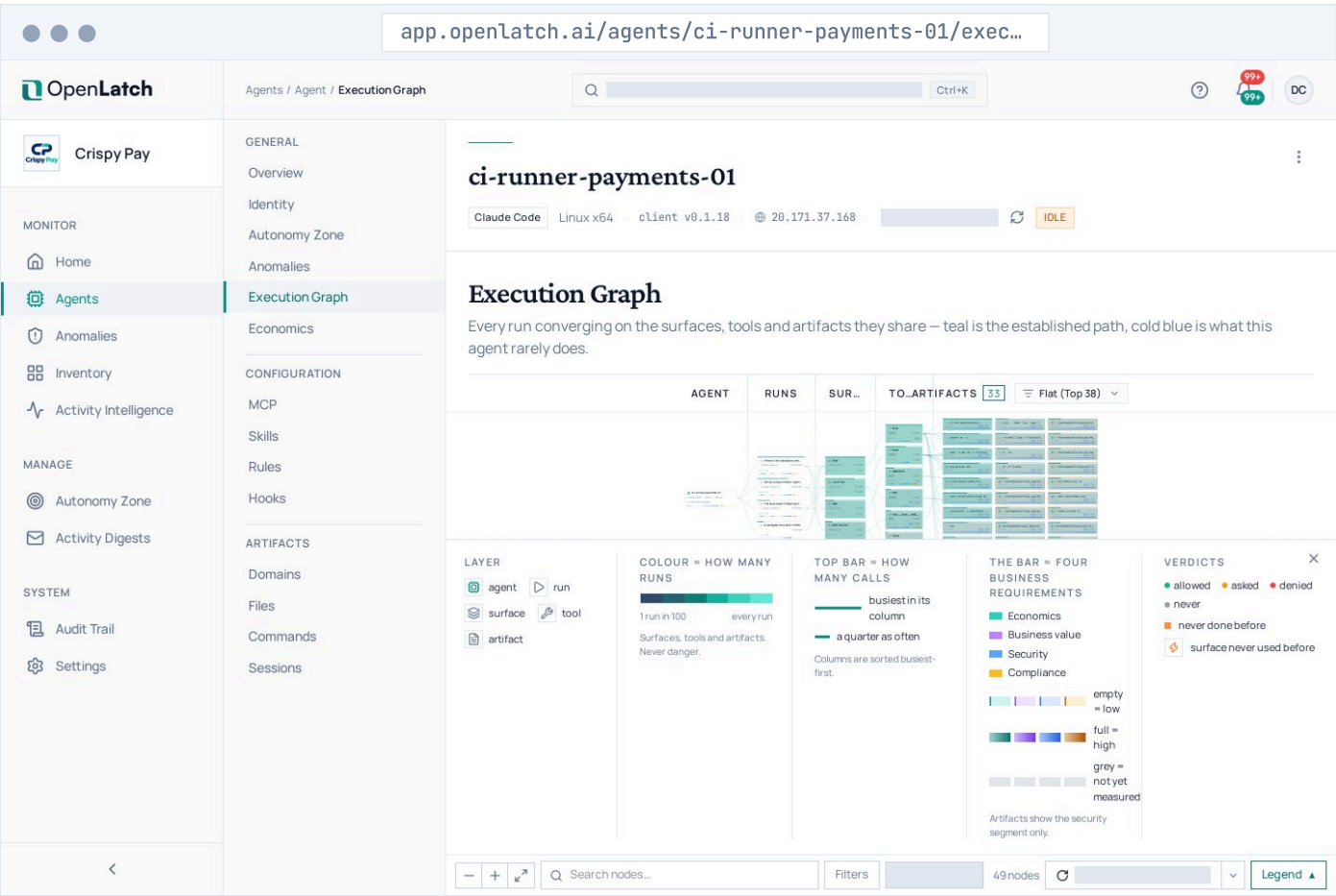
The execution graph is the record of what an agent actually did. It is built from the events the client sends after each decision, and it is what OpenLatch checks against the agent's Autonomy Zone.

What it shows

Each agent has an **Execution Graph** tab. It reads from left to right, from the agent to what it touched:

Lane	What it holds
Agent	The agent the graph is about
Runs	Its sessions, one card per run
Surfaces	What it reached for, derived from the tool names
Tools	What it invoked
Artifacts	What it touched or ran: files, network domains and commands

The lanes converge on what the runs share, so you see the paths an agent takes most and the ones it takes rarely.



An agent's Execution Graph tab: runs, surfaces, tools and artifacts, from left to right.

Two planes feed it

Plane	What the client captures
Command plane	Each tool call and shell command, caught at the agent's native hook before it runs
Request plane	Each outbound model request, caught at a listener on the same machine, with its model, its tokens and its cost

The other tabs of an agent break the same record down by kind: commands, files, domains, MCP servers, skills, hooks, sessions and economics.

From an action to a policy

When the graph shows a command you want to govern, open it and start a new intent policy from that action. The brief starts from what the agent actually ran, and the policy goes through the same review and ratification as any other. [Intent policies](#) explains that path.

An action that no policy decided is recorded as allowed by default, not as permitted by a decision. The graph tells the two apart.

Control levels

Every agent in the agent record carries a control level. It tells you what OpenLatch can actually do with that agent, so you never have to assume an agent is controlled because it is listed.

Control level	What it means
Enforced	The client sits in the agent's execution path. A verdict reaches the agent and is obeyed
Monitored	The integration is installed and events flow, but a verdict cannot be delivered yet, so nothing is prevented
Known	The agent is identified and classified, with the limits of control stated

Coverage is a deployment outcome. The control level is what the deployment achieved for that agent, not a promise about every agent of its kind.

Two planes

An agent is covered on two planes, and the control level is always stated per plane.

Plane	What it catches	What it can carry today
Command plane	A tool call or shell command, caught at the agent's native hook	Allow, Ask, Block or Optimize, decided before the action runs. Optimize is Monitor-first
Request plane	An outbound model request, caught at a listener on the same machine	Allow and record by default, so it is Observed: captured and measured. Ask has no meaning here, and Block is not shipped on this plane

Agents wired today

Agent	Delivery	Command plane	Request plane
Claude Code	Any	Enforced on the command plane	Observed on the request plane
Codex CLI	Managed, through MDM, a system setting or a cloud setup	Enforced on the command plane, and the developer cannot turn it off	Observed on the request plane
Codex CLI	Self-serve	Monitored on the command plane until the developer trusts the OpenLatch hook inside Codex CLI, then Enforced on that plane	Observed on the request plane
Cursor	Self-serve, in the IDE and the CLI, on macOS and Windows	Enforced on the command plane	Observed on the request plane
Cline	Self-serve, in VS Code, on macOS	Monitored on the command plane	Observed on the request plane

`openlatch status` and `openlatch doctor` report each agent on the machine with its control level.

How each agent receives a verdict

Not every agent can express every verdict natively. A verdict an agent cannot express is delivered as the nearest one it can, recorded as such, and never stricter than the rule intended.

Agent	Ask	Block
Claude Code	Its own native ask	Its native refusal
Codex CLI	A message carrying the rule's reason; the call proceeds	Its native refusal
Cursor	Native on shell commands and MCP calls in some of the IDE's run modes; elsewhere the action is allowed and the record says Ask could not be delivered	Its native refusal

How OpenLatch works

This page follows one action from the moment the agent attempts it to the moment its record reaches the platform. Policy is written once, on the platform. Each decision is made on the machine, by the client, with no network call.

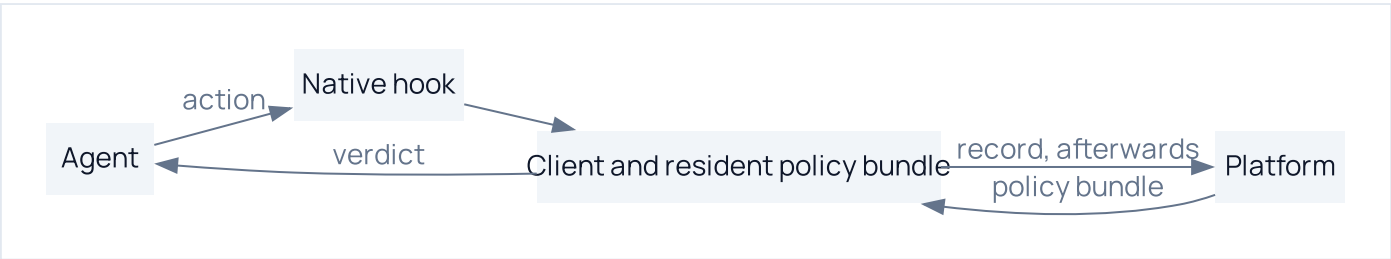


Figure 1. The platform compiles and distributes policy. The client decides each covered action locally, then sends its record to the platform.

1. Policy reaches the client

You ratify intent policies in the console. The platform compiles them and builds one policy bundle for each agent, from that agent’s effective zone. The client asks the platform for its bundle at regular intervals:

The platform answers	The client
Nothing changed	Keeps the bundle it holds
A new bundle	Checks its integrity, then swaps it in whole
Nothing, or a bundle that fails the check	Keeps the last valid bundle

To fetch the bundle at once, run `openlatch system aip refresh`.

2. The hook catches the action

The client plugs into the agent’s own native hooks. When the agent attempts a covered action, such as running a shell command or calling a tool, the hook passes it to the client before it executes. Outbound

model requests pass through a listener on the same machine.

3. The client decides

The client evaluates the action against its resident bundle, in memory, on the machine. There is no network call on this path, so the agent never waits on the network and the verdict never depends on it. When several policies match, the most restrictive verdict wins. [Verdicts](#) and [Modes](#) explain what each verdict does in Monitor and in Enforce.

4. The record follows

After the decision, the client redacts credentials on the machine and sends the record of the action and its verdict to the platform. The platform stores that record for the audit trail, the execution graph and reporting. It never decides again and never changes a verdict the client already returned.

Offline

The resident bundle is the authority, not a cache. With the network unplugged, the client keeps applying the last valid bundle, indefinitely. Fetching new policy and sending records need a connection; deciding does not.

Two situations leave an action undecided, and the action then proceeds: the machine has never held a valid bundle, or the hook cannot reach the client on its own machine. `openlatch doctor` reports both.

Privacy and telemetry

This page explains what leaves your machine and our websites, and how to turn off what you do not want sent.

What the client sends to the platform

After each decision, the client sends the platform the record of the action and its verdict: what the agent tried to do, such as a command, a file path, a tool or a network destination, and what was decided. That record is what the console shows in the execution graph, the anomalies and the audit trail.

- Credentials are redacted on the machine before anything leaves it.
- The client also keeps a local audit log on the machine.
- The decision itself never depends on the platform: see [How OpenLatch works](#).

Usage telemetry from the client

The client sends usage telemetry and crash reports to help us improve it. Until the client is signed in, usage telemetry carries only a random identifier of the installation. Once it is signed in, the client links that identifier to your account and organization identifiers, so its usage telemetry is no longer anonymous. Both go to the platform's own address, so they need no extra destination in your network rules. You can turn either off.

To	Do this
See whether usage telemetry is on, and why	<code>openlatch system telemetry status</code>
Turn usage telemetry off	<code>openlatch system telemetry disable</code> , or set <code>OPENLATCH_TELEMETRY_DISABLED=1</code> or <code>DO_NOT_TRACK=1</code>
Turn usage telemetry back on	<code>openlatch system telemetry enable</code>
Turn crash reports off	Set <code>OPENLATCH_CRASH_REPORTING=0</code>

Analytics on `openlatch.ai` and these docs

`openlatch.ai` and `docs.openlatch.ai` have no sign-in and load no third-party trackers. They record anonymous product analytics through [PostHog](#), hosted in the EU, through a first-party path on `openlatch.ai`.

We record:

- the pages you visit and how long you stay;
- conversion events, such as copying an install command or following a link to the console;
- Core Web Vitals, so we can keep the sites fast;
- a sample of session replays, with all text and form inputs masked;
- unhandled JavaScript errors, after a scrubber in your browser replaces anything shaped like an email address or an API token.

We never collect what you type in a form, we never identify you from these sites, and we do not sell or share this data.

Opt out

Add `?ph_opt_out=1` to any page address to turn analytics off on this device. The choice is stored in your browser and survives reloads. You can also block the `/ingest/` path with any content blocker.

Moving on to the console

When you follow a link from these sites to `app.openlatch.ai`, the anonymous identifier travels in the address fragment, after the #, so the console can join your earlier visits to your account once you sign up. The fragment never reaches our servers.

Guides

Each guide covers one task in the console at `app.openlatch.ai`, with the page you do it on. If you are new, start with [Write your first policy](#); to understand why the console behaves as it does, read the [Concepts](#).

Policies and zones

Guide	Console pages
Write an Agent Intent Policy	Autonomy Zone > Policies, New intent policy
Graduate a policy to Enforce	Autonomy Zone > Policies
Manage Autonomy Zones	Autonomy Zone, its Agents and Audit tabs
Review anomalies	Anomalies

Agents

Guide	Console pages
Find and read an agent	Agents , and an agent's Overview , Identity and Autonomy Zone tabs
Investigate what an agent did	An agent's Sessions , Execution graph , Commands , Files , Domains and Anomalies tabs
Review an agent's configuration	An agent's Hooks , MCP servers , Rules and Skills tabs
Track an agent's spend	An agent's Economics tab
Search the inventory	Inventory and its tabs
Follow live events	Events
Read the Home page	Home

Reporting

Guide	Console pages
Schedule activity digests	Digests
Set up notification channels	Settings > Notification Channels
Review the audit trail	Audit Trail , Settings > Audit Trail

Organization settings

Guide

Console pages

[Manage users and teams](#)**Settings > Users, Settings > Teams**

[Set up single sign-on](#)**Settings > SAML SSO, Settings > Auth Providers**

[Connect a directory for owner lookup](#)**Settings > Identity Providers**

[Edit the organization profile](#)**Settings > Organization Profile**

[Tune the catalogue](#)**Settings > Catalogue**

[Read licensing and usage](#)**Settings > Licensing**

Your account

Guide

Console pages

[Manage API keys](#)**Account > API Keys**

[Manage your account](#)**Account > Profile, Account > Security**

Deploying OpenLatch across an estate is its own section: [Deploy](#).

Write an Agent Intent Policy

Write an Agent Intent Policy (AIP) when you want an Autonomy Zone to allow, question or stop a kind of agent action. You need the editor role or higher. If the wizard says authoring is unavailable, an administrator of your OpenLatch site has not configured AI authoring yet.

An AIP is a set of intent atoms. From them OpenLatch generates one sentence and concrete test cases. You ratify the sentence, Yes or No, with the test cases shown beside it; you never edit the compiled rules. [Intent policies](#) explains the model.

Create the policy

- 1 Open **Autonomy Zone**, select the unit whose zone will hold the policy, then open **Policies** and choose **New intent policy**.
- 2 **Brief**. Say what you want in one or two sentences, in your own words, or start from an example.
- 3 **Policy**. Read the sentence OpenLatch generated, with the test cases beside it. Each case is a concrete situation and the verdict the policy would return. Answer Yes if the sentence says what you meant. Answer No and say what you meant instead, and OpenLatch regenerates the sentence and its cases.
- 4 **How it runs**. Choose the mode the policy is ratified in: Disabled, Monitor or Enforce. Start in Monitor unless you already trust the policy. Where a rule can ask a person, choose what happens until the question is answered: by default the action runs and is flagged for review; for an irreversible action you can opt in, per atom, to stop it until someone answers.
- 5 **Ratify**. Run the replay, which applies the policy to your recent agent activity and shows what it would have decided, then ratify. Nothing is enforced until you do.

app.openlatch.ai/zone/policies/new

OpenLatch

Autonomy Zone / New intent policy

Q

Ctrl+K

?

99+

99+

DC

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

New intent policy

Turn what you already have into an intent policy. Nothing runs until you ratify it.

1

 Brief

2

 Policy

3

 How it runs

4

 Ratify

Define the policy intent

Provide a brief or reference material. OpenLatch will use it to draft an Agent Intent Policy for your review.

Paste a brief

Type or paste what you mean, in your own words.

Upload a policy

A document, an SOP, a procedure. We read it and pull the intents out.

Record a conversation

Describe the rule out loud, or attach the meeting where it was agreed.

Your brief

Examples

e.g. Payments engineering change-control SOP:

- No production deployment or database migration without an approved change ticket.

- During declared incidents, the on-call engineer may act without a ticket, recording actions on the incident bridge.

- Destructive commands (DROP TABLE, rm -rf, kubectl delete namespace) are strictly forbidden on production clusters.

- All access to customer secrets or payment credentials must require an active session.

One rule per line. Anything we infer is marked as inferred, on the line itself.

Write what the agent does, not what it intends

Name the exact action: command, path, domain or tool (e.g. "Block Bash when command.program equals rm").

Say the outcome: block it, or ask a human before it runs.

Phrase exceptions as a fact the agent shows: "unless agent.environment equals development".

This policy will be created in Crispy Pay.

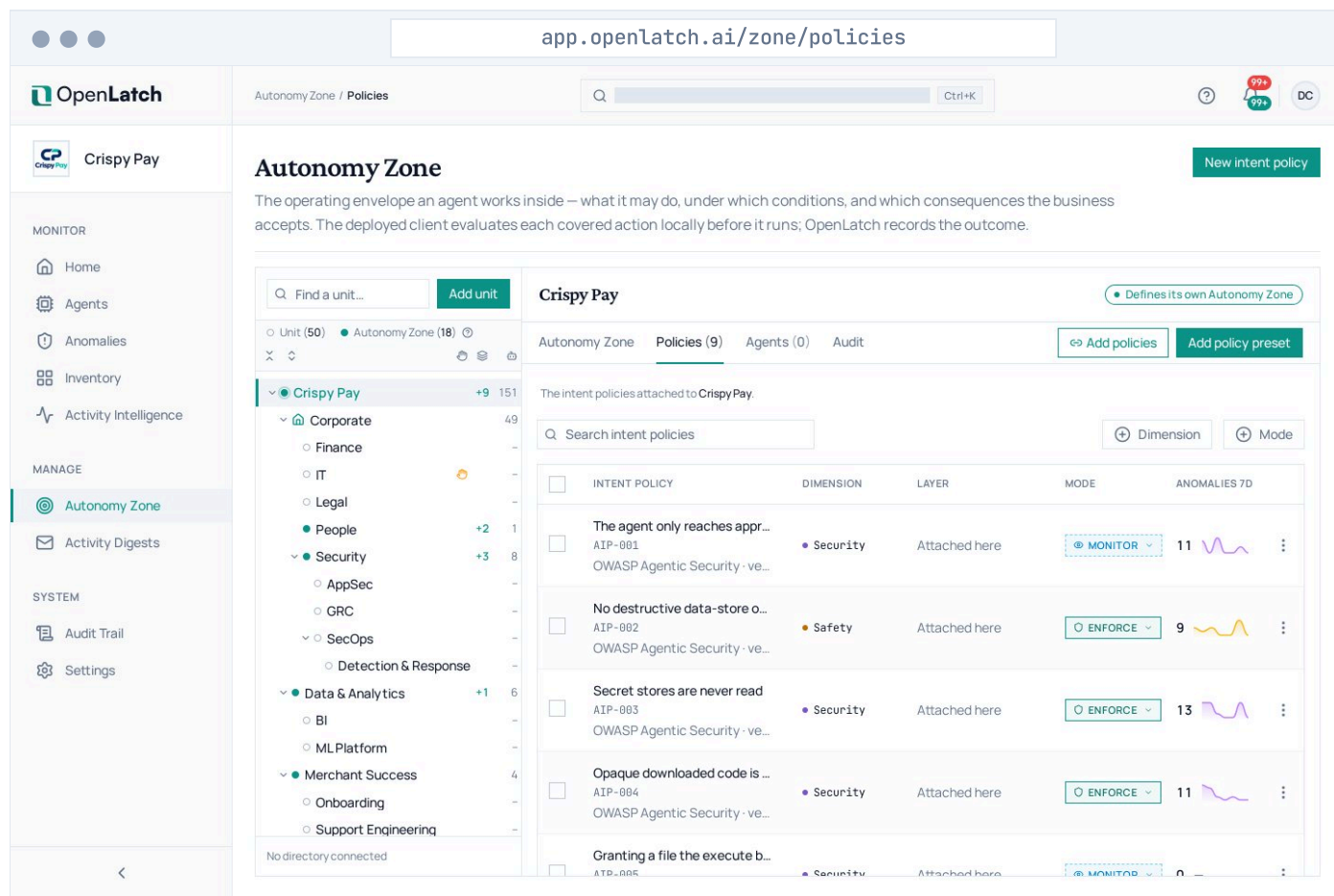
Turn it into a policy

The New intent policy wizard: Brief, Policy, How it runs, Ratify.

Ratifying rebuilds the policy bundle. Each client fetches it on its next poll; to fetch it at once on a machine, run `openlatch system aip refresh`.

Manage policies in the list

The **Policies** tab lists the intent policies authored in the selected unit, with their dimensions, layer, mode and the anomalies each raised over the last seven days. Turn on **Show inherited policies** to see the policies a unit inherits from the layers above it.



The Policies tab: each intent policy with its dimension, layer, mode and recent anomalies.

To	Do this
Read a policy	Open it. The drawer shows its intents, the generated test cases, its activity, its provenance and its history of ratified versions
Change its mode	Use Mode in the row's actions, or select several policies and choose Enforce , Monitor or Disable
Change an inherited policy	Open the layer that authored it. Bulk actions apply only to policies authored in the selected unit
Start from presets	Choose Add policy preset , review each preset policy, choose the mode each one should run in, and apply them
Delete a policy	Choose Delete . The policy retires for this layer and every layer beneath it, and its history stays in the zone's audit

A policy whose intents are all monitor-only can run in Monitor but not in Enforce: Monitor records what happens, and Enforce needs at least one enforceable intent.

Let coverage tighten a policy

OpenLatch learns which tools, programs and targets your agents' actions involve, and can propose changes that keep a policy covering what it was written for. The policy's **Coverage mode** decides what happens to those proposals:

Coverage mode	What happens
Off	Coverage is not analysed for this policy
Strict	Learning is counted and shown; nothing is proposed or applied
Explicit	Proposals wait for your review
Auto	Restrictive changes and catalogue extensions apply automatically, and each is written to the policy's history

Review proposals from the policy's **Coverage** section, approve or dismiss them, then apply them, which ratifies a new version. [Tune the catalogue](#) covers the entries coverage learns.

Next, [graduate the policy to Enforce](#) once its record shows it is accurate.

Graduate a policy to Enforce

Run a policy in Monitor until its record shows that it decides what you expect, then switch it to Enforce. You need the editor role or higher, on the unit that authored the policy.

In Monitor, a policy records its anomalies and never intervenes. In Enforce, the client applies its verdict before the action runs. [Modes](#) explains the difference.



Figure 1. Crispy Pay’s policies over their first six weeks: most graduate from Monitor to Enforce once their verdicts are trusted. Illustrative demo data.

Decide when to graduate

Graduate a policy when its record over a representative period, a week of normal work for most teams, holds no surprise:

Check	Where
Every anomaly it raised is an action you want it to act on	The policy's Activity , then Anomalies filtered to the policy
It raised no anomaly on work you consider normal	The same list, read with the agent owners
Its intents reach the enforce bar	The policy drawer. An intent marked monitor-only raises anomalies and never intervenes, in any mode

When the record shows a false positive, change the policy instead of graduating it: open it, answer No to its sentence with what you meant, and ratify the new version in Monitor.

Switch the mode

- 1 Open **Autonomy Zone**, select the unit that authored the policy, and open **Policies**.
- 2 In the policy's row, open its actions and set **Mode** to **Enforce**. To graduate several policies at once, select them and choose **Enforce**.
- 3 Each client applies the change on its next poll. To apply it at once on a machine, run `openlatch system aip refresh`.

From then on, the client decides the policy's enforceable intents before the action runs: Allow, Ask, Block or Optimize. An Optimize lever acts only in Enforce; in Monitor it measures what it would have done. Each policy's mode is recorded in the audit trail with who changed it.

Step back

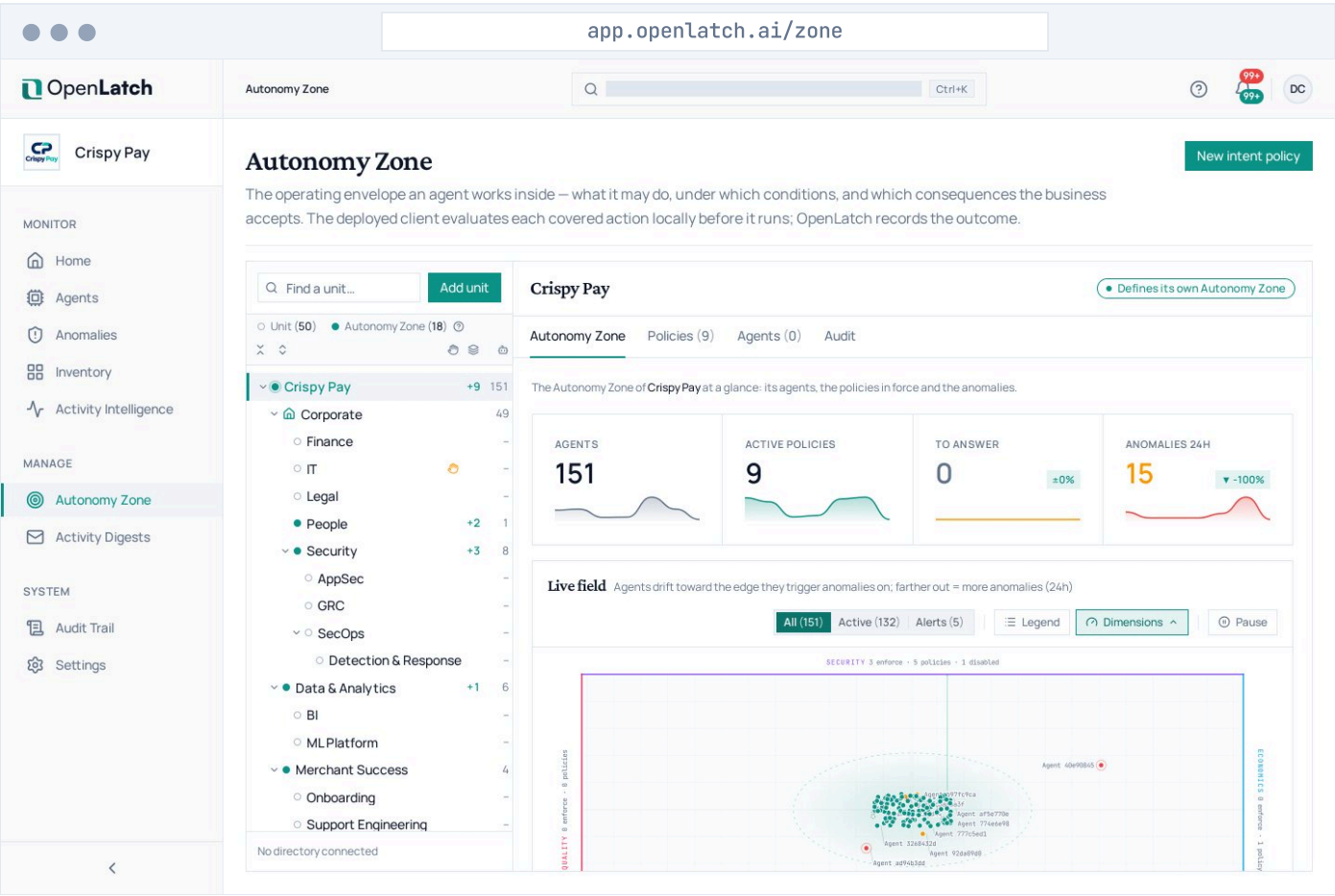
To step a policy back, set its mode to **Monitor**. It keeps recording its anomalies and stops intervening on the next poll. Set it to **Disabled** to stop evaluating it altogether; its history stays available.

Manage Autonomy Zones

Use the **Autonomy Zone** page to decide which parts of your organization carry their own rules, which agents sit where, and what each zone decided. You need the editor role or higher to change a zone; anyone can read it.

Autonomy Zones sit on your organization's tree of units. Any unit may carry a zone, and most do not: a unit without one inherits the nearest zone above it, and each agent resolves to one effective zone.

[Autonomy Zone](#) explains the model.



The Autonomy Zone page: the organization's tree of units, and the zone of the selected unit at a glance.

Read a zone

Select a unit in the tree. The page shows its zone at a glance: its agents, the intent policies in force and the anomalies they raised. The tabs go further:

Tab	What it shows
Policies	The intent policies authored in the unit, and on request the ones it inherits. See Write an Agent Intent Policy .
Agents	The agents the unit governs directly
Audit	Every action an agent in the unit attempted, with the rule it matched and the decision

app.openlatch.ai/zone/agents

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Autonomy Zone / Agents

Q Go to Organization Profile Ctrl+K

99+ 99+ DC

Autonomy Zone

The operating envelope an agent works inside — what it may do, under which conditions, and which consequences the business accepts. The deployed client evaluates each covered action locally before it runs; OpenLatch records the outcome.

Q Find a unit... Add unit

Unit (50)

Autonomy Zone (18)

Corporate

Finance

IT

Legal

People

Security

AppSec

GRC

SecOps

Detection & Response

Data & Analytics

BI

ML Platform

Merchant Success

Onboarding

Support Engineering

+9 151

49

-

-

-

+2 1

+3 8

-

-

-

+1 6

-

-

4

-

-

No directory connected

Crispy Pay

Defines its own Autonomy Zone

Autonomy Zone Policies (9) Agents (0) Audit

Assign agents

The agents Crispy Pay governs directly. 151 more sit in Autonomy Zones beneath.

Q Search agent, owner or platform...

Control level

Assignment

Platform

No agent is assigned to this Autonomy Zone.

The Agents tab: the agents a unit governs directly.

The screenshot shows the OpenLATCH interface at the URL `app.openlatch.ai/zone/audit`. The left sidebar contains navigation links: **MONITOR** (Home, Agents, Anomalies, Inventory, Activity Intelligence) and **MANAGE** (Autonomy Zone, Activity Digests). The **SYSTEM** section includes Audit Trail and Settings. The main content area is titled **Autonomy Zone** and includes a description: "The operating envelope an agent works inside – what it may do, under which conditions, and which consequences the business accepts. The deployed client evaluates each covered action locally before it runs; OpenLATCH records the outcome." A **New intent policy** button is visible. Below this, the **Crispy Pay** unit is selected, showing a tree view of its structure (Corporate, Finance, IT, Legal, People, Security, AppSec, GRC, SecOps, Detection & Response, Data & Analytics, BI, ML Platform, Merchant Success, Onboarding, Support Engineering). The **Audit** tab is active, displaying a summary of actions: **ALLOWED** (0), **ASKED A HUMAN** (0), **BLOCKED** (0), and **MONITORED** (0). A table below the summary is empty, with a message: "No runtime decisions in this window. Decisions appear here after a client reports evaluated activity for this Autonomy Zone."

The Audit tab: each action an agent in the unit attempted, with the rule it matched and the decision.

Give a unit its own zone

- 1 Select the unit and open its settings.
- 2 Turn on its own Autonomy Zone. The zone starts empty and behaves exactly like the zone above it until you author a policy in it: every unit inherits every layer above it either way.
- 3 Author the policies that apply only here. See [Write an Agent Intent Policy](#).

To remove a unit's own zone, use **Delete Autonomy Zone** in the unit's settings. The policies authored in it retire, and its agents move to the zone above.

Add or change a unit

Units usually come from your directory, through [Connect a directory for owner lookup](#). A directory unit's name and place in the tree belong to the directory and are overwritten on the next sync; its Autonomy

Zone is yours.

Add a unit by hand for a part of the business the directory does not model, such as a plant line or a joint venture. A unit added by hand is never overwritten by a sync, and it is the only kind of unit you can delete. Deleting it moves its children and its agents to its parent. The tree goes up to ten levels deep.

Place agents

By default an agent sits in its owner’s Autonomy Zone, from the owner’s unit in the directory. To place agents elsewhere, open the zone’s **Agents** tab and choose **Assign agents**: the drawer moves agents into this zone, or hands them back to their owner’s zone. An agent assigned by hand stays where you put it until you return it; **Remove manual assignment** lets the directory decide its zone again.

The agent’s own **Autonomy Zone** tab shows the effective zone it resolves to, layer by layer. See [Find and read an agent](#).

Choose where an Ask reaches a person

ASK delivery sets where an Ask from the zone’s intent policies reaches a person:

Setting	Where the Ask appears
Platform	On the Anomalies page
Agent	In the agent’s own prompt, where the agent can ask; elsewhere as Platform
Both	In the agent’s prompt and on the Anomalies page

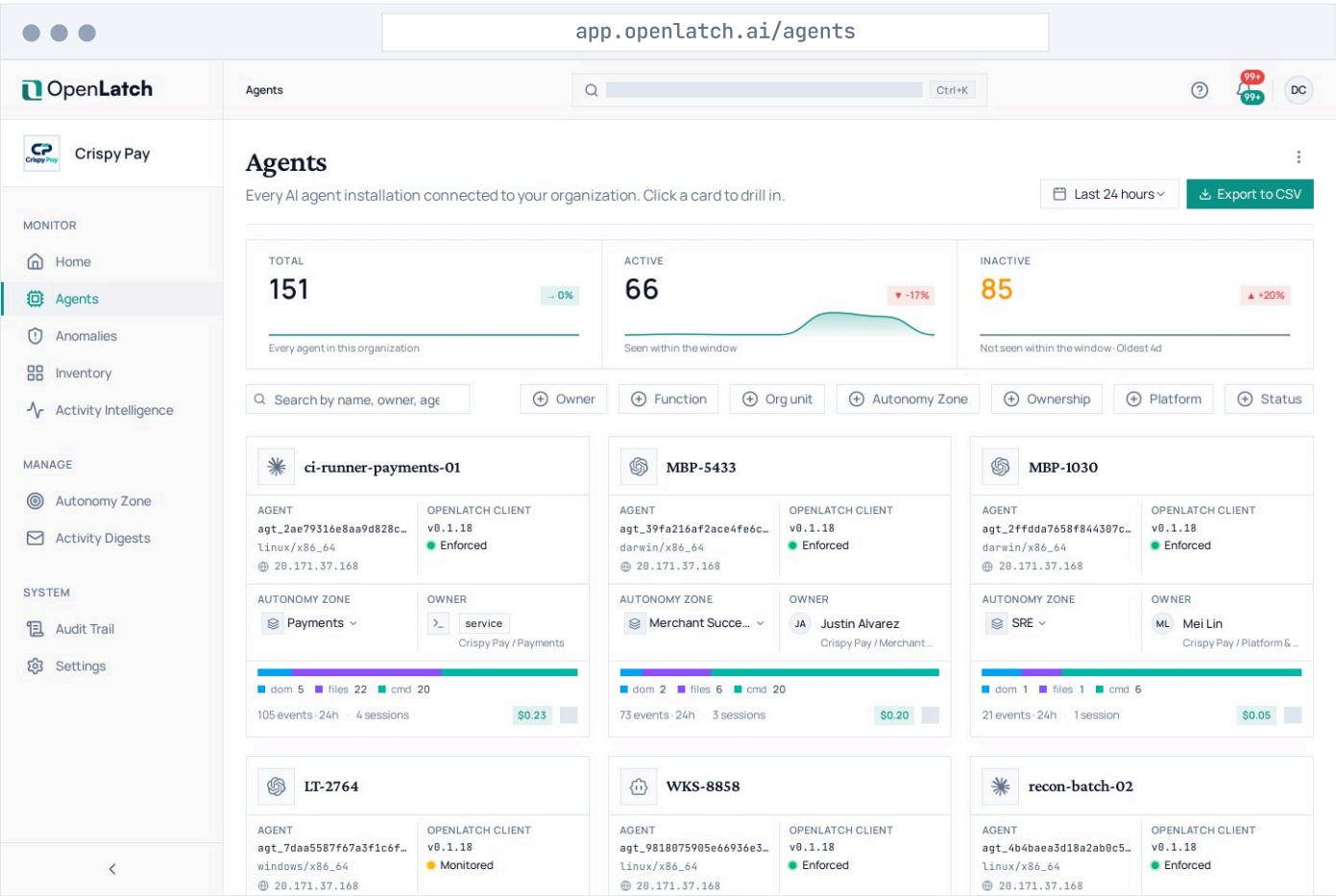
Child zones inherit the setting unless they set their own. Agents that cannot ask in their own prompt are listed beside the setting. By default an Ask lets the action run and flags it for review. See [Verdicts](#).

Find and read an agent

Use the **Agents** page to find any agent that has reported to your organization, and its tabs to see who it acts for and which rules it runs under. Every role can read it.

Find an agent

The **Agents** page lists every agent in the organization. The figures at the top count all agents, the agents seen within the selected window, and the agents not seen within it. Search by name, owner, platform or host, and filter the list to narrow it.



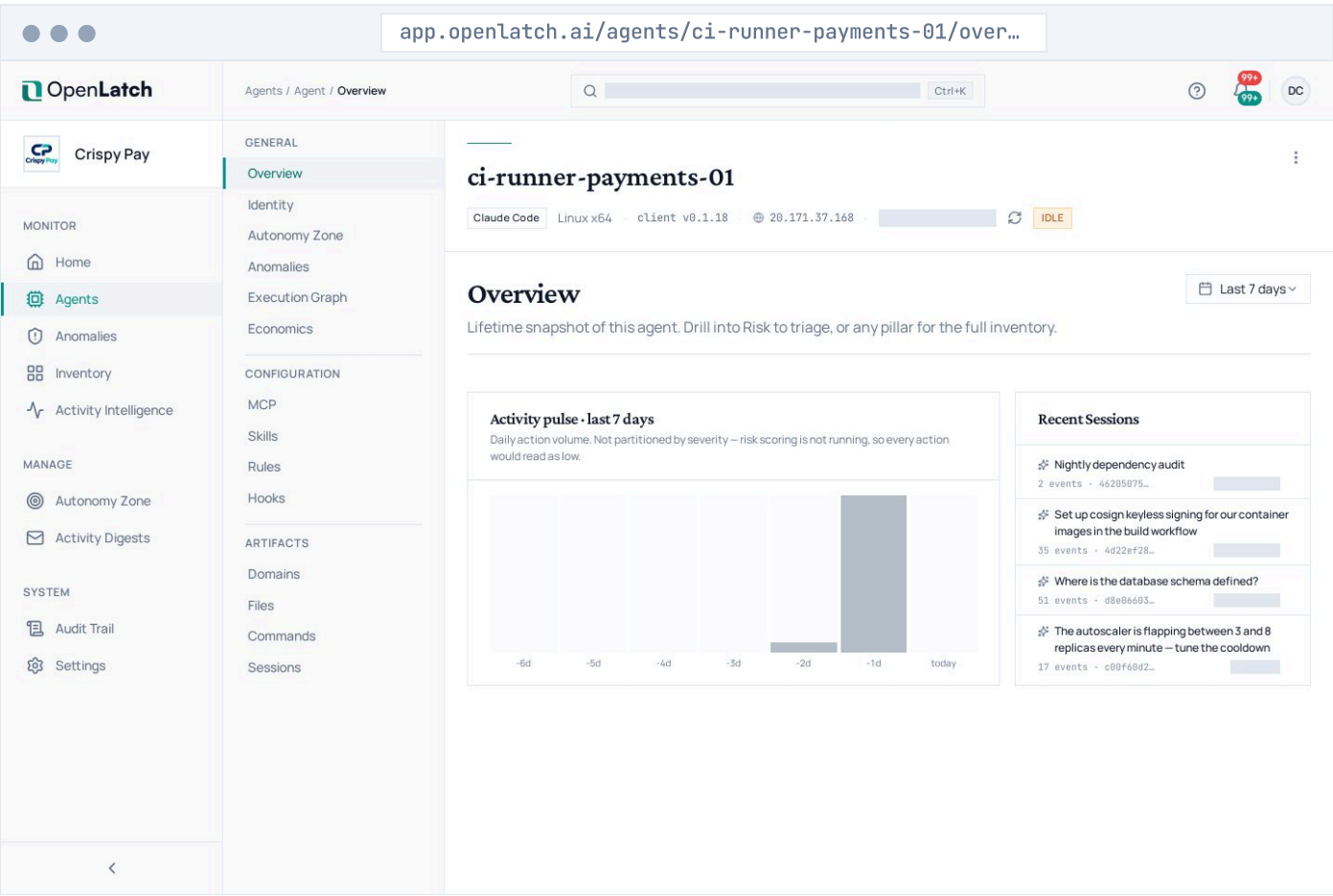
The Agents page: every agent in the organization, with its owner, control level and last activity.

Each agent carries a [control level](#): Enforced, Monitored or Known. It tells you what OpenLatch can do with that agent, so you never assume an agent is controlled because it is listed.

To connect a new agent, choose **Install an agent** and run the command it shows on the machine. [Install the client](#) explains the commands.

Read the overview

Open an agent. Its **Overview** tab summarizes what it did over the selected window: its actions, the elevated ones among them, the endpoints it reached, and the anomalies it raised.

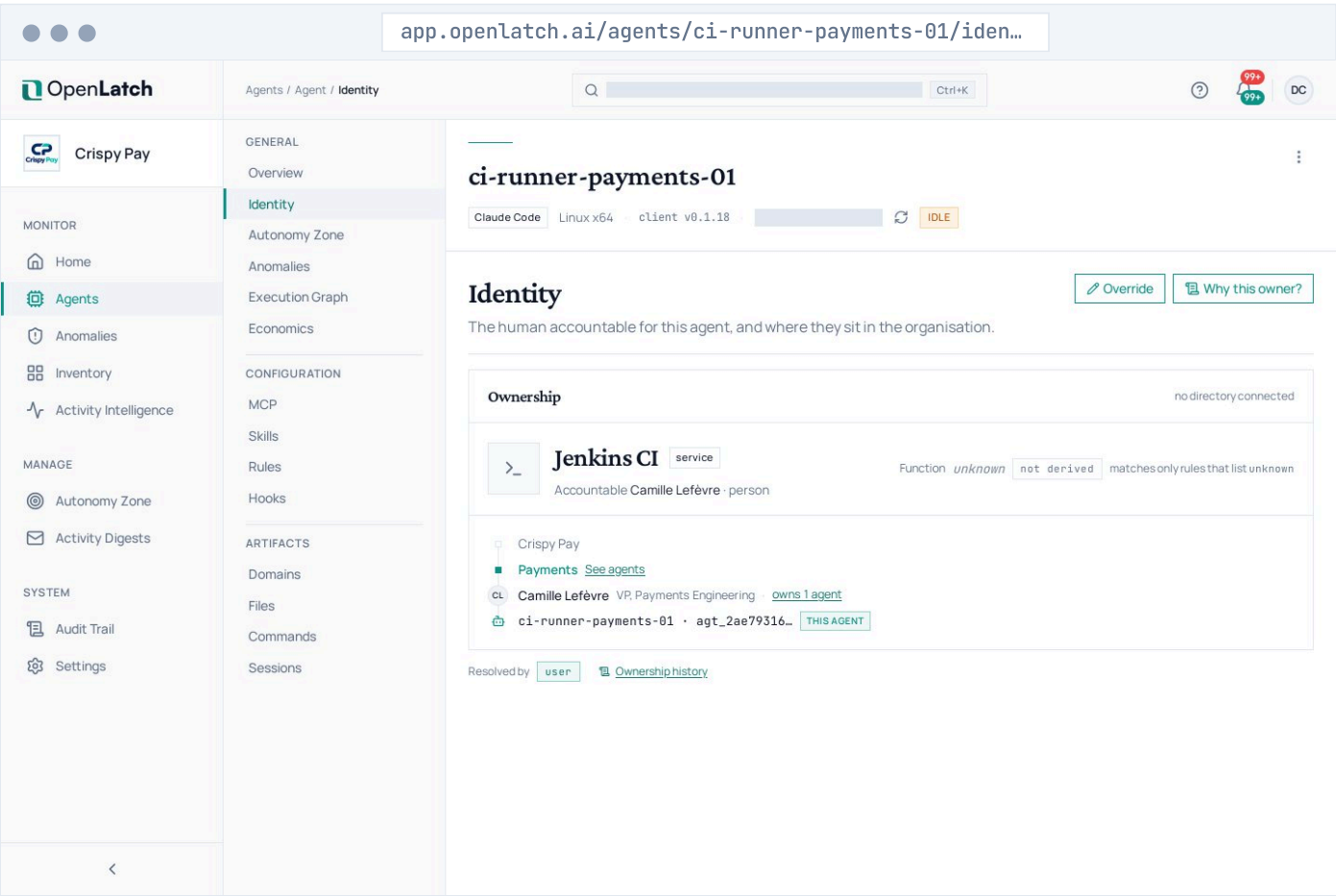


An agent's Overview tab: its activity over the window and what needs a look.

The other tabs are covered by [Investigate what an agent did](#), [Review an agent's configuration](#) and [Track an agent's spend](#).

Check who owns it

The **Identity** tab names the person accountable for the agent and where they sit in the organization. With a directory connected, the owner is resolved through a read-only lookup; see [Connect a directory for owner lookup](#).



An agent's Identity tab: its owner, and where they sit in the organization.

- 1 Choose **Why this owner?** to see how the owner was resolved, from the signals the machine reported to the directory record they matched.
- 2 If the owner is wrong, choose **Override ownership** and pick the right person. If that person is not in the directory, add them by hand with their email address and, optionally, their name and job title. For an agent that acts for a service rather than a person, name the service and the person accountable for it.

Overriding ownership needs the editor role or higher, and it is recorded in the audit trail.

See its Autonomy Zone

The **Autonomy Zone** tab shows the effective zone the agent resolves to: the layers it inherits, from the organization down to its own unit, and the intent policies in force on it. An agent at the Known control level is listed with its zone, but its zone is not enforced on it.

OpenLatch

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Agents / Agent / Autonomy Zone

Q

Ctrl+K

?

99+

99+

DC

Crispy Pay

GENERAL

Overview

Identity

Autonomy Zone

Anomalies

Execution Graph

Economics

CONFIGURATION

MCP

Skills

Rules

Hooks

ARTIFACTS

Domains

Files

Commands

Sessions

ci-runner-payments-01

Claude CodeLinux x64cClient v0.1.1820.171.37.168IDLE

Autonomy Zone

12 intent policies reach this agent from 2 Autonomy Zone layers – none authored on the agent.

AUTONOMY ZONE

Payments

inherits 2 layers - Crispy Pay

POLICIES IN FORCE

12

9 inherited - 3 in its Autonomy Zo...

ENFORCING

6 of 12

the other 6 record, and do not act

ANOMALIES - 14 DAYS

0

nothing flagged

2 Autonomy Zone layers apply here

Assigned to Payments, resolved top-down from the org tree. A layer below may tighten what it inherits.

Crispy Pay

Payments

Effective intent policies

INTENT POLICY	ALSO IN	LAYER	MODE	ANOMALIES 14D
Economics 1 policy - 0 enforcing - 1 in Monitor				
AIP-007 A session stays under sixty tool runs	-	Crispy Pay	MONITOR	0

An agent's Autonomy Zone tab: the layers it inherits and the intent policies in force.

To move an agent to another zone, see [Manage Autonomy Zones](#).

Remove an agent

Delete agent, in the agent's actions menu, permanently deletes the agent with every session and event recorded for it; the history cannot be recovered. If the client on that machine sends another event, the agent comes back with a new ID. Remove the client from the machine first if the agent should stay gone; see [Operate](#). The same menu exports the agent as JSON.

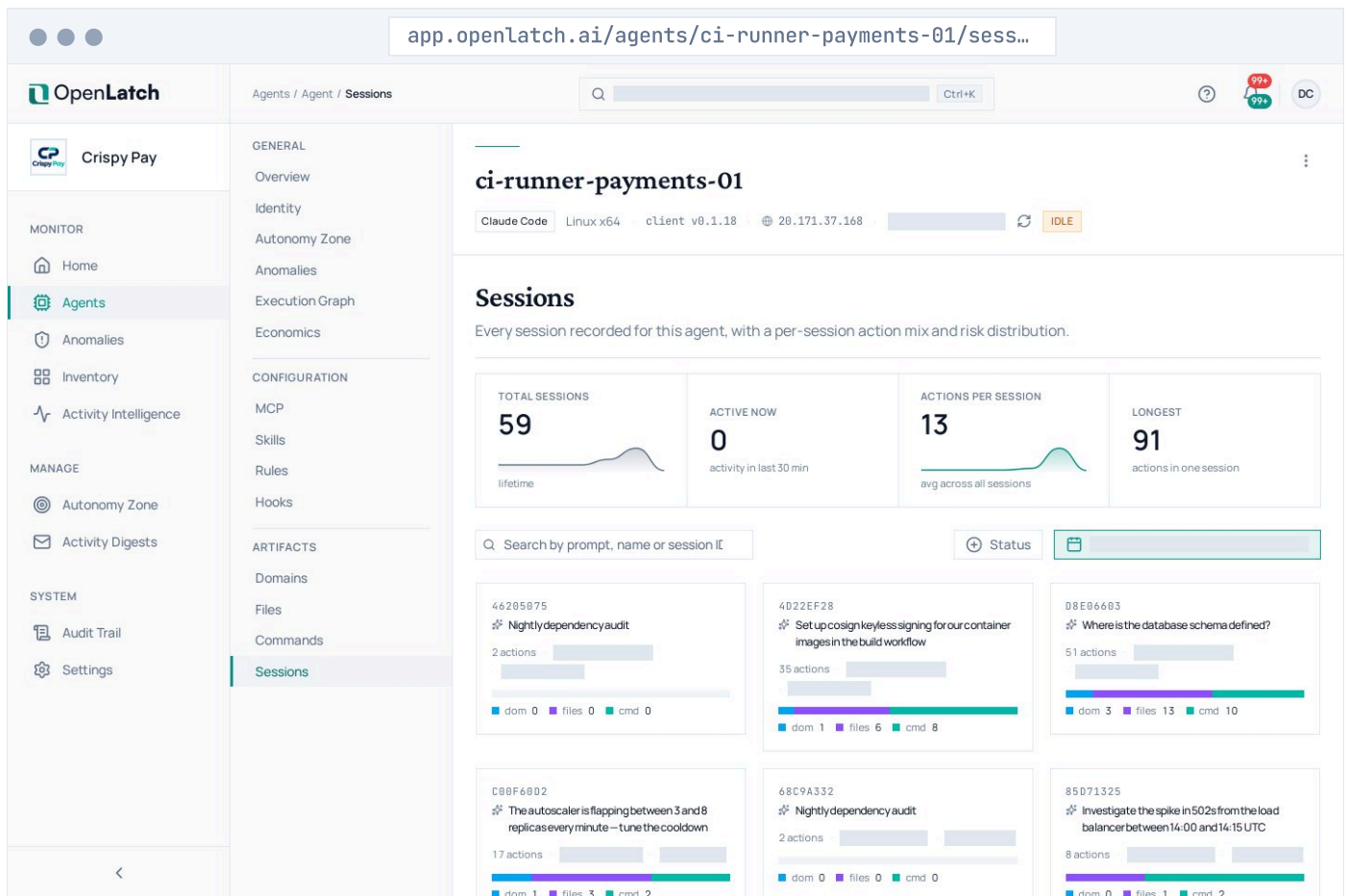
Investigate what an agent did

Use an agent's activity tabs to answer “what did this agent do, and why was it decided that way?”. Open the agent from the **Agents** page, set the time window at the top, then pick the tab that matches your question. Every role can read them.

Your question	Tab
What did it do in one working session?	Sessions
How do its runs, tools and the things it touched fit together?	Execution graph
Which shell commands did it run?	Commands
Which files did it read, write or delete?	Files
Which websites did it reach?	Domains
Which of its actions violated an intent policy?	Anomalies

Sessions

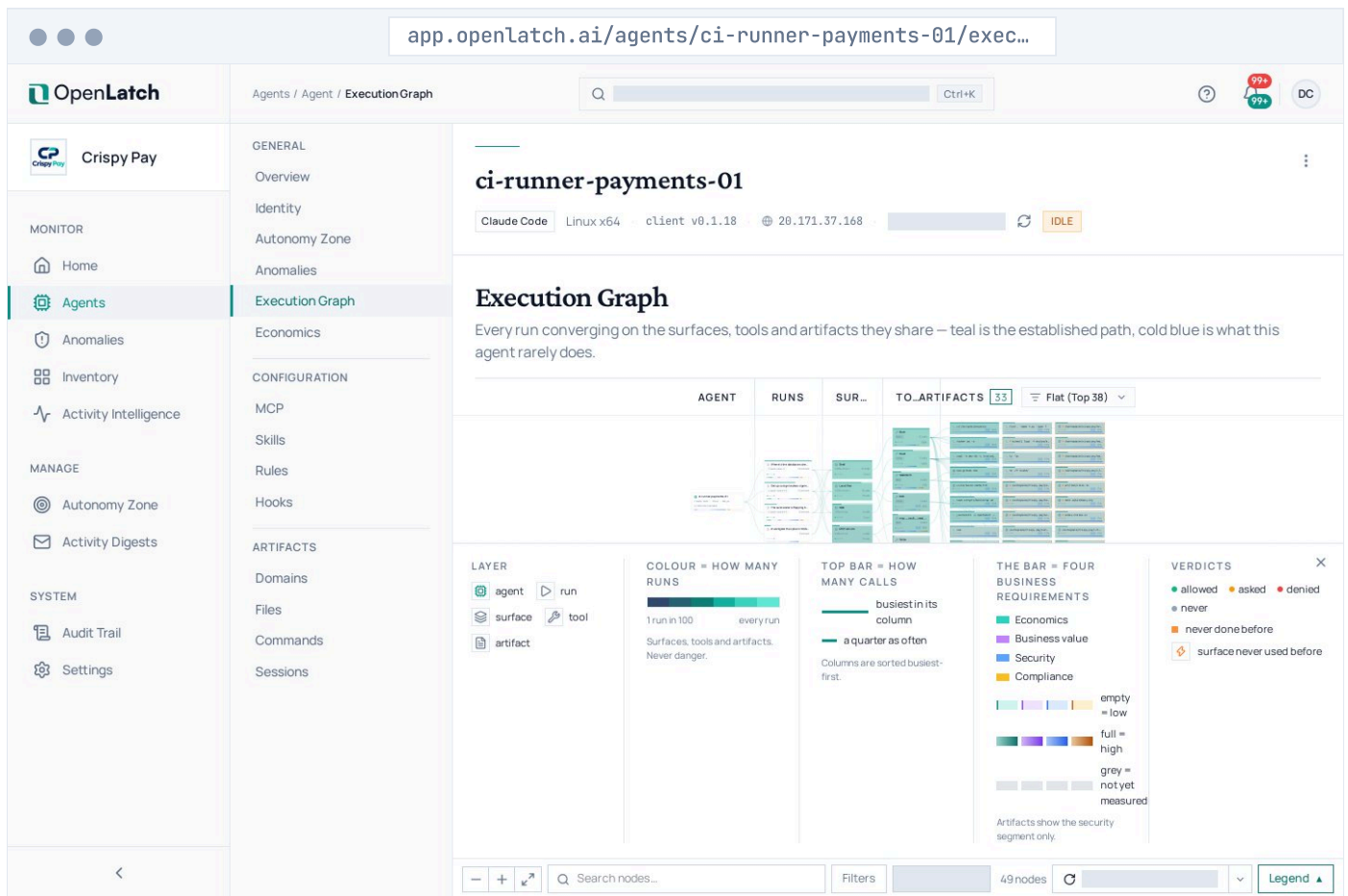
The **Sessions** tab lists the agent’s working sessions, newest first. Open one to read its actions in order, each with the verdict it received and the rule that decided it.



An agent's Sessions tab: each working session, with its actions in order.

Execution graph

The **Execution graph** tab draws the agent's runs over the window, converging on the surfaces, tools and artifacts they share: the agent, then its runs, then what each run reached for, what it invoked, and the files, hosts and commands it touched or ran. The colour of a path shows how often it occurs in the window. Select a node to read the actions behind it. [Execution graph](#) explains the model.



An agent's Execution graph tab: runs converging on the surfaces, tools and artifacts they share.

Commands, files and domains

These three tabs list each kind of action the agent took, with figures over the window at the top. Select a row to open its detail, with every occurrence and the decision each received.

Tab	What it lists
Commands	Every shell command the agent ran, parsed into its program and arguments
Files	Every file the agent read, wrote or deleted, with reads of credential files and writes to configuration and source code counted apart
Domains	Every website the agent reached, with the domains it reached for the first time

app.openlatch.ai/agents/ci-runner-payments-01/comm...

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Agents / Agent / Commands

Q

Ctrl+K

?

99+

99+

DC

GENERAL

Overview

Identity

Autonomy Zone

Anomalies

Execution Graph

Economics

CONFIGURATION

MCP

Skills

Rules

Hooks

ARTIFACTS

Domains

Files

Commands

Sessions

ci-runner-payments-01

Claude Code

Linux x64

cClient v0.1.18

28.171.37.168

↻

IDLE

Commands

Every shell command this agent ran, ordered by time and scored per occurrence.

TOTAL COMMANDS

152

15 unique binaries

Q Search by command text...

Binary

TIME ↓	BINARY ↑↓	COMMAND ↑↓
	cd	cd /scripts/projects
	cd	cd /src/tmp
	psql	psql -h dev-db -c truncate table scratch_events
	find	find . -name *.py -type f
	bash	bash scripts/bootstrap.sh

5 of 152 commands

<

1

2

...

7

>

An agent's Commands tab: every shell command it ran.

app.openlatch.ai/agents/ci-runner-payments-01/files

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Agents / Agent / Files

GENERAL

Overview

Identity

Autonomy Zone

Anomalies

Execution Graph

Economics

CONFIGURATION

MCP

Skills

Rules

Hooks

ARTIFACTS

Domains

Files

Commands

Sessions

ci-runner-payments-01

Claude CodeLinux x64cClient v0.1.1828.171.37.168IDLE

Files

Every unique file this agent touched, classified by intent and ordered by risk.

UNIQUE PATHS5853 actions across all paths

SECRET FILES READ0credential-risk reads

CONFIGS MODIFIED4configuration writes / edits

SOURCE FILES EDITED11source-code writes / edits

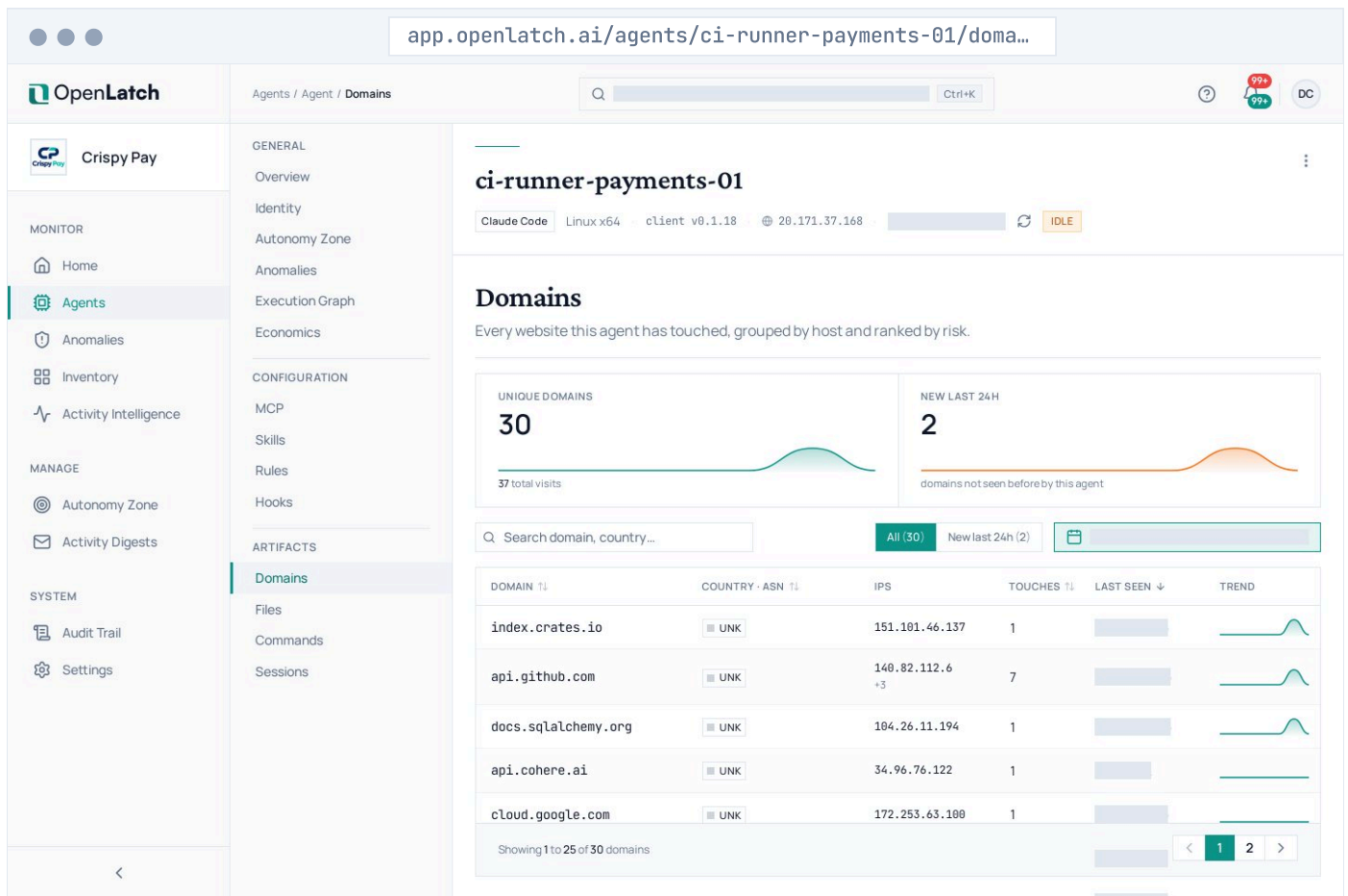
Q Search by path...Intent: All (58)ActionLast 24 Hours

NAME ↕	INTENT ↕	ACTION MIX	TOUCHES ↕	LAST SEEN ↕
OrganizationContext.tsx /workspace/crispy-pay/webapp/src...	SOURCE		1	
biome.json /workspace/crispy-pay	CONFIG		1	
t1l kv/ci/build-cache	DOCS		12	
.gitconfig	DOCS		1	

Showing 1 to 25 of 58 files

<123>

An agent's Files tab: every file it read, wrote or deleted.



An agent's Domains tab: every website it reached.

Command text and file paths are masked on the machine before they are sent, so a credential never appears here. See [Data handling and security](#).

Anomalies

The **Anomalies** tab lists only this agent's anomalies: its actions that violated an intent policy of its zone, each with the policy, its dimension and the verdict. [Review anomalies](#) covers how to work through them across the organization.

The screenshot displays the OpenLATCH web application interface. The browser address bar shows `app.openlatch.ai/agents/ci-runner-payments-01/anom...`. The interface includes a sidebar with navigation menus for 'MONITOR' (Home, Agents, Anomalies, Inventory, Activity Intelligence) and 'MANAGE' (Autonomy Zone, Activity Digests). The 'Agents' menu is currently selected. The main content area is titled 'ci-runner-payments-01' and shows the 'Anomalies' tab. It features a summary section with four metrics: 'WAITING ON YOUR ANSWER' (0), 'ANOMALIES - 7 DAYS' (0), 'BLOCKED' (0), and 'ALLOWED AND FLAGGED' (0). Below this, a message states 'Nothing is waiting on your answer' with a green checkmark icon, explaining that every flagged action has an answer against it.

Waiting on your answer	All
0 nothing waiting	0 across 0 agents

Nothing is waiting on your answer
Every flagged action has an answer against it, so the policy knows what to do next time.

An agent's Anomalies tab: its actions that violated an intent policy.

Turn a finding into a policy

When an action shows something you want decided differently next time, write the intent policy that says so: see [Write an Agent Intent Policy](#). OpenLATCH raises an anomaly only where a ratified policy covers the action.

Review an agent's configuration

Use an agent's configuration tabs to see what the agent is set up to do on its machine, before it does it: the hooks it declares, the MCP servers it connects to, the rules files it reads and the skills it can run. The client reports this configuration from the machine, so a change made there shows up here. Every role can read these tabs.

Open the agent from the **Agents** page and pick a tab.

Tab	What it lists	Figures at the top
Hooks	The lifecycle hooks the agent declares, including the ones that connect it to the client	Hook blocks, their handlers, and the sources changed
MCP servers	The servers connected to the agent's MCP host, with the tools each one exposes	Servers, tools across them, and the sources changed
Rules	The rules files and agent memory found on the host	Files, the sources changed, and the largest file
Skills	Every skill and slash command the agent can run	Skills, and the sources changed

Select an entry to open its detail: where it was declared, its content, and when it last changed.

app.openlatch.ai/agents/ci-runner-payments-01/hooks

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Agents / Agent / Hooks

Q

Ctrl+K

?

99+

99+

DC

GENERAL

Overview

Identity

Autonomy Zone

Anomalies

Execution Graph

Economics

CONFIGURATION

MCP

Skills

Rules

Hooks

ARTIFACTS

Domains

Files

Commands

Sessions

ci-runner-payments-01

Claude CodeLinux x64cClient v0.1.18

IDLE

Hooks

Lifecycle hooks declared by this agent.

TOTAL HOOK BLOCKS

4

Across this host

HANDLERS

7

Across all blocks

MODIFIED 7D

4

Sources mutated

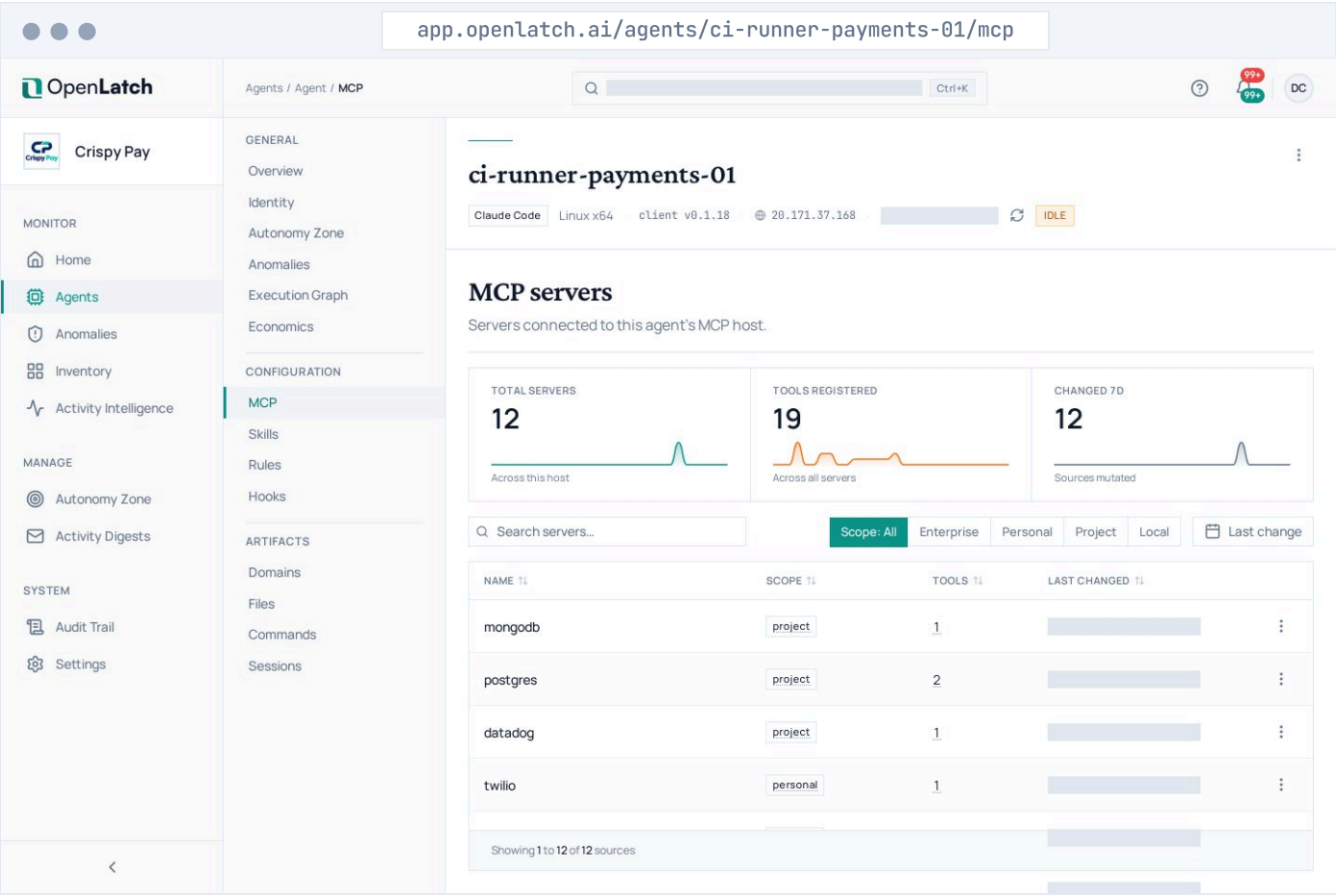
Q Search lifecycle hooks...

LifecycleLast change

NAME	LIFECYCLE	HANDLERS	LAST CHANGED
Claude Code hooks	PreToolUsePostToolUse+1	3	
Claude Code hooks	PostToolUse	1	
Claude Code hooks	PostToolUse	1	
Claude Code hooks	StopSessionStart	2	

Showing 1 to 4 of 4 sources

An agent's Hooks tab: the lifecycle hooks it declares.



An agent's MCP servers tab: the servers connected to its MCP host and their tools.

app.openlatch.ai/agents/ci-runner-payments-01/rules

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Agents / Agent / Rules

GENERAL

Overview

Identity

Autonomy Zone

Anomalies

Execution Graph

Economics

CONFIGURATION

MCP

Skills

Rules

Hooks

ARTIFACTS

Domains

Files

Commands

Sessions

ci-runner-payments-01

Claude CodeLinux x64cClient v0.1.1828.171.37.168IDLE

Rules files

Rules files and agent memory observed on this host.

TOTAL FILES6

LARGEST FILE340

MODIFIED 7D6

CLAUDE / AGENTS / memory

Body characters

Sources mutated

Q Search files...

Last change

NAME	PATH	LAST CHANGED
MEMORY.md	MEMORY.md	
AGENTS.md	AGENTS.md	
CLAUDE.md	CLAUDE.md	
CONTRIBUTING.md	CONTRIBUTING.md	

Showing 1 to 6 of 6 sources

An agent's Rules tab: the rules files and memory found on its host.

The screenshot shows the OpenLATCH interface for agent `ci-runner-payments-01`. The sidebar on the left contains navigation links for **MONITOR** (Home, Agents, Anomalies, Inventory, Activity Intelligence), **MANAGE** (Autonomy Zone, Activity Digests), and **SYSTEM** (Audit Trail, Settings). The main panel displays the **Skills** tab for the selected agent. It shows a summary of 8 total skills and 8 sources mutated. Below this is a table of skills:

NAME ↕	LAST CHANGED ↕
log-analyzer	
i18n-extractor	
changelog-writer	
frontend-design	

The table indicates that 1 to 8 of 8 sources are shown.

An agent's Skills tab: every skill and slash command it can run.

What to look for

Check

Why

The hook that connects the agent to the client is present

Without it, the agent's actions do not reach the decision point.
Run `openlatch doctor` on the machine if it is missing

An MCP server or skill you did not expect

A new server or skill gives the agent new tools to act with. Write an intent policy if its use should be decided

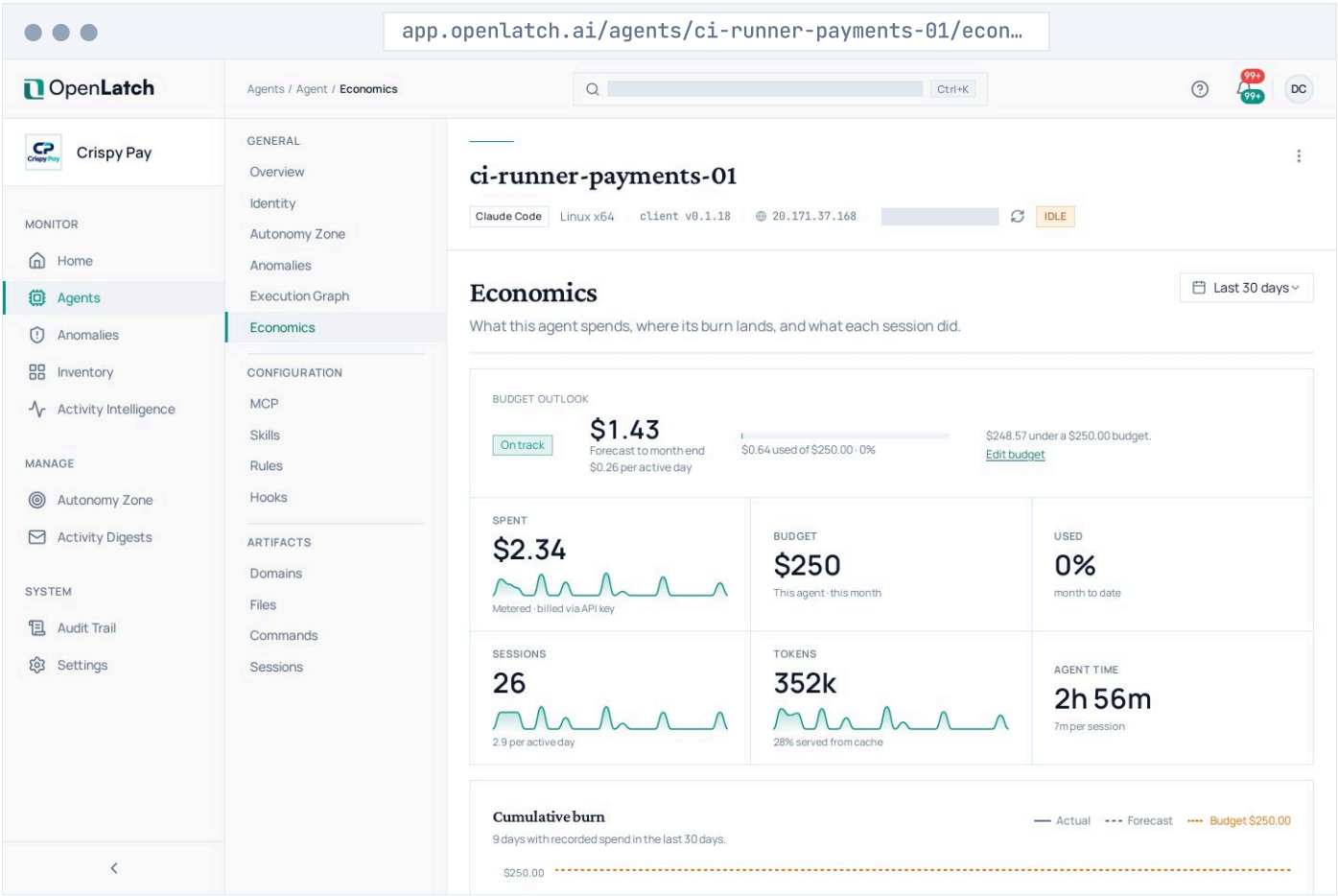
A source that changed recently

A change to a rules file or a hook changes how the agent behaves from its next session

An empty tab means the agent has declared nothing of that kind yet. To see the same configuration across every agent at once, use the [inventory](#).

Track an agent's spend

Use an agent's **Economics** tab to see what it spends on models, where that spend lands, and what each session did for it. Every role can read it; setting a budget needs the admin role or higher.



An agent's Economics tab: spend, budget outlook and the cost of each session.

Read the spend

Section	What it shows
Spend	What the agent spent over the window. Spend billed through an API key is metered; spend under a subscription is estimated, and labelled so
Budget outlook	How the month's spend compares with the agent's budget
Cumulative burn	The actual spend to date, projected to the end of the period
Cost by model	How the spend divides between the models the agent used
Share of input served from cache	How much of the agent's input the model provider served from its cache
Sessions	Each session with its cost, its duration and the tools it used. A session still running shows a running total, not a final figure

Model requests and responses are never sent to OpenLatch; spend comes from the token counts, the model used and the cache split the client records. See [Data handling and security](#). Where the price of a model is unknown, the tab says that some spend is missing from its figures.

Set a budget

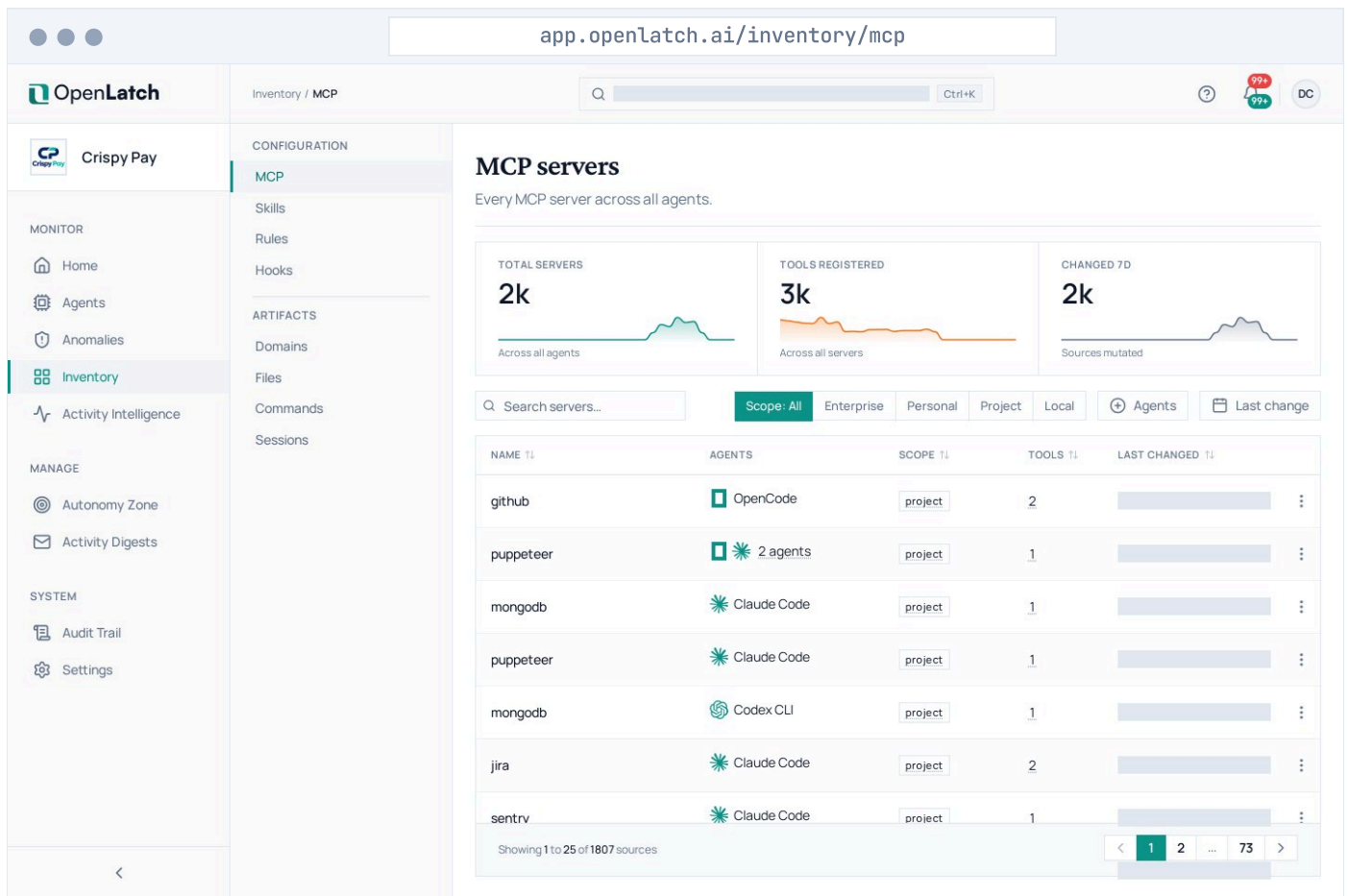
- 1 On the **Economics** tab, open the agent's monthly budget.
- 2 Enter the amount for the month and save it. The budget resets on the first of each month; nothing carries over.

An agent without a budget of its own uses your organization's default monthly budget, set in **Settings** › **Organization Profile**. The default applies to each such agent separately: a default of \$500 gives each agent \$500 a month, not \$500 shared. See [Edit the organization profile](#).

Search the inventory

Use the **Inventory** page when your question starts from a thing rather than an agent: which agents connect to this MCP server, which ran this command, which reached this domain. Each tab lists one kind of thing across every agent in the organization, counted once however many agents share it. Every role can read it.

Tab	Lists, across all agents
Commands	Every shell command the agents ran
Files	Every file the agents read, wrote or deleted
Domains	Every website the agents reached
Sessions	Every working session, each with the agent it belongs to
Hooks	Every hook configuration block the agents declare
MCP servers	Every MCP server the agents connect to
Rules	Every rules file the agents read
Skills	Every skill the agents can run



The MCP servers inventory: every server, with the number of agents it was seen on.

Find which agents use something

1. Open **Inventory** and the tab for the kind of thing you are looking for.
2. Search for it, or filter the list.
3. Read the **Agents** column. Hover it to see which agents the entry was seen on, or open the entry for every occurrence, each linked to its agent.

From an occurrence, open the agent to read the action in context; see [Investigate what an agent did](#).

app.openlatch.ai/inventory/commands

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

Inventory / Commands

CONFIGURATION

MCP

Skills

Rules

Hooks

ARTIFACTS

Domains

Files

Commands

Sessions

Commands

Every shell command this agent ran, ordered by time and scored per occurrence.

TOTAL COMMANDS

30k

40 unique binaries

Search by command text...

Agents

Binary

TIME	BINARY	COMMAND	AGENTS
	aws	aws iam attach-role-policy --role-name s...	122 agents
	grep	grep -rn TODO src/	147 agents
	psql	psql -h dev-db -c truncate table scratch...	121 agents
	bash	bash scripts/bootstrap.sh	122 agents
	cd	cd /app/scripts/projects	2 agents
	pwd	pwd	149 agents
	pytest	pytest -q services/authorizer -x	122 agents
	pytest	pytest -q services/authorizer	26 agents

Showing 1 to 25 of 1231 commands

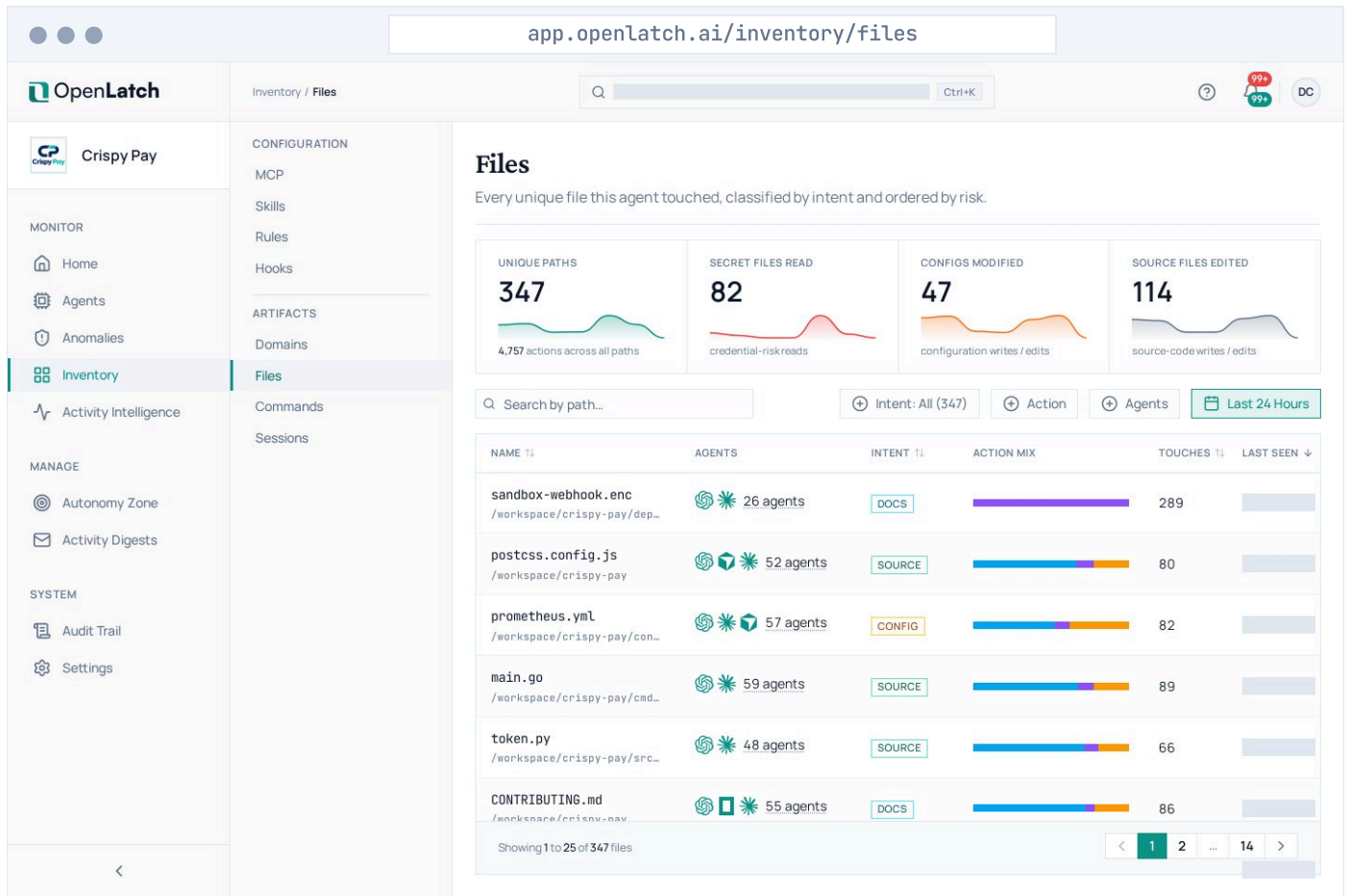
1

2

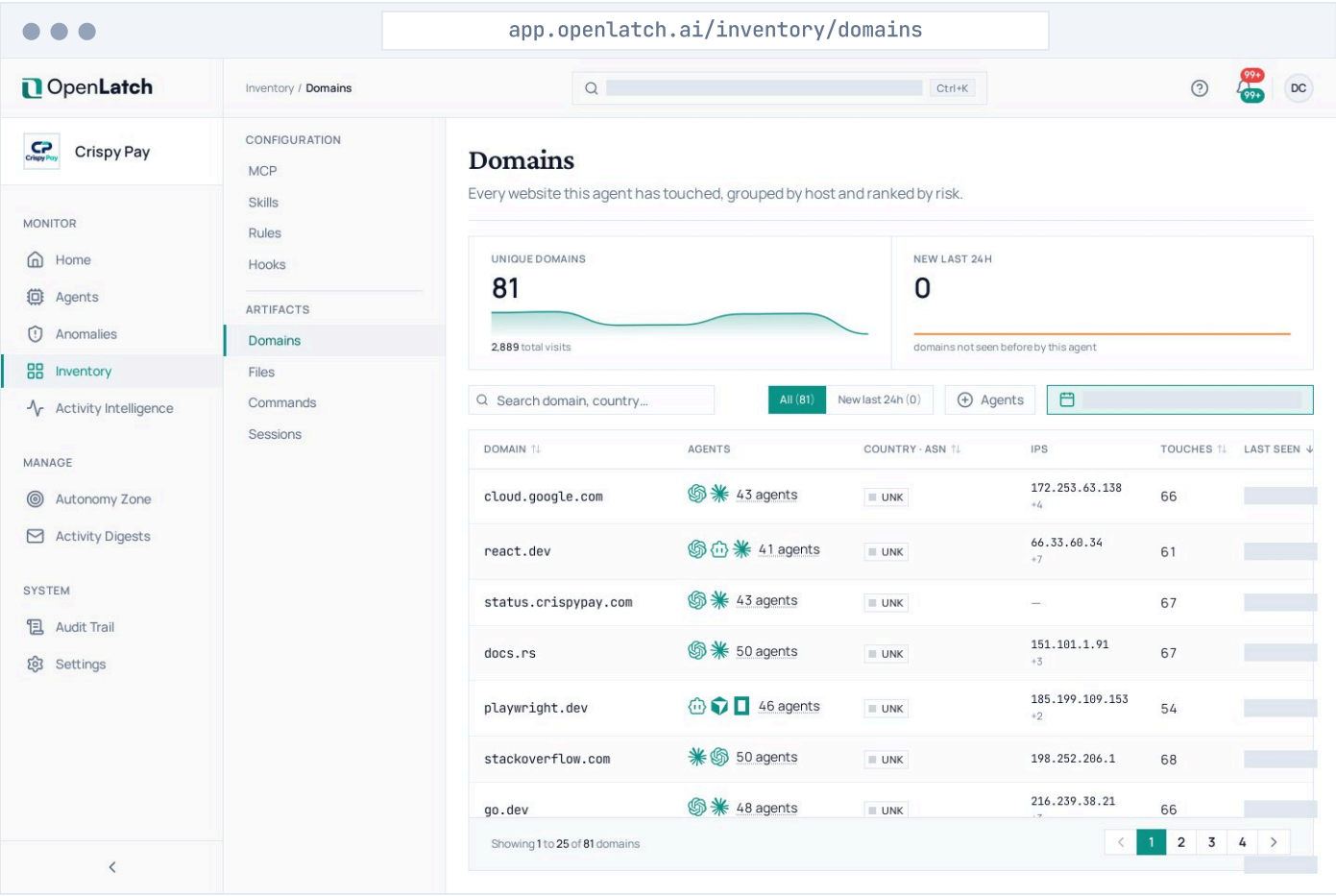
...

50

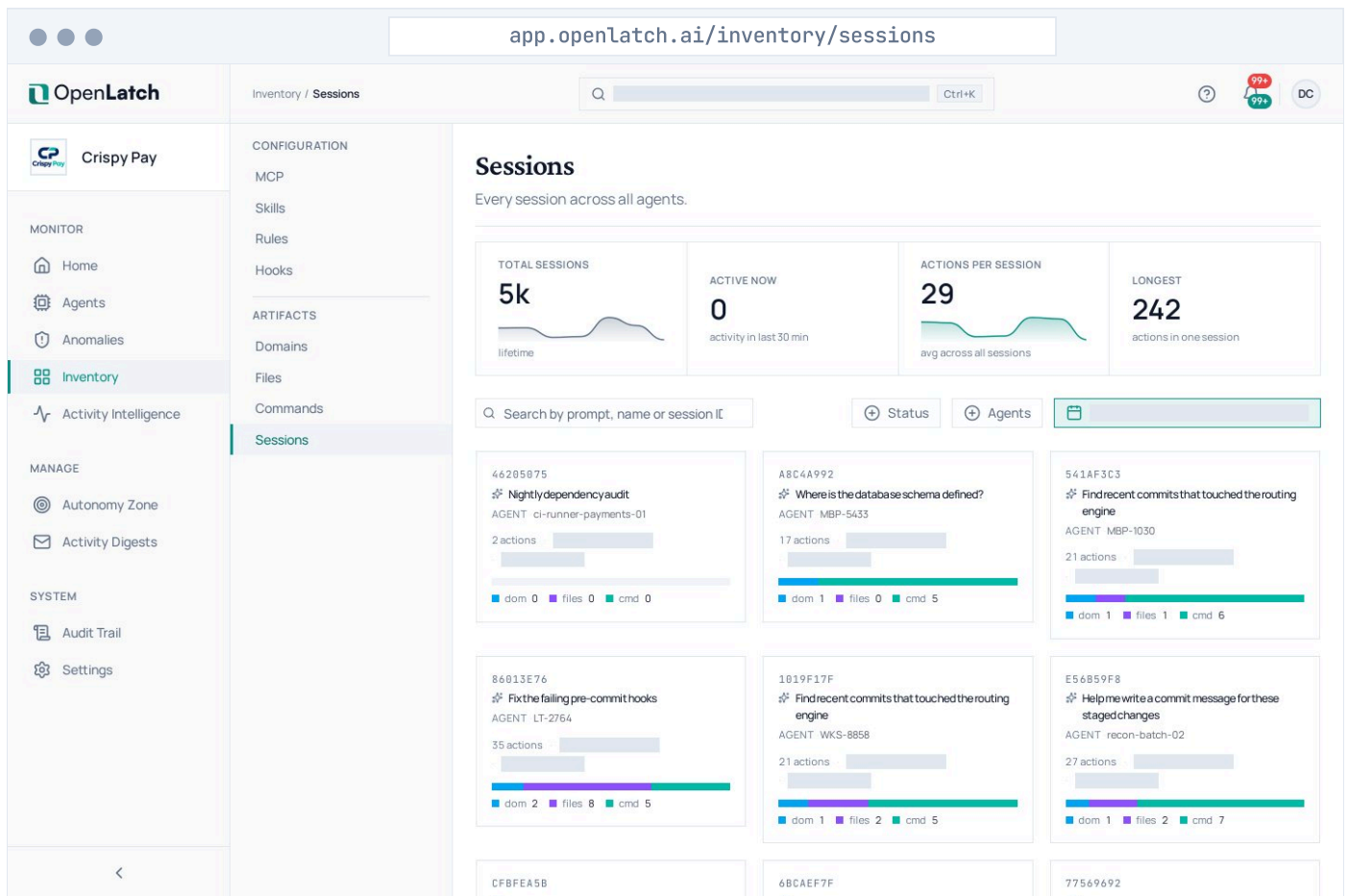
The Commands inventory: every shell command, across all agents.



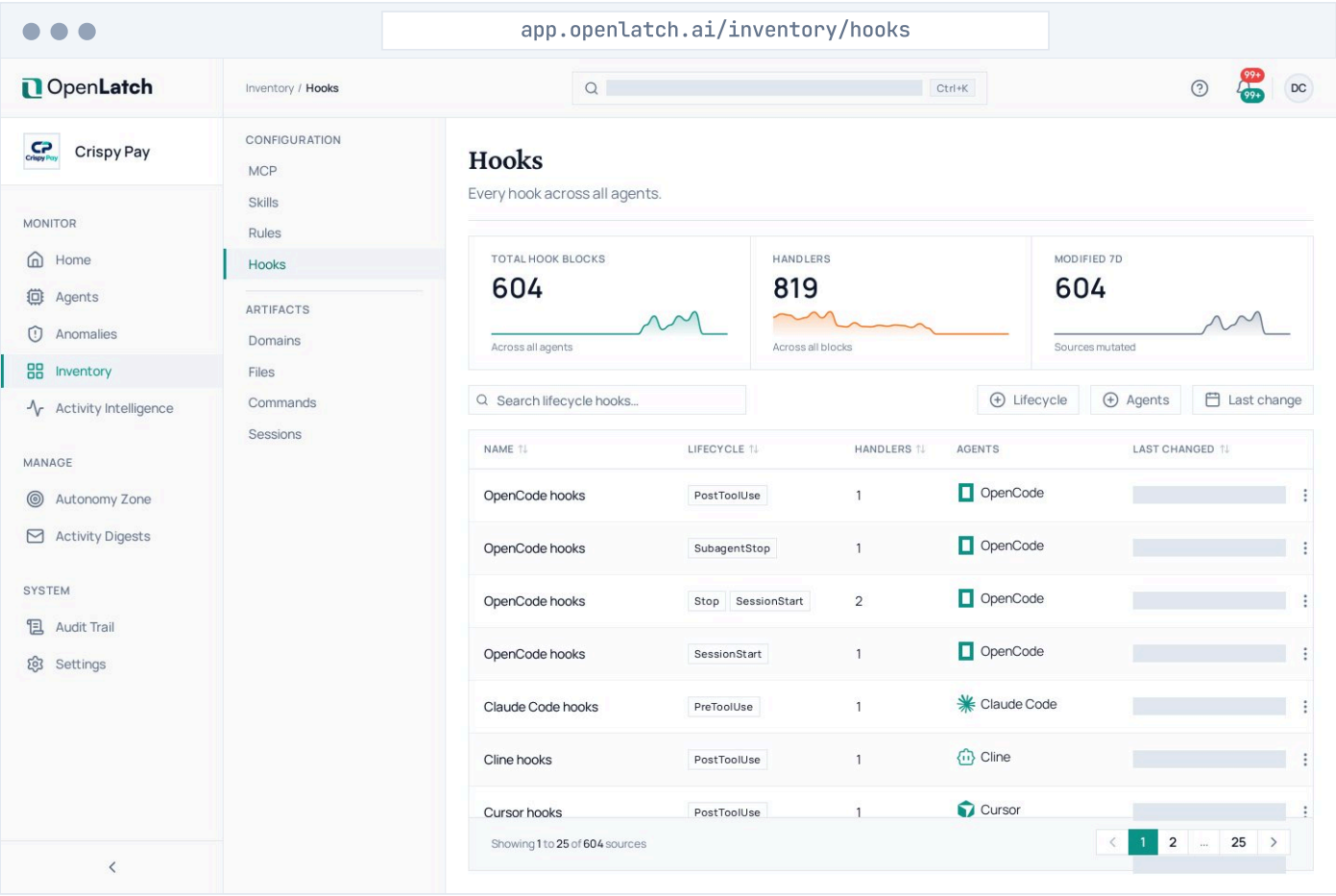
The Files inventory: every file the agents read, wrote or deleted.



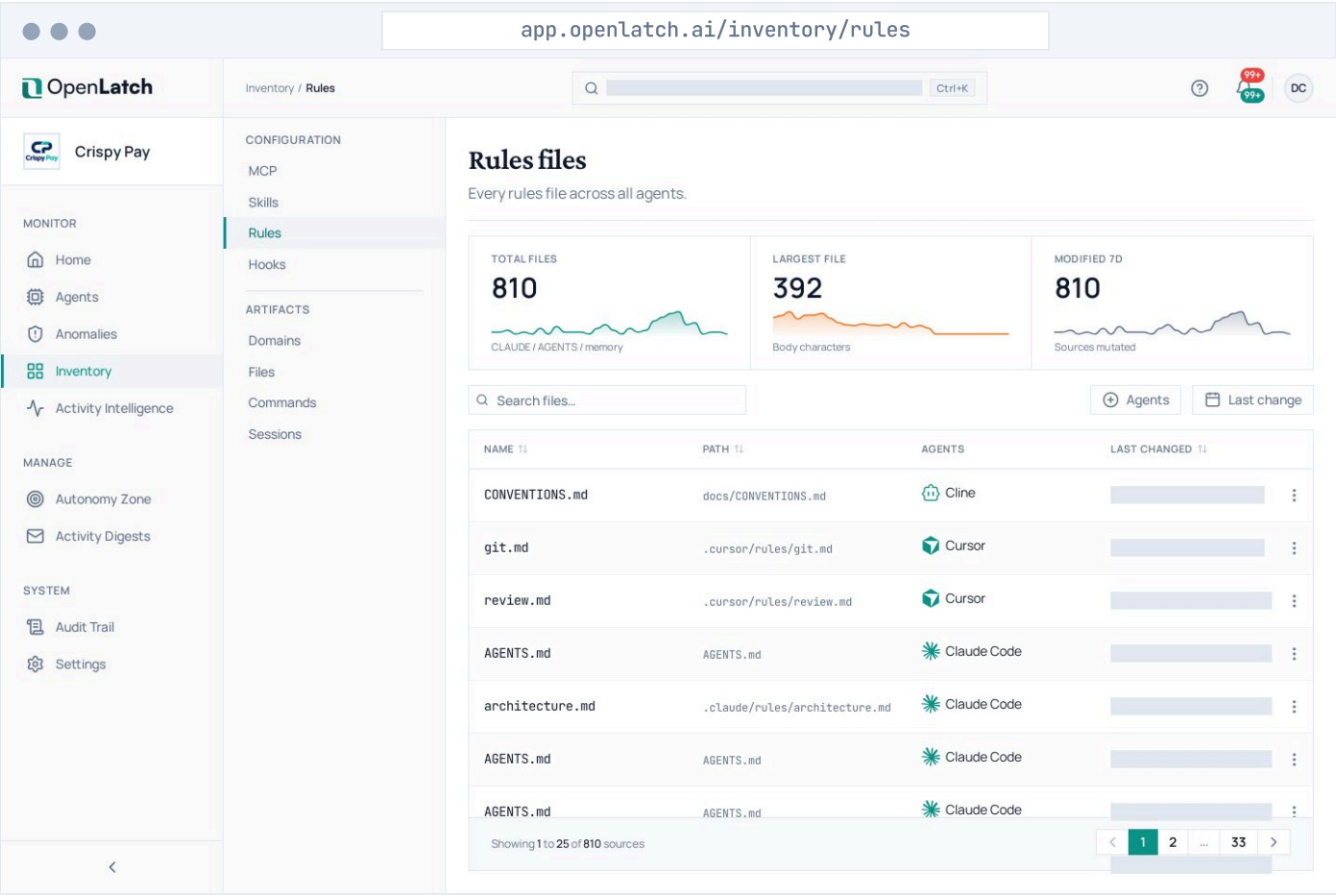
The Domains inventory: every website the agents reached.



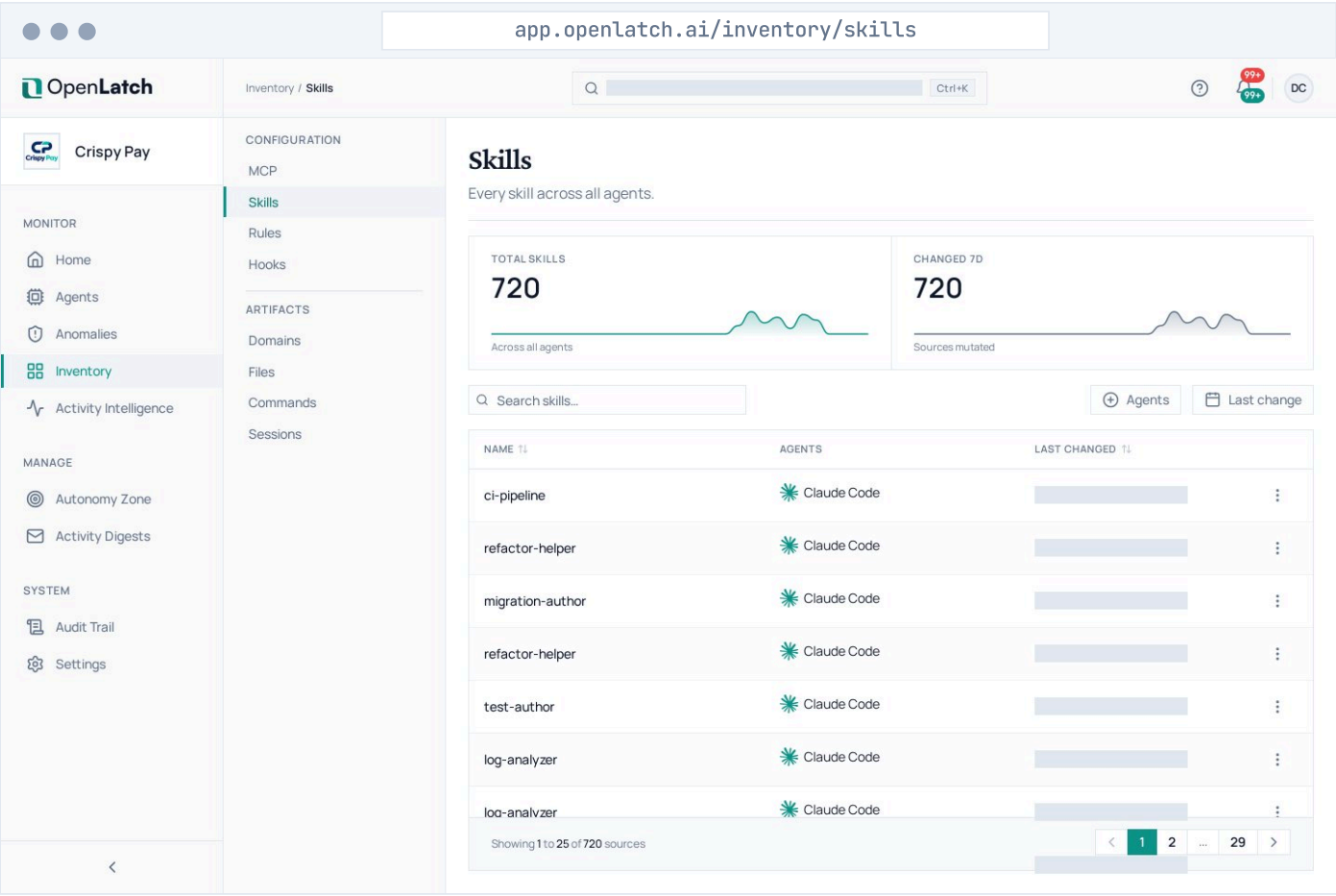
The Sessions inventory: every working session, with the agent it belongs to.



The Hooks inventory: every hook configuration block the agents declare.



The Rules inventory: every rules file the agents read.



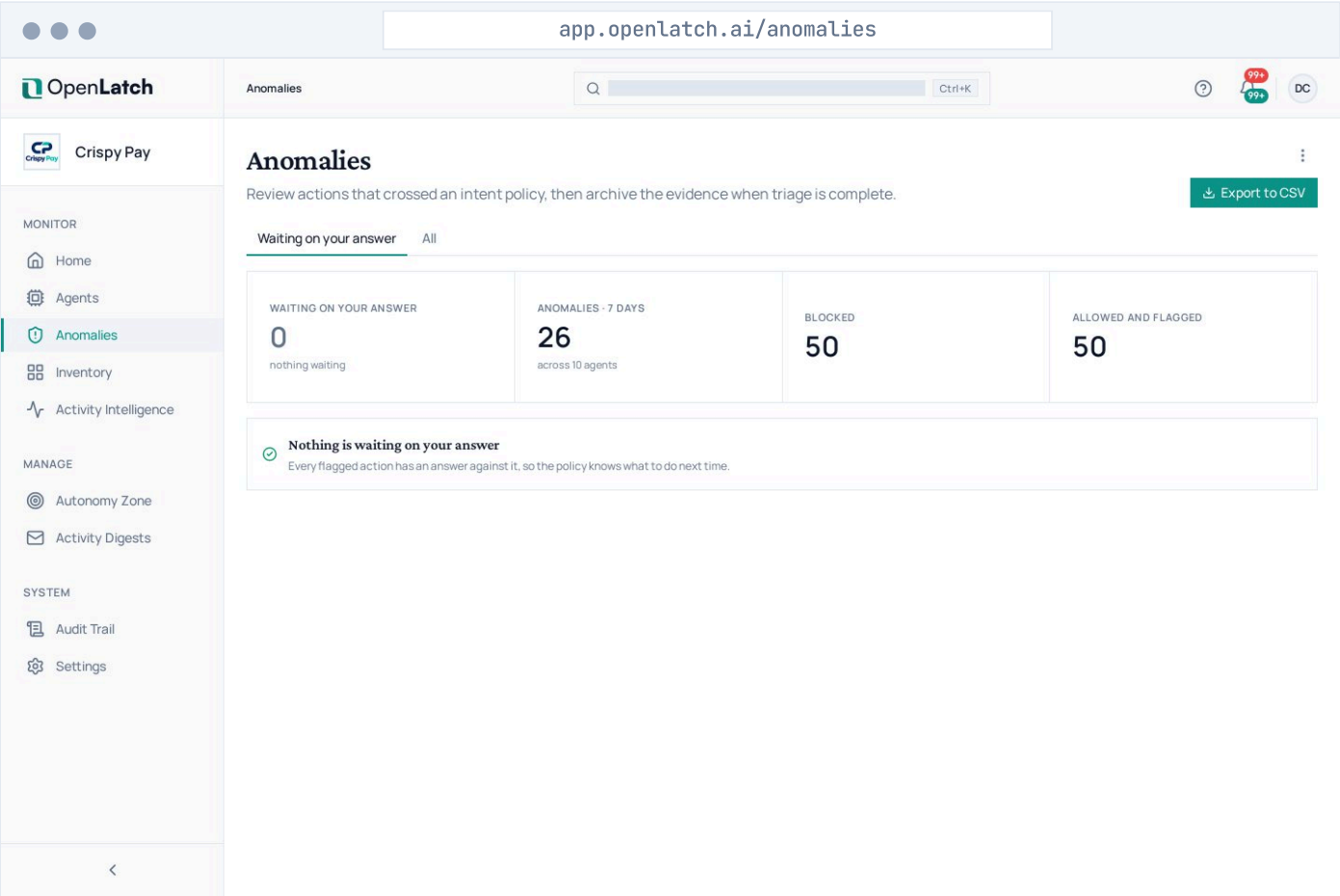
The Skills inventory: every skill the agents can run.

An empty tab means no agent has declared or done anything of that kind yet.

Review anomalies

Use the **Anomalies** page to work through every agent action in your organization that violated an intent policy. Every role can read it; answering an Ask needs the editor role or higher.

An anomaly is an agent action that violates an intent policy of one of the dimensions of the agent's Autonomy Zone. Each one names the policy, its dimension, the zone the policy came from and the verdict the action received. [Anomalies](#) explains the model.



The Anomalies page: each action that violated an intent policy, with the policy, its dimension and the verdict.

Work through the list

1. Set the time window and use the filters to narrow the list to what you are reviewing.

2. Open an anomaly. Its detail shows which Autonomy Zone and which policy it came from, which intent matched, the action that raised it, how often this happens, and the agent.
3. Open the agent from the anomaly to read the action in its session; see [Investigate what an agent did](#).

What an anomaly means depends on the policy's mode:

Policy mode	What happened to the action
Monitor	It proceeded, and the anomaly records what the policy would have decided
Enforce	It received the policy's verdict before it ran, and the anomaly records it

Answer an Ask

When a zone's ASK delivery includes the platform, the actions flagged by an Ask wait at the top of this page for a person to answer. By default those actions have already run: your answer does not undo them, it decides what the policy applies the next time it meets a similar action. Ask prevents the recurrence, not the first occurrence.

Answer	Effect
Allow next time	The policy lets similar actions run from now on
Block next time	The policy stops similar actions from now on
Not now – 24 h	The question comes back in a day. Until you answer, it fires again on the same ground

Where a policy opted in, per atom, to stop an irreversible action until someone answers, the action is listed as blocked until answered. Every answer is recorded in the audit trail with your name. [Manage Autonomy Zones](#) covers ASK delivery.

From an anomaly's detail, **Promote to Enforce** switches its policy to Enforce in one step, when the policy has enforceable intents.

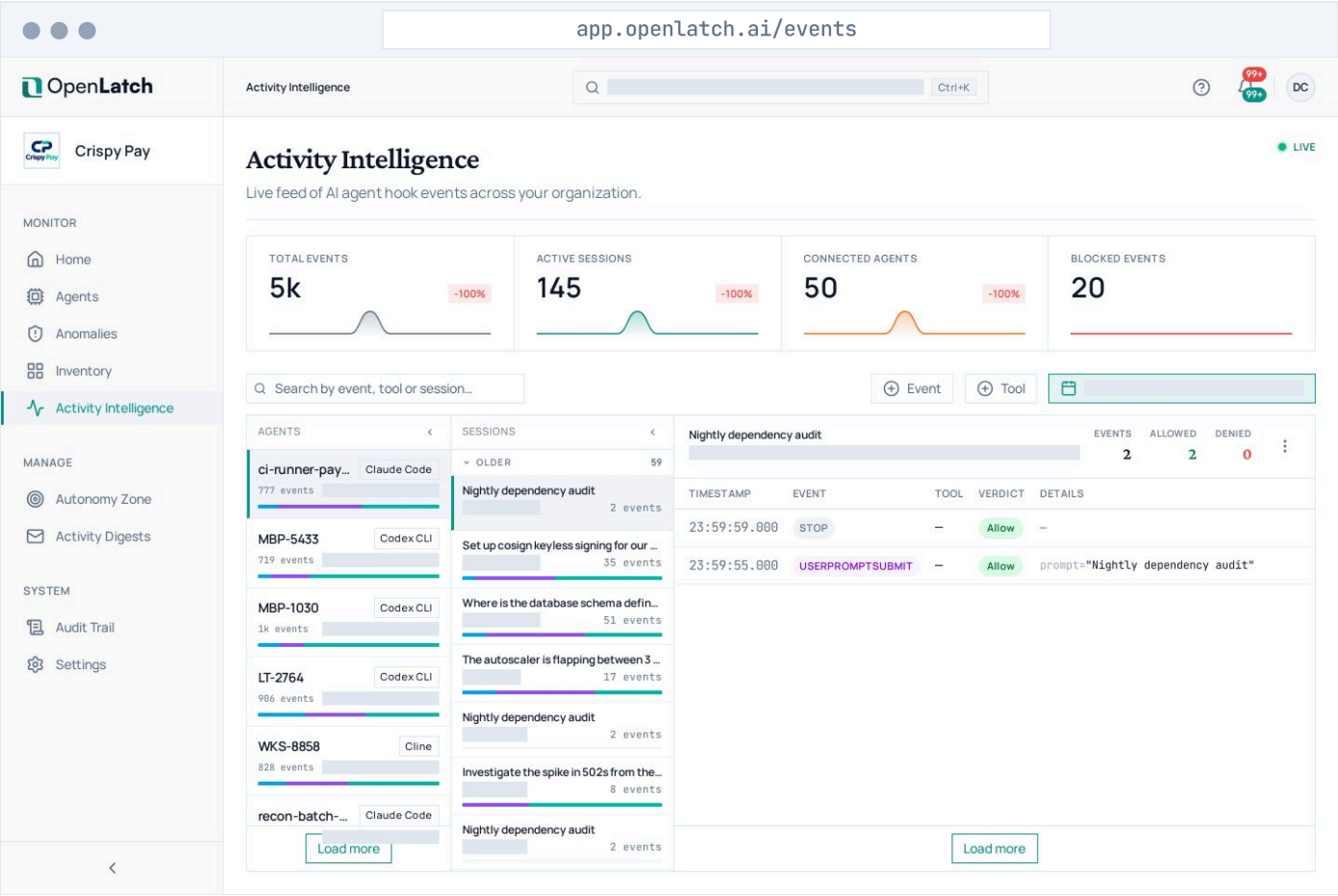
Tune the policy behind an anomaly

An anomaly on work you consider normal means the policy says more than you meant. Open the policy from the anomaly, answer No to its sentence with what you meant, and ratify the new version; see [Write an Agent Intent Policy](#). A policy whose anomalies are all ones you want acted on is ready to [graduate to Enforce](#).

To see one agent's anomalies only, open its **Anomalies** tab from the **Agents** page.

Follow live events

Use **Activity Intelligence**, the console's live events page at `/events`, to watch what agents are doing right now: every hook event across your organization, as it arrives. Every role can read it.



Activity Intelligence: the live feed of agent hook events, grouped by session.

Read the feed

The figures at the top count, over the selected time range, the events received, the active sessions, the connected agents and the blocked events, each compared with the previous half of the window. A live indicator shows that the feed is connected.

Below them, sessions are listed on the left, grouped into **Active now**, **Recent** and **Older**. Select a session to read its events on the right, in order. Open an event for its detail.

To	Do this
Find an event	Search by event, tool or session
Narrow the feed	Filter by event type, tool or time range
Start over	Choose Clear filters

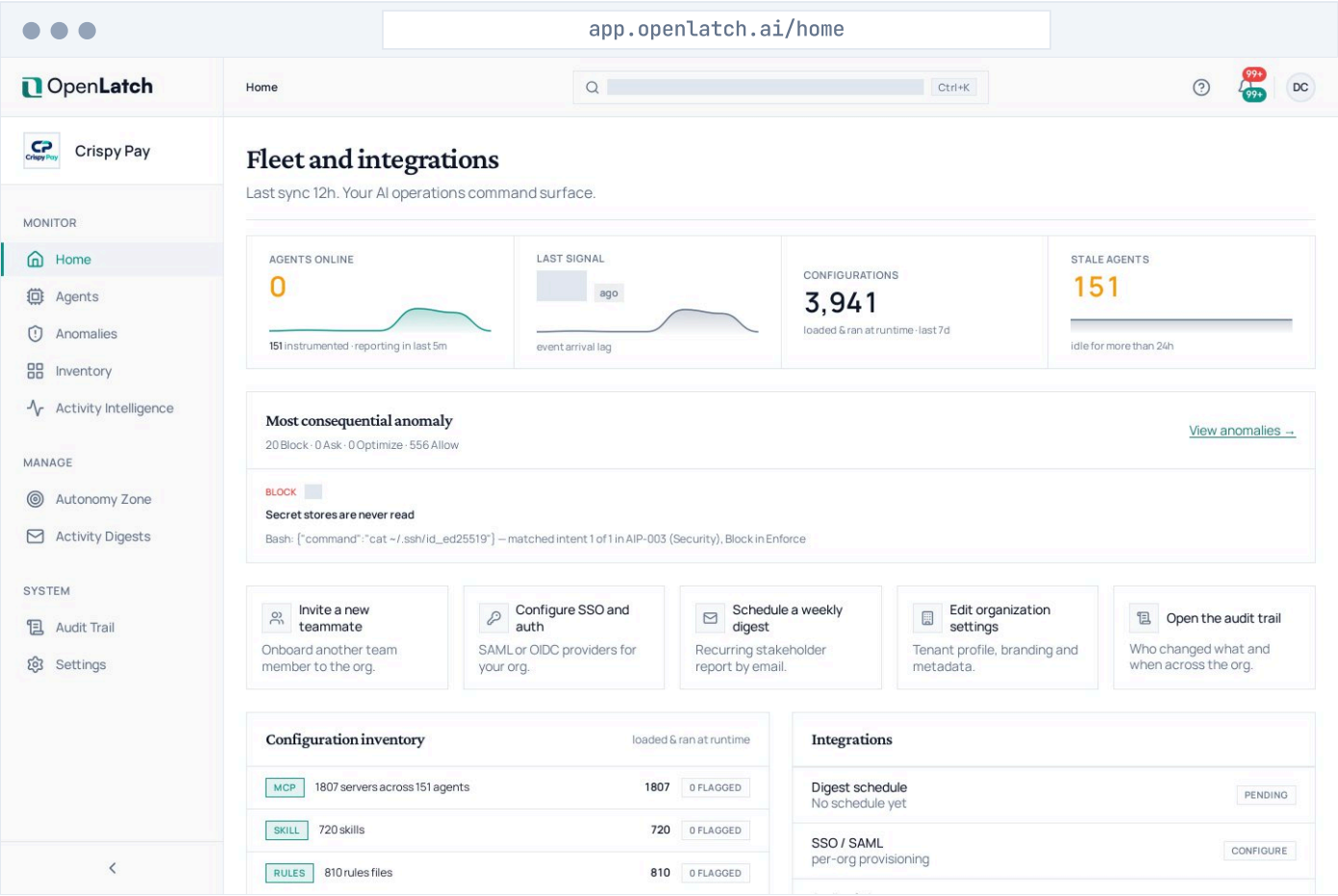
Before the first event

Right after you connect a client, the page shows **Awaiting first agent activity**: the client is connected and nothing has happened yet. The first event appears as soon as an agent invokes a tool. If nothing appears after the agent has worked for a while, run `openLATCH doctor` on the machine; see [Verify and roll out](#).

For the full history of one agent rather than the live feed, use its tabs; see [Investigate what an agent did](#).

Read the Home page

The **Home** page is where the console opens after you sign in. It summarizes the state of your organization and points to what needs a look. Opening `app.openlatch.ai` without a path lands you here.



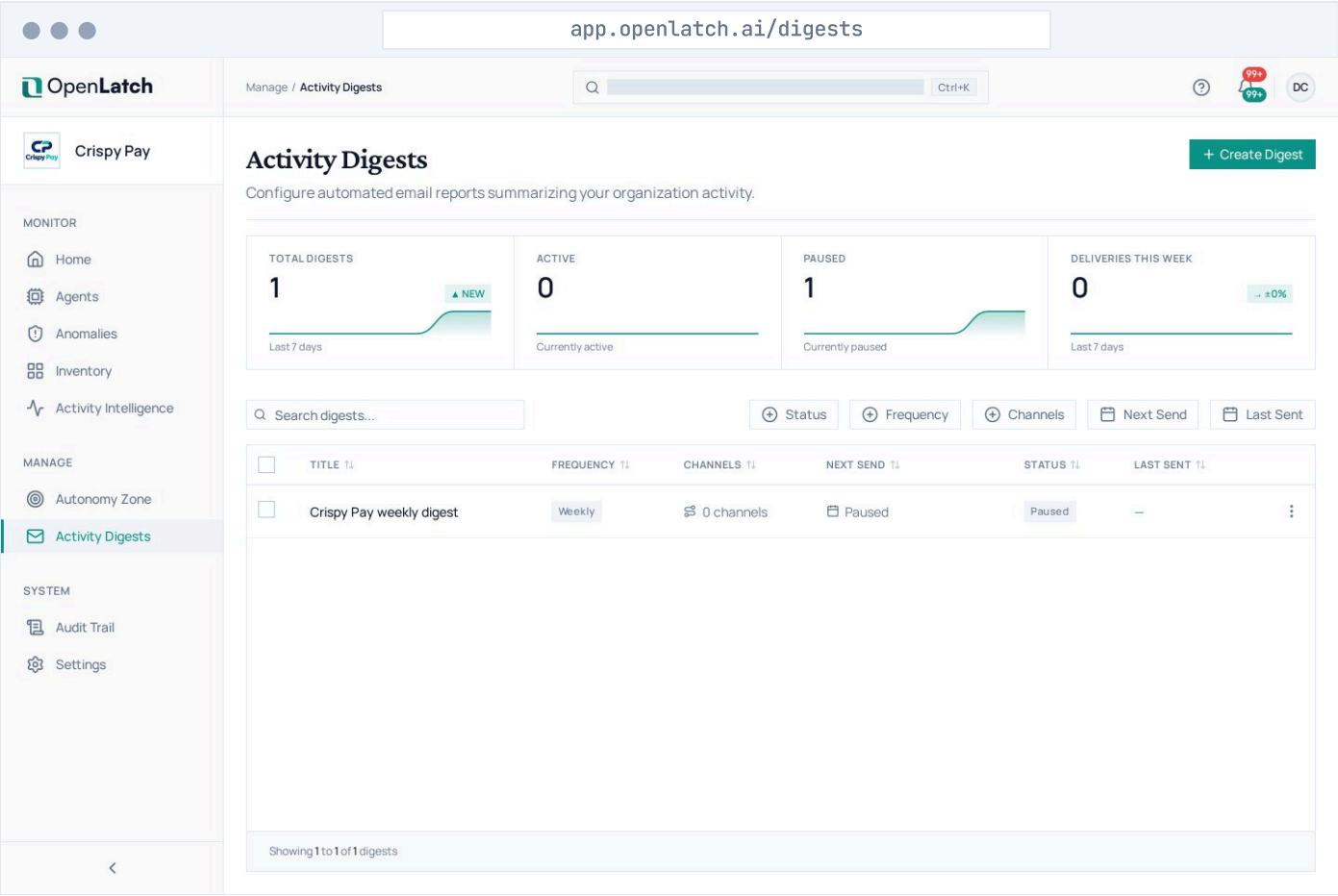
The Home page: the agents reporting, what changed since your last visit, and what to look at next.

Panel	What it tells you	Where it leads
Fleet health	How many of your agents are reporting	Find and read an agent
Configuration inventory	How many configuration sources (hooks, MCP servers, rules files, skills) were added, modified or removed since your last visit	Search the inventory
Integrations	Your active digest schedules	Schedule activity digests
Single sign-on	The state of your sign-in providers. Reading it needs an administrator role	Set up single sign-on
Recent changes	The latest entries of the audit trail	Review the audit trail
Team	How many members the organization has	Manage users and teams
Most active users	The people with the most agent actions over the last seven days	Manage users and teams
If you have 5 minutes	A short list of what to do next, such as inspecting a changed configuration source, connecting another agent or setting up a digest, or a note that you are all caught up	The page of each step

The **Since you last visited** list names each configuration change since your previous visit. Open one to read it in the inventory. **Reopen setup** takes you back to the organization's setup wizard.

Schedule activity digests

Use **Digests** to send a report of your organization's agent activity by email on a schedule, to people inside or outside the console. You need the editor role or higher to create or change a digest.



Digests: each schedule with its frequency, channels, next send and status.

Create a digest

- 1 Open **Digests** and choose **Create Digest**.
- 2 Give it a title and a subject line. The subject can use {period} and {organization_name}.
- 3 Under **Notification Routing**, select your Email channel. The recipients field appears only once an Email channel is selected.
- 4 Add the recipients, up to 100 email addresses. Recipients do not need a console account.

- 5 Choose the frequency, daily, weekly, monthly or quarterly, then the day, the send time and the time zone.
- 6 Choose the content sections the report carries, and save.

Digests are sent through the email channel; set it up first in [Set up notification channels](#).

Manage digests

The list shows each digest's frequency, channels, next send, last send and status. Search it, or filter by status, frequency, channel or send date.

To	Do this
Stop a digest for a while	Pause it. Its status reads Paused until you make it active again
Change it	Open it and edit its details
Read what it sent	Open it, then its Generated Reports tab

Read a report

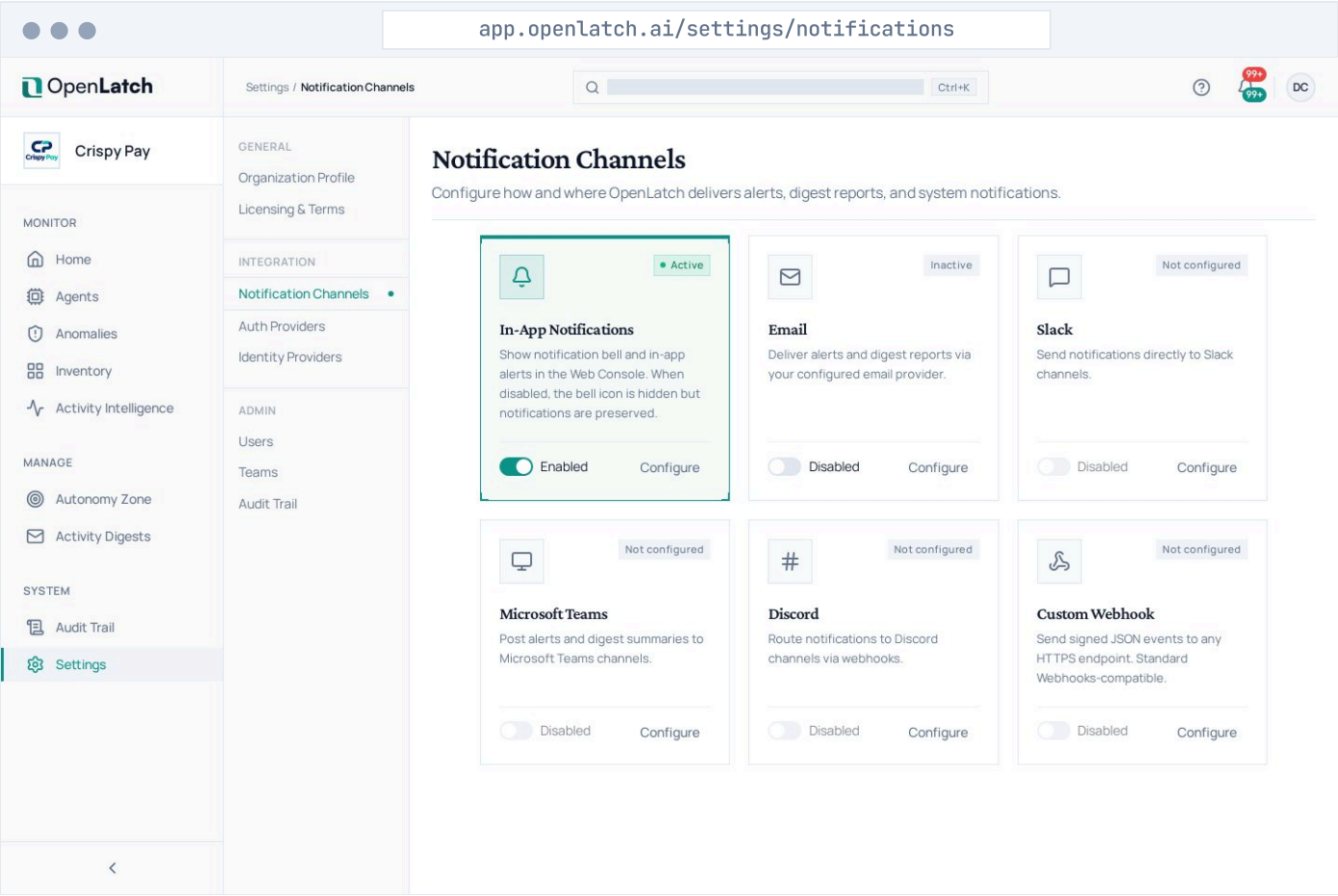
A generated report opens on its own page, laid out as it is sent: a plain-language summary of the reporting window, the figures behind it, and a section that explains how each figure was derived and what each term means.



A digest report: the summary of the window, the figures, and how each was derived.

Set up notification channels

Use **Settings > Notification Channels** to choose how and where OpenLatch delivers notifications and digest reports. You need the admin role or higher.



Notification Channels: each channel OpenLatch can deliver to, and whether it is configured.

Channels

Channel	What it does
In-app notifications	Shows the notification bell in the console. Turning it off hides the bell; notifications are kept
Email	Delivers notifications and digest reports through the email provider your OpenLatch site is configured with
Slack	Sends notifications to Slack channels
Microsoft Teams	Posts notifications and digest summaries to Microsoft Teams channels
Discord	Routes notifications to Discord channels through webhooks
Custom webhook	Sends signed JSON events to any HTTPS endpoint, compatible with the Standard Webhooks specification

Slack, Microsoft Teams, Discord and custom webhook channels can each be added more than once, for example one per team.

Add a channel

- 1 On the channel's card, choose **Configure**.
- 2 Follow the setup guide in the drawer, then enter the destination: a Slack, Microsoft Teams or Discord webhook address, or your own HTTPS endpoint.
- 3 Choose which notifications the channel receives, and save.
- 4 Send a test, and check that it arrives.

Verify webhook signatures

A custom webhook signs every event with a signing secret, so your endpoint can check that an event came from OpenLatch. The secret is shown once, when you reveal it to copy into your endpoint's configuration. **Rotate signing secret** issues a new one; update your endpoint at the same time.

To send a recurring report rather than individual notifications, see [Schedule activity digests](#).

Review the audit trail

Use the **Audit Trail** page to review every change and action across your organization: who did what, to what, and when. The trail is append-only and hash-chained, so an edited or removed entry breaks the chain. Its settings, under **Settings** > **Audit Trail**, need the admin role or higher.

OpenLatch

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

app.openlatch.ai/audit-trail

Manage / Audit Trail

Q

Ctrl+K

?

99+

99+

DC

Audit Trail

Review all changes and actions across your organization.

Q Search by actor, target, or changes..

Source

Action

Target Type

Status

Date Range

☐	TIMESTAMP	ACTOR	SOURCE	ACTION	TARGET	STATUS	CHANGES
☐		User Documentation user-doc@openlatch.ai	Auth	Login	Session user-doc@openlatch.ai	Success	
☐		Docs Capture Owner docs-capture@crispypayex...	Auth	Login	Session docs-capture@crispyp...	Success	
☐		Docs Capture Owner docs-capture@crispypayex...	Auth	Login	Session docs-capture@crispyp...	Success	
☐		Luc Delsalle luc.delsalle@openlatch.ai	API	Directory Providi	Directory Provider Corporate Dir...	Success	
☐		Luc Delsalle luc.delsalle@openlatch.ai	API	Directory Providi	Directory Provider Corporate Dir...	Success	
☐		User Documentation user-doc@openlatch.ai	Auth	Login	Session user-doc@openlatch.ai	Success	
☐		Docs Capture Owner docs-capture@crispypayex...	Auth	Login	Session docs-capture@crispyp...	Success	
☐		Docs Capture Owner docs-capture@crispypayex...	Auth	Login	Session docs-capture@crispyp...	Success	
☐		Luc Delsalle					

Showing 1 to 25 of 815 events

< 1 2 ... 33 >

Export to CSV

The Audit Trail page: every change and action, with its actor, target and status.

Find an event

- 1
- Open **Audit Trail**.
- 2
- Search by actor, target or change, or filter by source, action, target type, status and date range.
- 3
- Open an event. Its detail shows the actor, the context, the status, each changed field with its old and new value, and the event's place in the integrity chain.

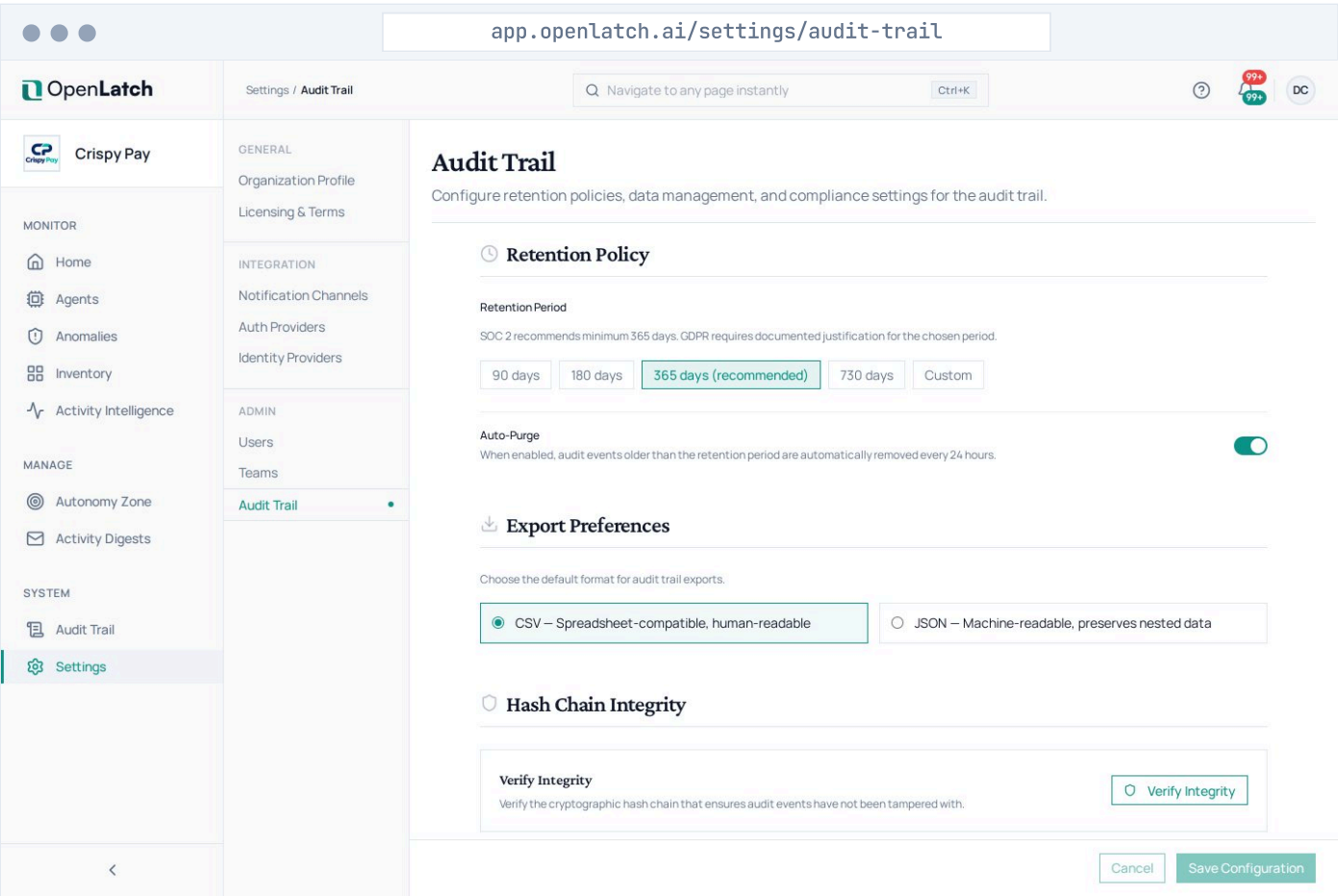
Every policy change appears here with the actor and the state before and after, and so does every answer given to an Ask.

Export it

Choose **Export** to download the events that match your search and filters, as CSV or JSON. The export itself is recorded as an audit event.

Set retention and check integrity

Open **Settings > Audit Trail**.



The Audit Trail settings: retention, auto-purge, export format and integrity check.

Setting	What it does
Retention period	How long events are kept: 90, 180, 365 or 730 days, or a custom period. The console recommends 365 days
Auto-purge	When on, events older than the retention period are removed every 24 hours
Export format	The default format of an export: CSV, readable in a spreadsheet, or JSON, which keeps nested data
Verify integrity	Checks the hash chain of every event. A purge breaks the chain where it removed events, and the check reports that break

The same page holds two irreversible actions: anonymizing the trail, which replaces the actor's email, name and user ID on every event with anonymized values while keeping the records and the chain (IP addresses and the other details of each event stay as recorded), and deleting every audit event, which breaks the chain. Each asks you to type a confirmation word.

The audit trail is the one record whose retention you set. [Data handling and security](#) lists the fixed retention of every other record.

Manage users and teams

Use **Settings > Users** to decide who can use the console and what each person can do, and **Settings > Teams** to group people. You need the admin role or higher.

Roles

Four roles run in a strict order of privilege, each including everything below it:

Role	What it can do
owner	Full ownership: every permission, plus team and organization management
admin	Full access: users, teams, settings and all content
editor	Manages resources and configuration, such as intent policies, zones and agent ownership
viewer	Read-only access to resources and configuration

Every invitee starts as a viewer. Grant the higher roles deliberately, to the few people who need them. The **Role Permissions Reference** on the page lists each permission per role.

Invite someone

- 1

Open **Settings > Users** and choose **Invite User**.
- 2

Enter their email address and the role they start with, and send the invitation.
- 3

The person receives an email with a registration link tied to that address. You can also copy the invitation link and share it yourself.

An invitation can expire or be revoked, and each one can be used once. Whoever has an address in one of your organization’s allowed email domains can also join through single sign-on without an invitation; see [Set up single sign-on](#).

app.openlatch.ai/settings/users

OpenLatch

Settings / Users

Q Search by name or email... + Status + Team + Email Created Last Login

☐	NAME ↕	EMAIL ↕	TEAMS	ROLE	STATUS ↕	LAST LOGIN ↕	CREATED ↕	
☐	DO Docs Capture Owner	docs-cap...	Administr...	Owner	Active			⋮
☐	RK Rohan Kapoor	rohan.ka...	Policy Edi...	Editor	Active			⋮
☐	CL Camille Lefèvre	camille.le...	Policy Edi...	Editor	Active			⋮
☐	MW Marcus Washington	marcus...	Policy Edi...	Editor	Active			⋮
☐	IS Ingrid Solberg	ingrid.sol...	Administr...	Admin	Active			⋮
☐	UD User Documentation	user-doc...	Administr...	Admin	Active			⋮
☐	Luc Delsalle	luc.delsal...	Administr...	Owner	Active			⋮

Showing 1 to 7 of 7 users

Settings > Users: each member with their role and status.

Change, deactivate or remove someone

Open a user to change their role or their details, or to reset their password. Select several users to activate, deactivate or delete them together. A deactivated user keeps their history and cannot sign in. Every change of role or membership is recorded in the audit trail.

Organize people in teams

Teams group users, for example by department or by the agents they look after.

- 1 Open **Settings > Teams** and choose **Create Team**.
- 2 Name it, describe what it is responsible for, and add its members.

app.openlatch.ai/settings/teams

OpenLatch

Settings / Teams

Q

Ctrl+K

?

99+

99+

DC

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

GENERAL

Organization Profile

Licensing & Terms

INTEGRATION

Notification Channels

Auth Providers

Identity Providers

ADMIN

Users

Teams

Audit Trail

Teams

+ Create Team

Organize users into teams with shared permissions.

Q Search by name or description...

+ Role

+ Members

Created

☐	NAME ↑	DESCRIPTION ↑	ROLE ↑	MEMBERS ↑	CREATED ↑	
☐	Administrators System	Organization administrators	Admin	4		
☐	Policy Editors	—	Editor	3		
☐	Users System	Default user team	Viewer	0		

Showing 1 to 3 of 3 teams

ROLE PERMISSIONS REFERENCE

Settings > Teams: each team, its description and its members.

Deleting a team removes its memberships; the users themselves stay in the organization.

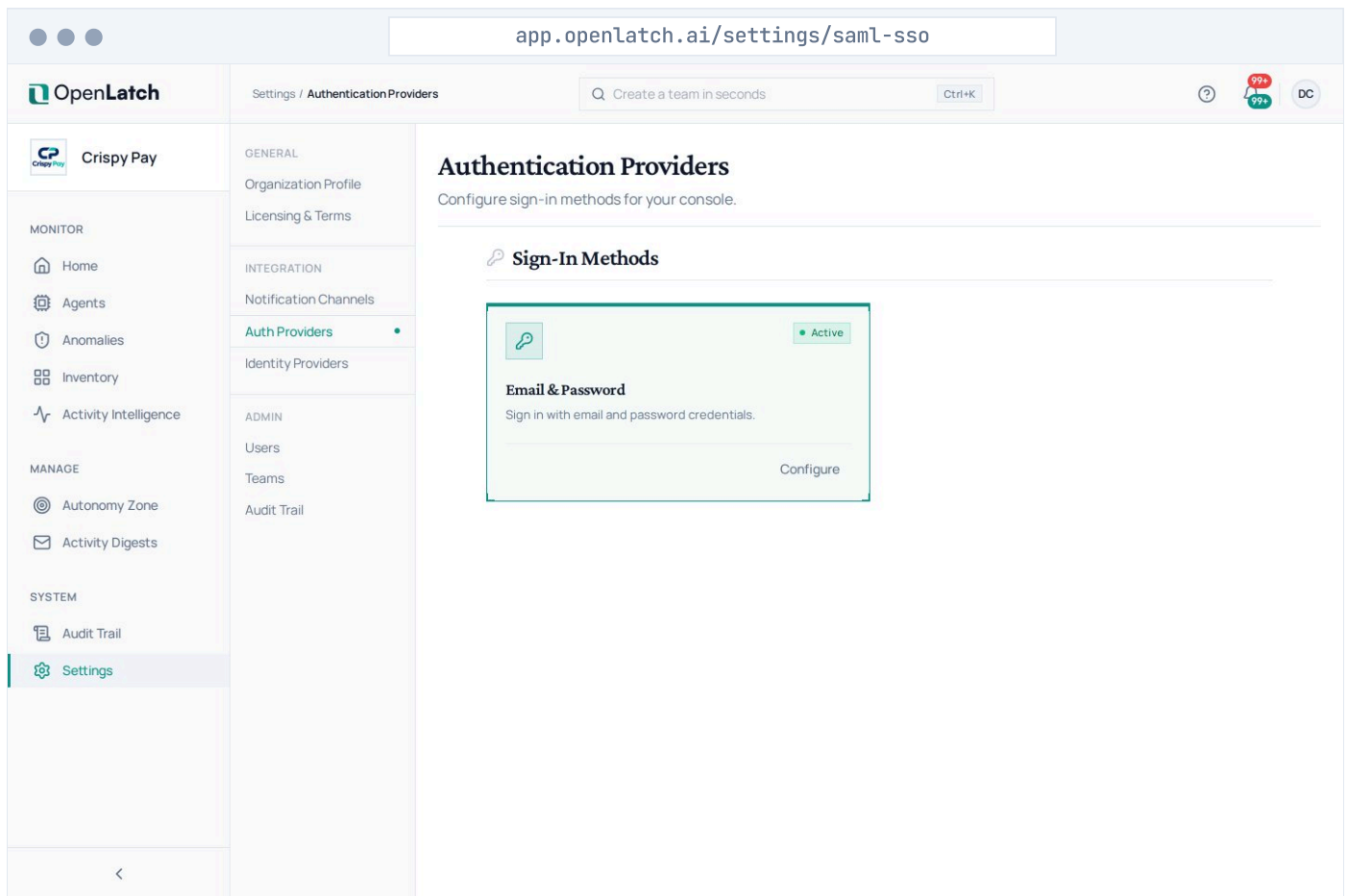
Set up single sign-on

Use **Settings > SAML SSO** to connect your corporate identity provider, and **Settings > Auth Providers** to review your organization's sign-in methods and require two-factor authentication. You need the admin role or higher.

Connect SAML single sign-on

The console supports SAML 2.0, configured per organization. Microsoft Entra ID, Okta, PingOne and Active Directory Federation Services connect this way.

- 1 Open **Settings > SAML SSO**. Copy the service provider information into your identity provider when you create the SAML application there: the Assertion Consumer Service (ACS) URL, the SP entity ID and the SP metadata URL.
- 2 Select your identity provider for setup instructions written for it, then enter its details: its entity ID, its sign-on URL, and its signing certificate or metadata URL. Your identity provider's assertion must carry the user's email address and display name.
- 3 Add the allowed email domains. People with an address in one of them can sign in through SAML. A domain can belong to one organization only; the console tells you if another organization has already claimed it.
- 4 Save, then run the connection test. It checks, against the saved configuration, that your identity provider's metadata URL or sign-on URL answers; it does not perform a SAML sign-in. Then sign in through SAML yourself, in a private browser window, before you tell anyone else to.

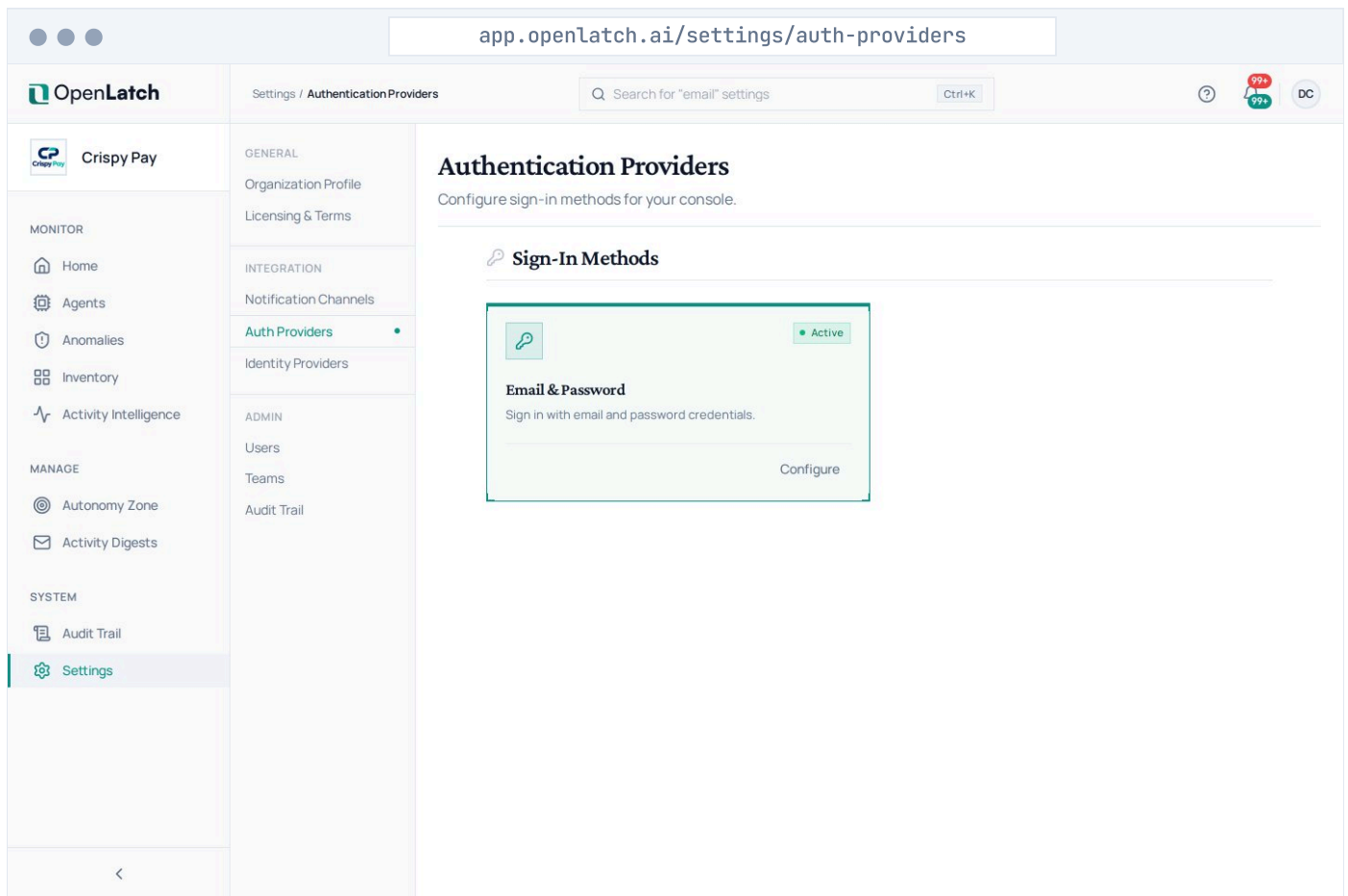


Settings > SAML SSO: the service provider information to copy, and your identity provider's details.

A per-organization SAML connection is the only sign-in method that admits a new member without an invitation, and only for the allowed domains.

Review sign-in methods and require two-factor authentication

Settings > Auth Providers shows your organization's sign-in methods as cards, each with its state: **Email & Password**, and **SAML SSO** when SAML is available on your OpenLatch site. **Configure** on a card opens its settings.



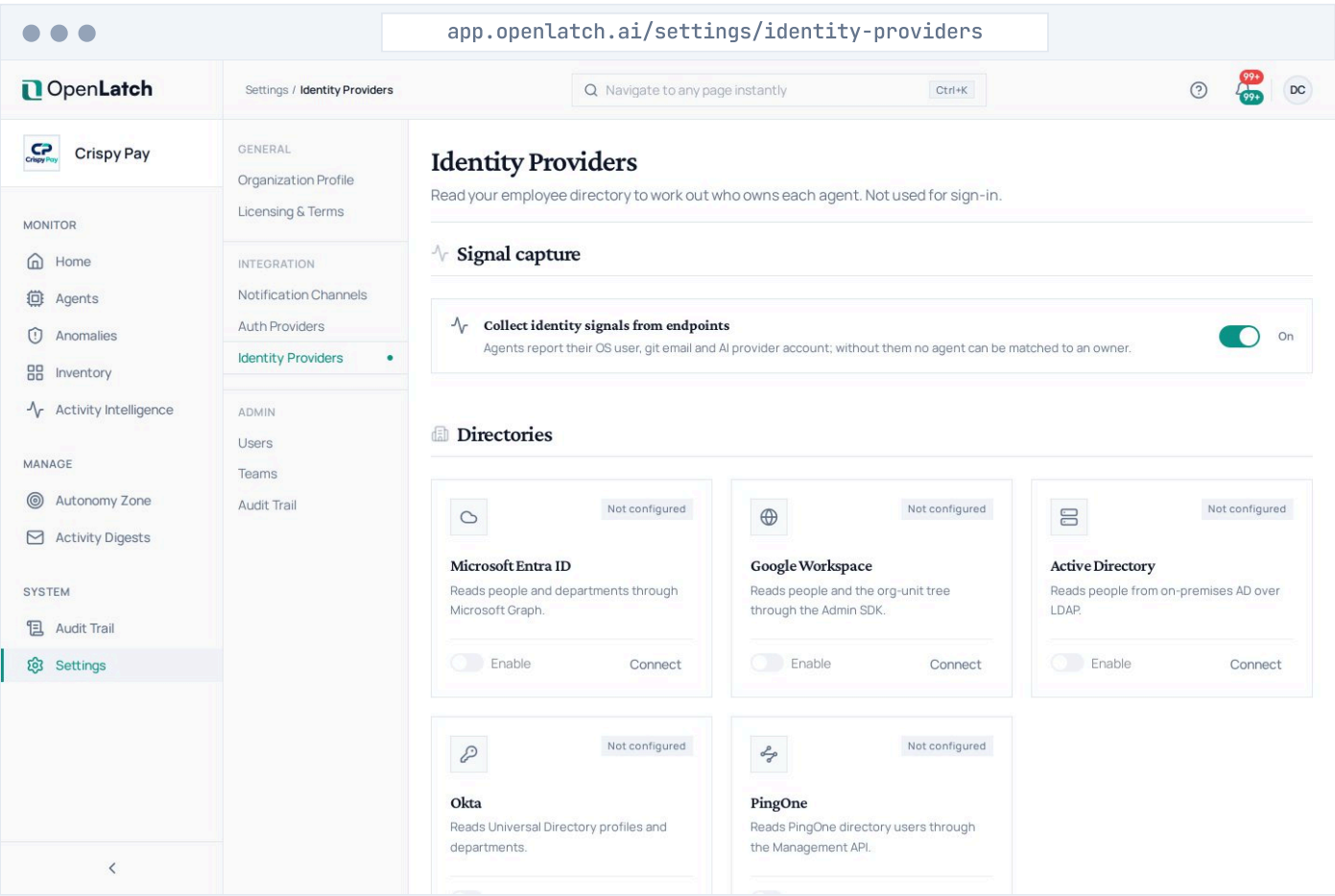
Settings > Auth Providers: the sign-in methods your organization allows.

Card	Configure opens
Email & Password	The two-factor policy. Require two-factor authentication to prompt every password-based user without it to set it up. After the grace period you choose, they are sent to their security settings at each sign-in until they do. Users who sign in only through an external provider are exempt. The card shows MFA Required while the policy is on
SAML SSO	The SAML connection, configured as above

Each person sets up their own two-factor authentication from their account; see [Manage your account](#).

Connect a directory for owner lookup

Use **Settings > Identity Providers** to read your employee directory, so OpenLatch can work out who owns each agent and where they sit in your organization. The connection is read-only and is not used for sign-in; sign-in is [Set up single sign-on](#). You need the admin role or higher to change it.



Settings > Identity Providers: the signal capture switch and the connected directories.

Ownership is matched from two sides, and both must be present:

Side	What it provides	Where
Signal capture	The machines report the signals that identify who is working: the operating system user, the git email and the AI provider account	The switch Collect identity signals from endpoints , on by default
Directories	The directory turns those signals into a named person, with their job title and their place in the organization	The directory cards

With signal capture off, no agent can be matched to an owner, however well the directory is connected.

Connect a directory

- 1 Open **Settings** > **Identity Providers**.
- 2 Choose the card of your directory: Microsoft Entra ID, Google Workspace, Active Directory, Okta or PingOne. Each is a read-only source: none creates users in OpenLatch, and none copies the directory into it.
- 3 Follow the drawer's steps to grant OpenLatch read access in your directory, then save. If the directory is missing a permission OpenLatch needs, the card says which one.
- 4 Check the result on an agent you know: its **Identity** tab should name its owner. See [Find and read an agent](#).

An organization with no directory connected is a supported state. An agent whose owner cannot be resolved is recorded as unattributed rather than guessed, and an editor can set its owner by hand from its **Identity** tab.

What the directory also gives you

The directory's units form the tree of your organization on the **Autonomy Zone** page, and each agent sits by default in its owner's unit. A unit that comes from the directory keeps the directory's name and place, and is updated on each sync; the Autonomy Zone you put on it stays yours. See [Manage Autonomy Zones](#).

Disconnect a directory

Disconnecting a directory removes its connection, its stored credentials and the people it resolved. Agents matched through it lose their owner until another directory resolves them.

The address `/settings/directory-sync` is an earlier name of this page and opens it.

Edit the organization profile

Use **Settings > Organization Profile** to manage how your organization appears and the defaults that apply to all of it. Opening **Settings** lands you on this page. You need the admin role or higher to change it.

app.openlatch.ai/settings/organization-profile

OpenLatch

Settings / Organization Profile

Q Go to Organization Profile Ctrl+K

?

99+

99+

DC

Crispy Pay

MONITOR

Home

Agents

Anomalies

Inventory

Activity Intelligence

MANAGE

Autonomy Zone

Activity Digests

SYSTEM

Audit Trail

Settings

GENERAL

Organization Profile

Licensing & Terms

INTEGRATION

Notification Channels

Auth Providers

Identity Providers

ADMIN

Users

Teams

Audit Trail

Organization Profile

Manage your organization's public identity and primary domain.

Identity

Organization Logo

Recommended size 400x400px

Crispy Pay

Change Logo Remove

Organization Name

Crispy Pay

The legal name of the entity operating this property.

System Metadata

ORGANIZATION ID

208bbd63-67fd-49c5-9bdf-9c6d55dd7f72

ORGANIZATION SLUG

crispy-pay

CREATED AT

SITE

us-east-1 (United States)

Your organization's data is stored in this region. OpenLatch's residency statement is available on request, in the trust pack and the DPA.

Agent Budgets

Reset to Defaults

Save Changes

Settings > Organization Profile: the organization's identity, system metadata and setup.

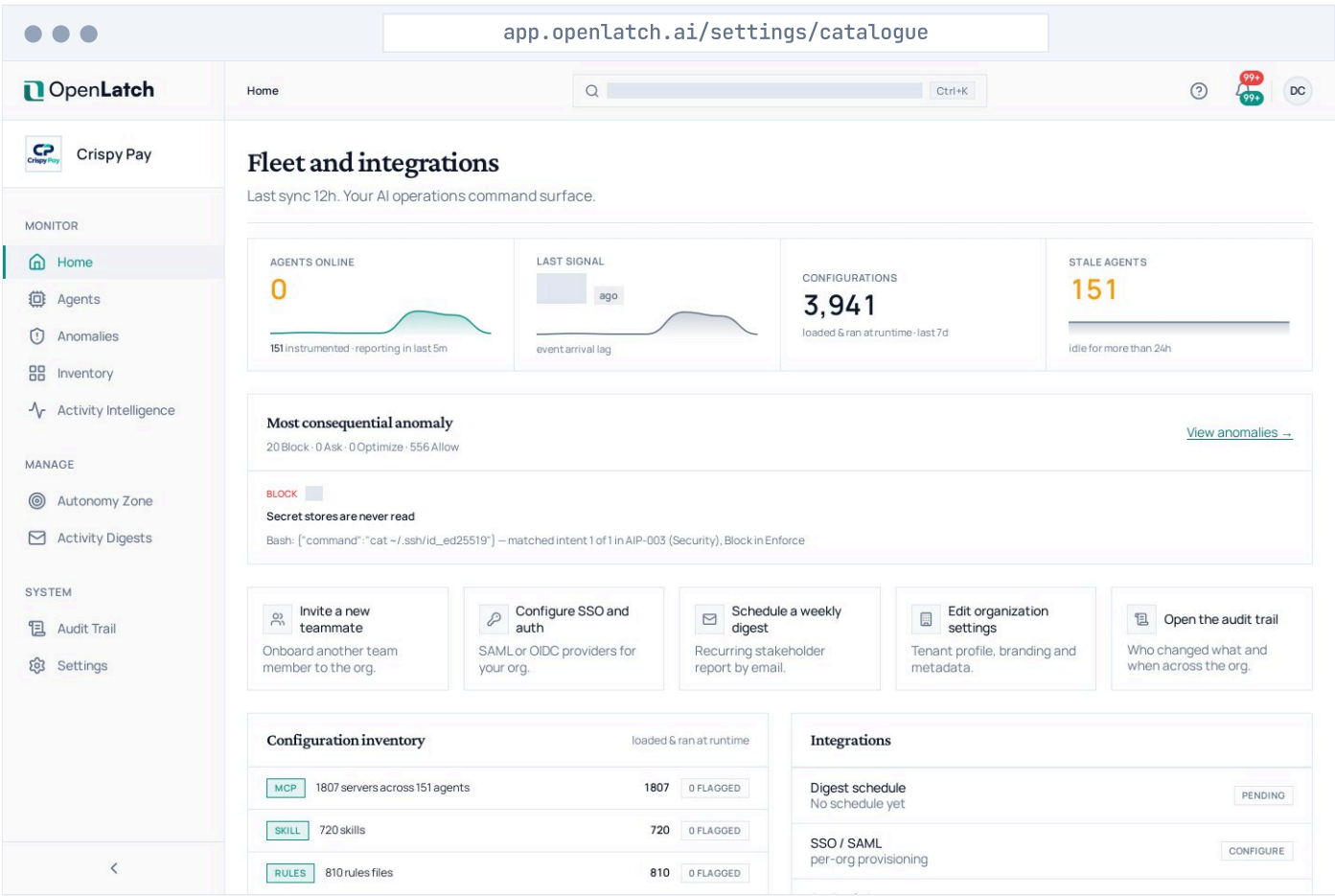
Section	What you can do
Identity	Change the organization's name, the legal name of the entity it represents, and upload or remove its logo, ideally 400×400 pixels. The name is the one your audit trail carries
System metadata	Read the organization ID, its slug, when it was created, and its site: the region where your organization's data is stored. Quote the ID when you contact support
Agent budgets	Set a default monthly spend target for every agent that has no budget of its own. The default applies to each such agent separately: a default of \$500 gives each one \$500 a month, not \$500 shared. Clearing it never overwrites a budget someone set on an agent
Setup	Re-run the setup wizard to review the organization's connection and profile

Save your changes with **Save Changes**. An agent's own budget is set from its **Economics** tab; see [Track an agent's spend](#).

The other settings pages each have their own guide: [users and teams](#), [single sign-on](#), [identity providers](#), [notification channels](#), the [audit trail](#), the [catalogue](#) and [licensing](#).

Tune the catalogue

Use **Settings > Catalogue** to see and adjust how OpenLatch recognizes what an agent action does and what it touches. Intent policies are written in terms of what actions do, such as reading a credential or pushing code; the catalogue is what maps a concrete tool, program or target to those terms. Every role can read it; only an admin can add or change entries.



Settings > Catalogue: the tools, programs and targets OpenLatch recognizes, with each entry's effect and source.

Read the catalogue

The catalogue has three tabs:

Tab	What its entries recognize
Tools	The tools an agent calls
Programs	The programs an agent runs in a shell
Targets	What an action touches, such as a file, a host or a repository

Each entry shows its key, its effect, its source and when it was added. Entries come from several sources: OpenLatch's own seed, learning from your agents' actions, policy authoring, or an admin. **Needs review** filters the entries that wait for a decision, and **Show OpenLatch seed** adds the read-only seed entries to the list.

Learned entries apply to an intent policy according to that policy's coverage mode; see [Write an Agent Intent Policy](#). An empty catalogue fills as OpenLatch learns the tools, programs and targets your agents use.

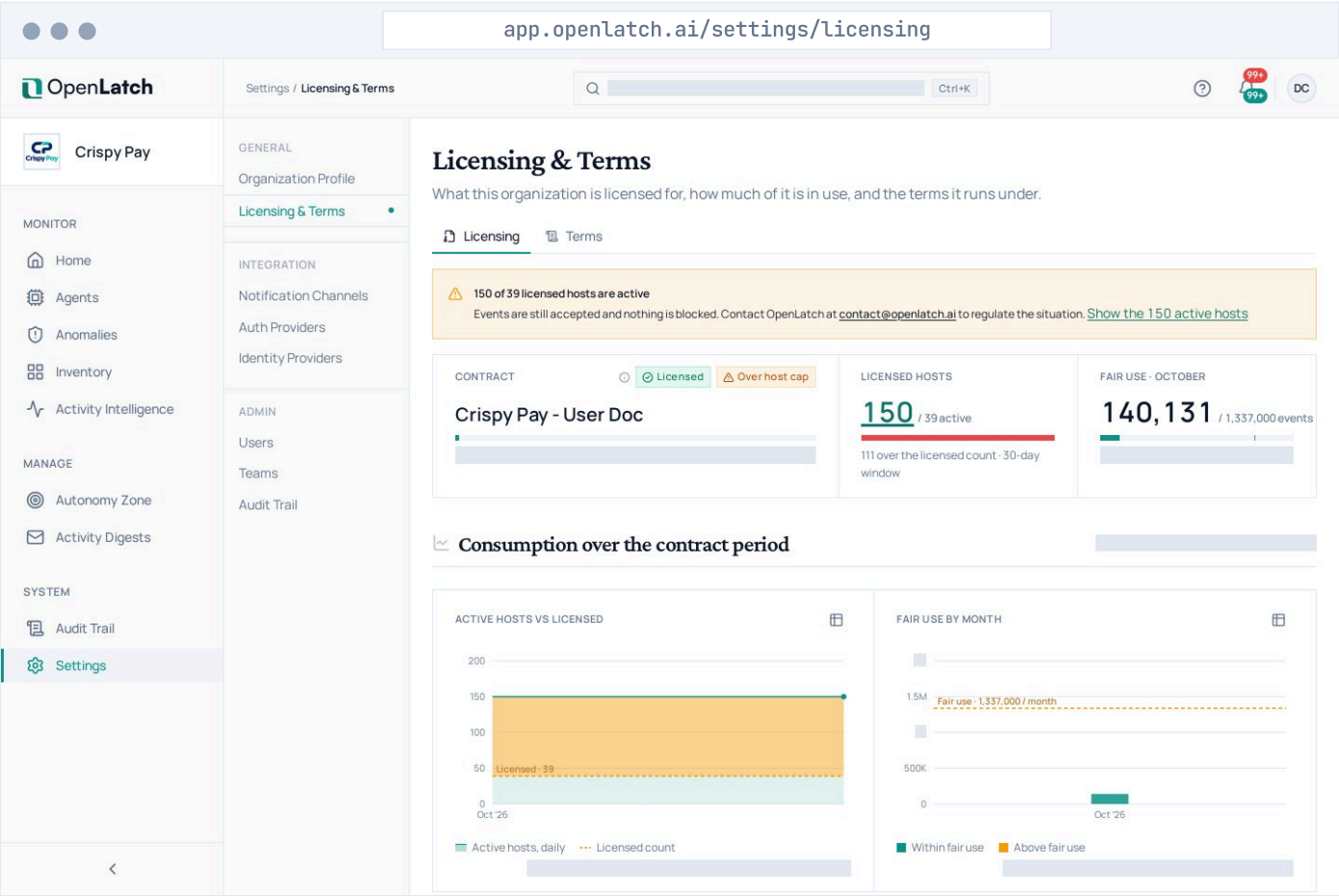
Adjust an entry

To	Do this
Add what OpenLatch does not recognize yet	Choose Add entry , and give the key and its effect
Change what a learned entry means	Choose Override on the entry
Make a seed entry stricter for your organization	Choose Narrow on it. The seed itself is read-only; your entry narrows its key

Read licensing and usage

Use **Settings > Licensing** to see what your organization is licensed for, how much of it is in use, and the terms it runs under. Retiring a host needs the admin role or higher.

Your license counts hosts: the machines that run the client. Each license also carries a monthly fair-use allowance of events.



Settings > Licensing: the contract, the licensed hosts in use and the month's fair use.

Read the meters

The **Licensing** tab opens on three meters:

Meter	What it shows
Contract	The period of your terms, and how much of it has elapsed
Licensed hosts	How many hosts are active, meaning seen in the last 30 days, against the number licensed
Fair use	The events used this calendar month, against the monthly allowance. It resets at the start of each month, in UTC

A banner appears only when something needs your attention: the terms have not started yet, the contract ends soon, more hosts are active than licensed, or the month's fair use has reached 100%. Going over the licensed hosts or the fair-use allowance blocks nothing; events are still accepted, and the banner asks you to contact OpenLatch at contact@openlatch.ai. The contract dates are different. Before your terms start, the console is available but client events are refused. Once the contract has ended, client events are refused and the console is restricted until the terms are renewed; contact OpenLatch at contact@openlatch.ai before the end date to renew.

Below the meters, the charts show active hosts against the licensed count and fair use month by month over the contract period, and the **Terms** tab holds the terms themselves.

Find a host

The licensed hosts table lists each host with its state, operating system, owner and last activity. Filter it by state, operating system, owner or last activity, or search it. Open a host to see the agents that run on it.

Retire a host

Retire a machine that no longer runs the client, so it stops counting toward your licensed hosts.

- 1 Select the host, or several hosts, in the table.
- 2 Choose **Retire host** and confirm.

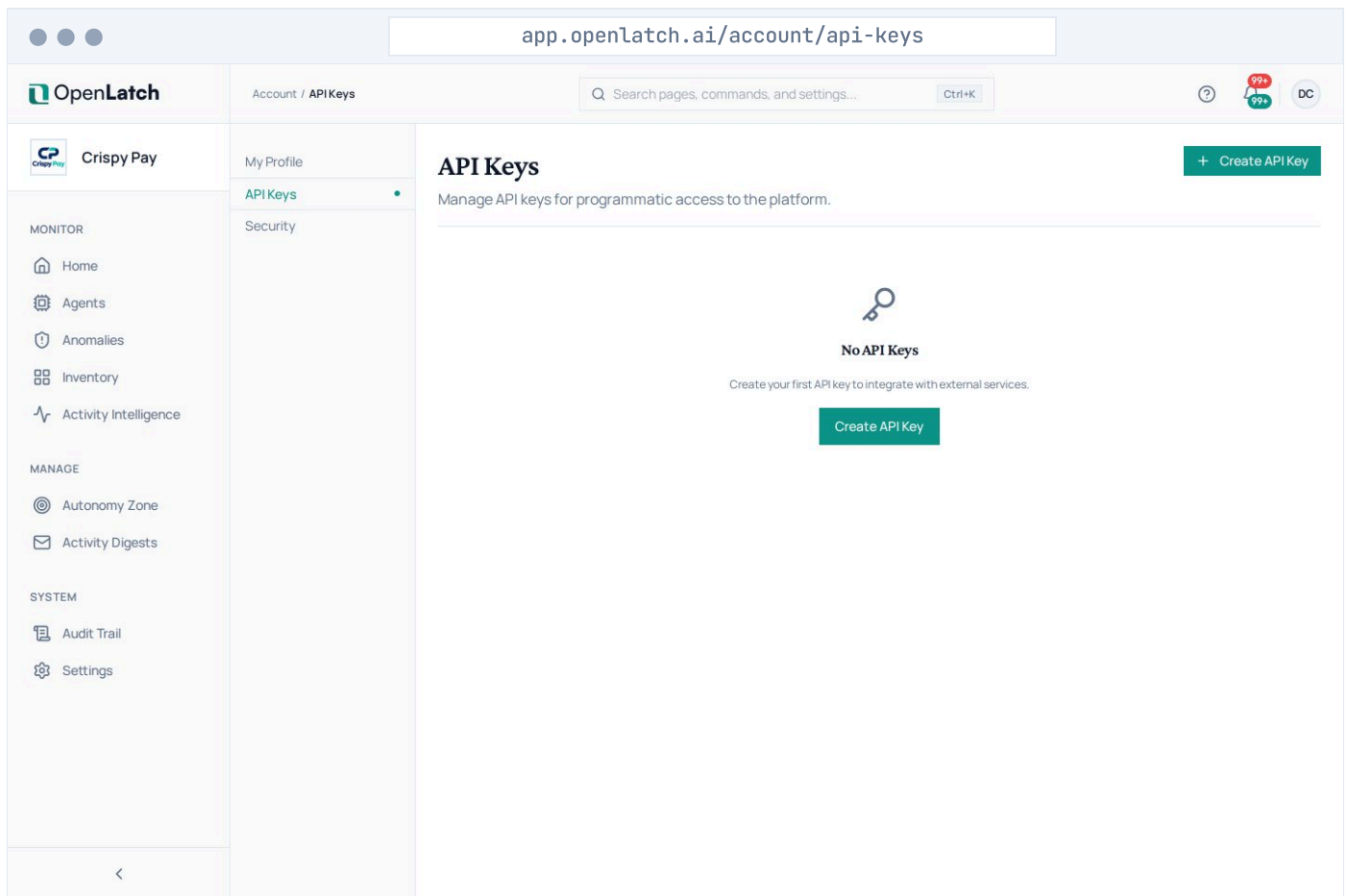
A retired host stops counting at once. Its event history is kept, and the retirement is recorded in the audit trail with your name. If the host is seen again, it counts again automatically. To remove the client from a machine, see [Operate](#).

Manage API keys

Use **Account > API Keys** to create the keys that give a program access to OpenLatch on your behalf. A key serves two purposes:

Use	How the key is supplied
Enrolling a machine without a browser sign-in, for a managed or unattended install	As OPENLATCH_API_KEY in the environment the install runs in. See Deploy the client
Calling the OpenLatch public REST API, under /api/public/v1 on app.openlatch.ai	In each request's Authorization: Bearer header

A key acts with your role in the organization. One created by an editor cannot change an identity setting, and revoking the key or lowering your role both close the same door.



Account > API Keys: each key with its name, masked value, status, creation date and last use.

Create a key

- 1 Open your account's **API Keys** page and choose **Create API Key**.
- 2 Name the key after what will use it, for example the rollout wave or the integration, and choose **Create Key**.
- 3 Copy the key value now. It is shown once, in full; after you dismiss it, the list shows only a masked value.
- 4 Store it in your organization's secret store. Never write it into a script, a package, a command line or a shared document.

Use one key per purpose: one per rollout wave, one per integration. A key you can revoke alone is a key you can revoke without disturbing anything else.

Revoke a key

Choose **Revoke** on the key and confirm. Revoking cannot be undone, and anything that uses the key loses access at once. A machine enrolled with a wave key stores that key and authenticates with it, so revoking the key disconnects the machine. Give such machines a new key first: set it as `OPENLATCH_API_KEY` and run `openlatch init --yes` again on each.

The **Last used** column tells you whether a key is still being presented before you revoke it. A key past its expiry is listed as **Expired**.

 Caution

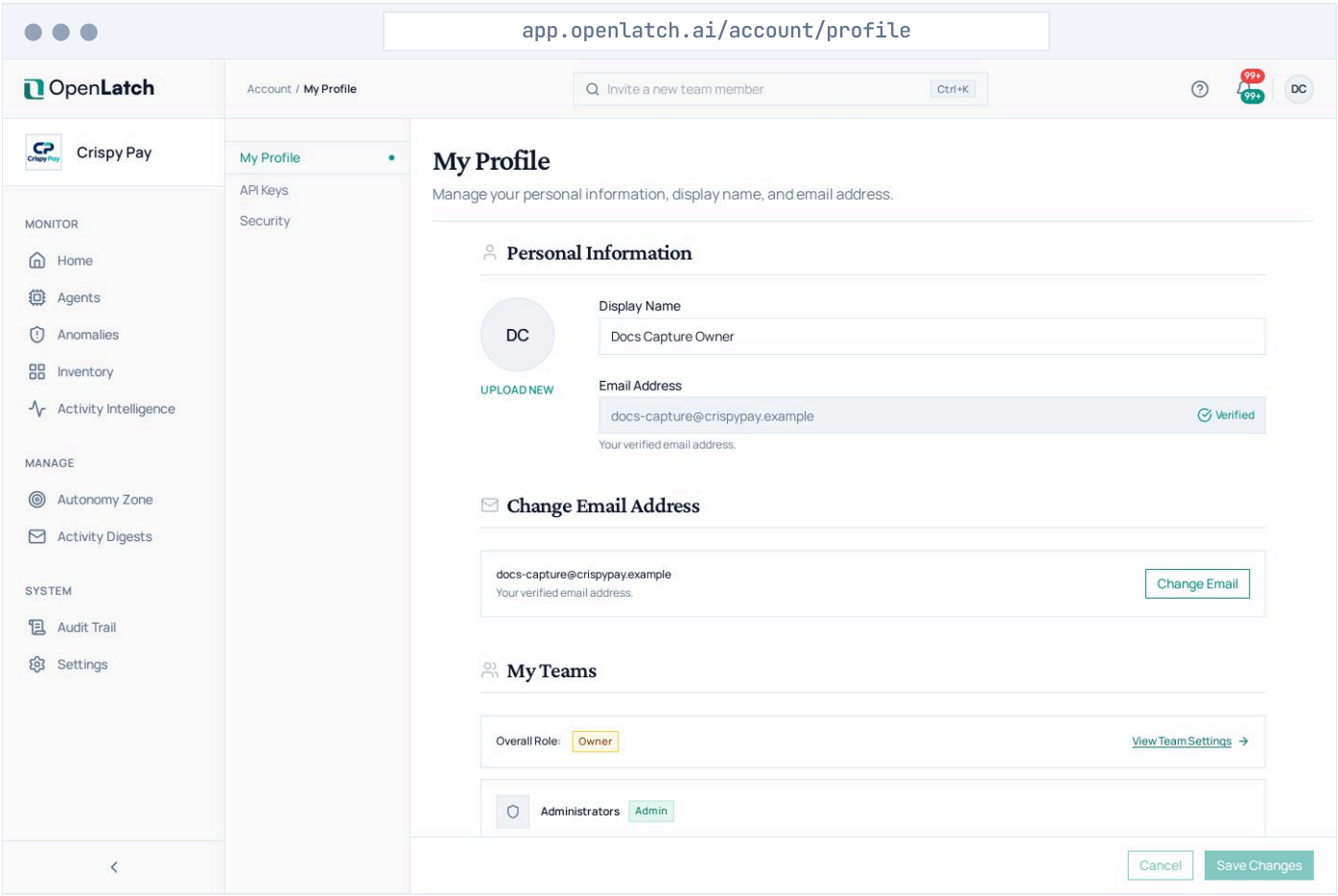
A key passed on a command line reaches shell history, process listings and configuration-management logs, and the client refuses a credential passed that way. Supply a key through the environment, from your secret store.

Manage your account

Your account settings are your own: every role can change them. Open them from your avatar in the console; **Account** opens on your profile.

Edit your profile

Account > Profile holds your personal information: your display name and your email address. Changing your email address asks for your current password.

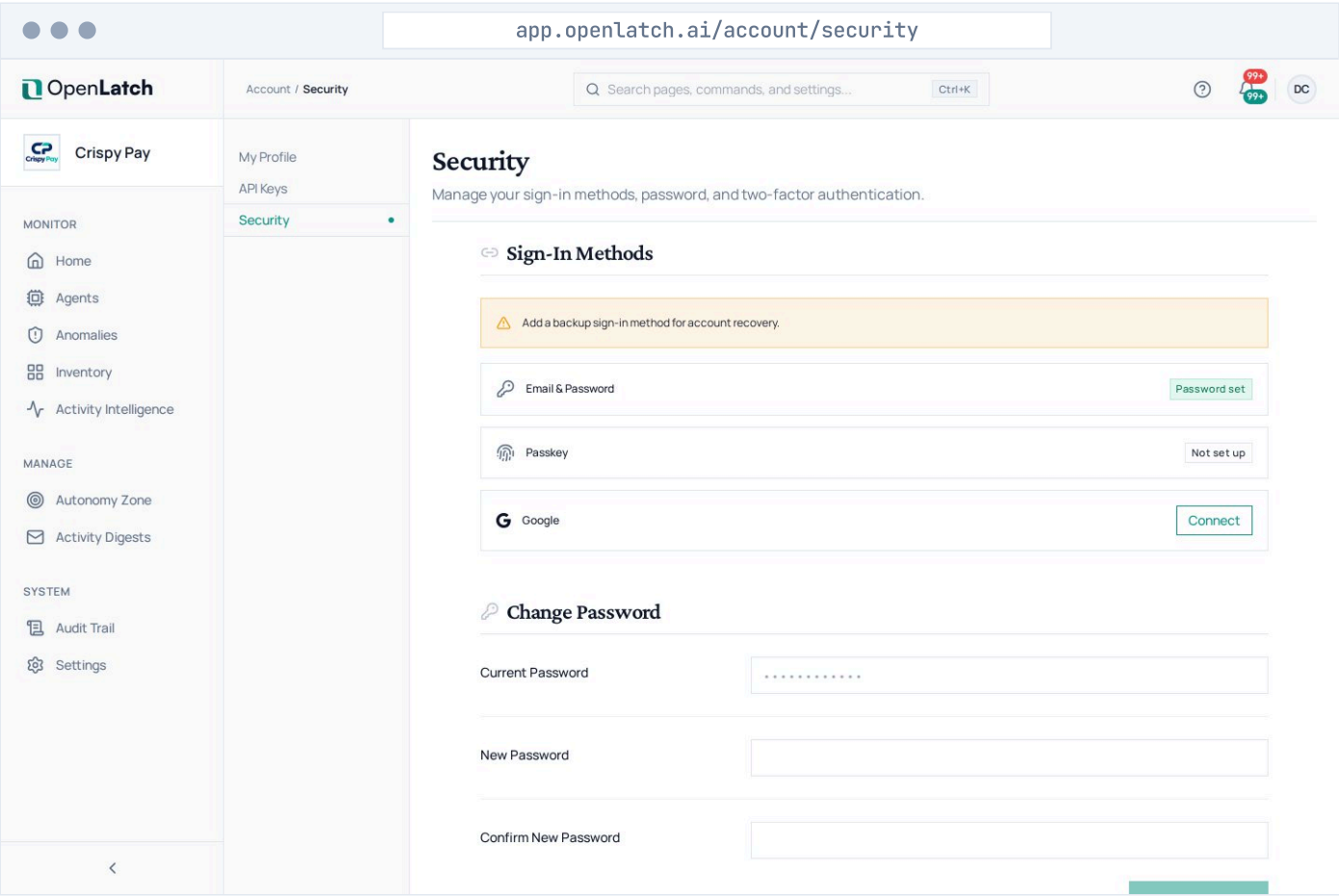


Account > Profile: your display name and email address.

Your account also lists your active sign-in sessions across devices. Sign out of any session you do not recognize.

Secure how you sign in

Account > Security manages your sign-in methods, your password and two-factor authentication.



Account > Security: your sign-in methods, password, two-factor authentication and passkeys.

Setting	What it does
Password	Change your password; the console asks for the current one first
Two-factor authentication	Adds a code from an authenticator app to each password sign-in
Passkeys	Sign in with a passkey stored on your device or in your password manager. Name each passkey so you can tell them apart
Delete account	Deletes your account. The console asks for your password to confirm

To set up two-factor authentication:

- 1 On **Account > Security**, start the authenticator app setup and confirm your password.
- 2 Scan the QR code with your authenticator app.
- 3 Save the recovery codes somewhere safe. Each code works once, and it is how you sign in if you lose the device.
- 4 Enter a six-digit code from the app to finish.

If your organization requires two-factor authentication, the console sends you to this page until you set it up; see [Set up single sign-on](#). Which sign-in methods you can use at all is your organization's choice.

Keys for programs and for enrolling machines are on **Account > API Keys**; see [Manage API keys](#).

Deploy OpenLatch

This section takes you from nothing to a working deployment: create the organization, connect your identity provider, deploy the client to every machine that runs an agent, and set the console up for the people who will use it. It is written so that you can do the work without a call.

What the section covers

You want to	Read
Understand what OpenLatch is and what runs where	Introduction · Architecture
Prepare: what to have on hand, and the one outbound destination to allow	Prerequisites · Network requirements
Install: the organization, the client on each machine, verification and rollout	Configure the organization · Deploy the client · Verify and roll out
Keep it running	Operate
Answer a security review, and the questions that follow it	Data handling and security · Deployment FAQ
Look something up	Terms and egress

How to use this section

Read it in order the first time. The introduction and the architecture page explain what OpenLatch is and what runs where; they are enough to answer a first review before anything is installed. The prerequisites and the network requirements are what you prepare in advance: the values to have on hand, and the one outbound destination to allow. The next three pages are the installation itself, from the first sign-in to a verified rollout, and [Operate](#) covers what comes after it.

[Data handling and security](#) sets out what OpenLatch handles, stores and never collects. It is the page to hand to a security review. The [Deployment FAQ](#) answers the questions that follow, and [Terms and egress](#) holds the terms and the egress table.

Every page in this section prints on its own, without the site's navigation, so you can attach it to a change request or a review.

Feedback and security reports

Send corrections and gaps to hello@openlatch.ai, naming the page. Security reports take a separate route: send them to security@openlatch.ai. OpenLatch acknowledges a security report within two business days.

Introduction

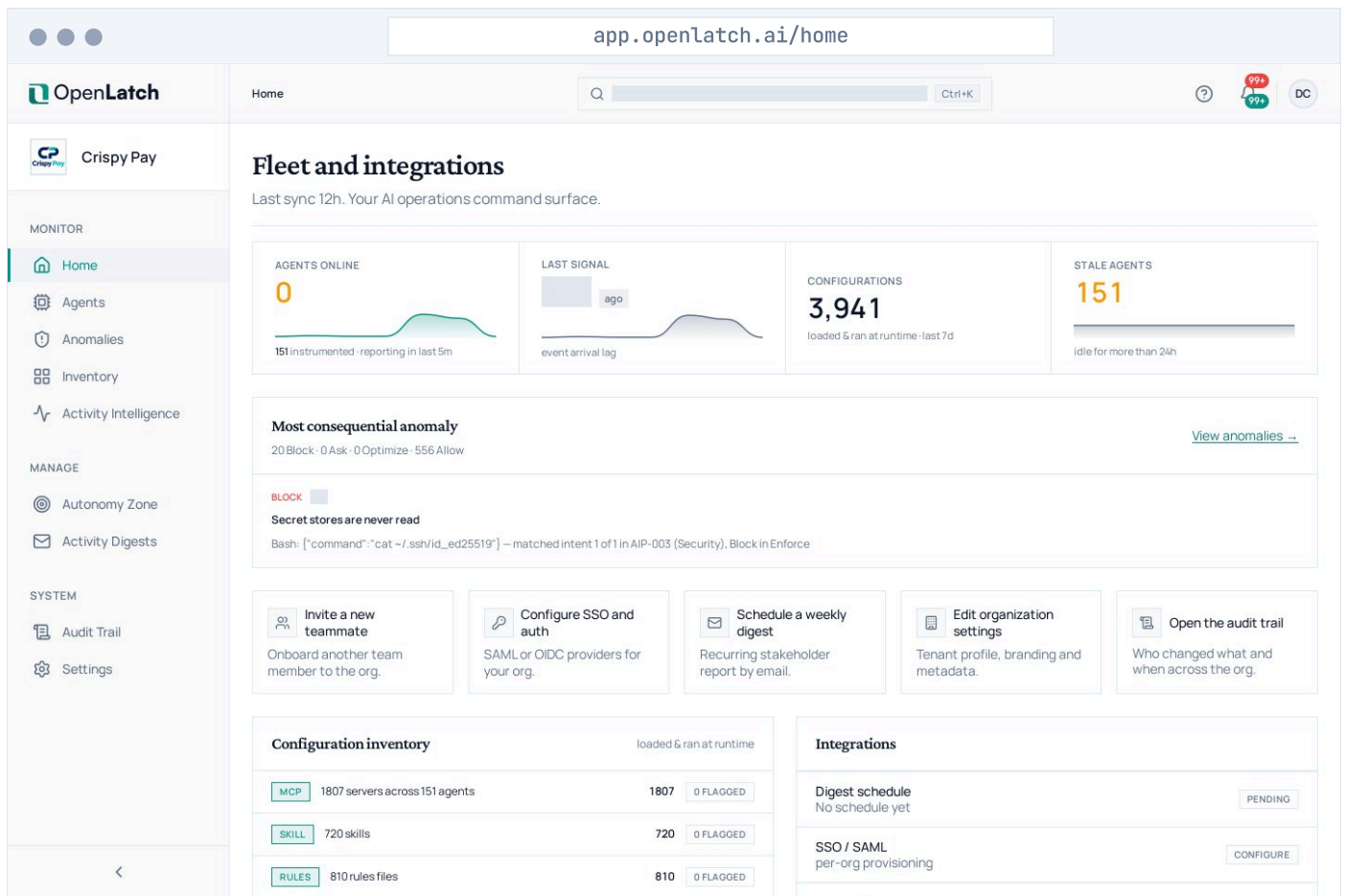
Read this page to know what you are deploying before you prepare anything.

What OpenLatch is

OpenLatch is the execution control layer for enterprise AI agents. It runs inside the agent's own lifecycle, evaluates each covered action against your organization's Agent Intent Policies, and returns a deterministic verdict before the action executes: Allow, Ask, Block or Optimize.

Deterministic has a precise meaning here. Every covered action reaches the policy decision point, and the verdict that an explicit policy produced is the verdict that is enforced. [Architecture](#) shows the path an action takes to get there.

What you end up with is one place that holds every agent in the organization, the rules each one works under, and the decisions already taken on its behalf.



The console once agents are connected and the organization has set what they may do.

The four verdicts

Verdict	What happens
---------	--------------

Allow	The action proceeds
-------	---------------------

Ask	The action proceeds and is flagged for asynchronous review. It waits for an answer only where a policy sets the per-atom opt-in for irreversible actions
-----	--

Block	The action is stopped before it executes
-------	--

Optimize	The path is changed. Applied in Enforce, measured in Monitor
----------	--

[Verdicts](#) explains each one in full.

What runs where

OpenLatch has two parts: a console you work in, and a client you install on the machines that run agents.

Part	What it is	Where it runs
The console	Holds the organization, its users and roles, the agent record and the audit trail, and is where policies are authored. It never applies a verdict itself	Delivered as a service at <code>app.openlatch.ai</code> . Hosting and data handling are on Data handling and security .
The client	One installed unit. It evaluates each covered action on the machine, returns the verdict before the action runs, and sends the record of what happened to the console. The install detects the agents present on the machine and wires each one	Each machine that runs an agent, once per machine. It installs and runs at user scope, and no administrator rights are required

What the deployment does not ask of your network:

Concern	What OpenLatch needs
Inbound ports	None. The client listens only on the loopback interface, and nothing on the machine is reachable from the network
Certificates	None. OpenLatch issues no certificate and requires none
Traffic interception	None. OpenLatch terminates no traffic and re-signs none
Network change	One outbound allowlist entry: <code>app.openlatch.ai</code> , on TCP 443
Your existing proxy	The client works behind it, including under TLS inspection when the interception certificate authority is trusted. See Network requirements

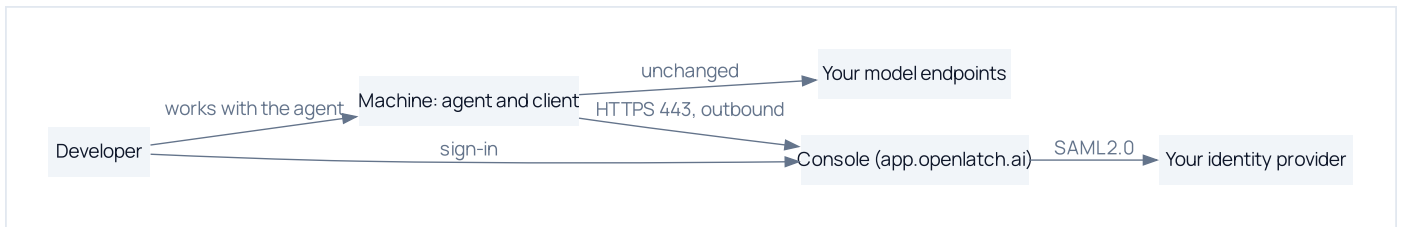


Figure 1. OpenLatch in the organization: the client on each machine, the console as a service, and the identity provider.

What you will set up

Step	What it means	Where
1 · Create the organization	Sign in, name the organization, invite the administrators and set their roles	Configure the organization
2 · Connect the identity provider	Configure single sign-on over SAML 2.0, and test the exchange before you invite anyone else	Configure the organization
3 · Connect the agents	Deploy the client to each machine, on the developer path or through your managed deployment tooling	Deploy the client
4 · Verify and roll out	Prove each machine is connected, then widen the deployment in three waves	Verify and roll out

What you need to start

Requirement	Detail
An account in the console	OpenLatch creates the organization and invites its owner. That account invites everyone else
Outbound HTTPS to <code>app.openlatch.ai</code> on TCP 443	The one destination OpenLatch requires, through your proxy. No inbound port is opened. Allowlist it by name, because there is no static address range

[Prerequisites](#) turns this into a readiness inventory, with the source of every value.

In short

- The verdict is produced on the machine, before the action runs. The console is not on the decision path.
- Policies stay in force on the machine when the console is unreachable. Records are held locally and sent when it is reachable again.
- One outbound destination is required. No inbound ports, no certificates to install, no traffic interception.
- Setup is four steps: the organization, the identity provider, the client on each machine, then verification and rollout.

Architecture

Read this page to plan around what the client puts on a machine and what crosses the machine's boundary.

What runs on the machine

Here, a machine means the operating system the agents run on. In most estates that is a developer workstation, and it is equally a virtual desktop, a build host or a container image, wherever an agent is started. If agents run there, the client is installed there.

OpenLatch installs one thing on such a machine: the OpenLatch client. It is installed once, and it covers every agent on that machine. There is nothing to place in front of the agent, no service to stand up beside it, and no second product to keep in step with the first.

Fact	Detail
What is installed	The OpenLatch client, once per operating system that runs agents
Privileges	Installs and runs at user scope. No administrator rights are required, at install or afterwards
Network posture	Listens only on the loopback interface. No inbound port is opened, and nothing on the machine is reachable from the network
Files	The client keeps its configuration and local records in one directory: <code>~/ .openlatch</code> on macOS and Linux, <code>%APPDATA%\openlatch</code> on Windows. <code>OPENLATCH_DIR</code> moves it
Startup	Starts with the user's session, under the operating system's own service manager
Where decisions happen	On the machine, before the action runs. The console is not on that path
Loss of connectivity	Keeps working. Policies stay in force, and records are held until the console is reachable again

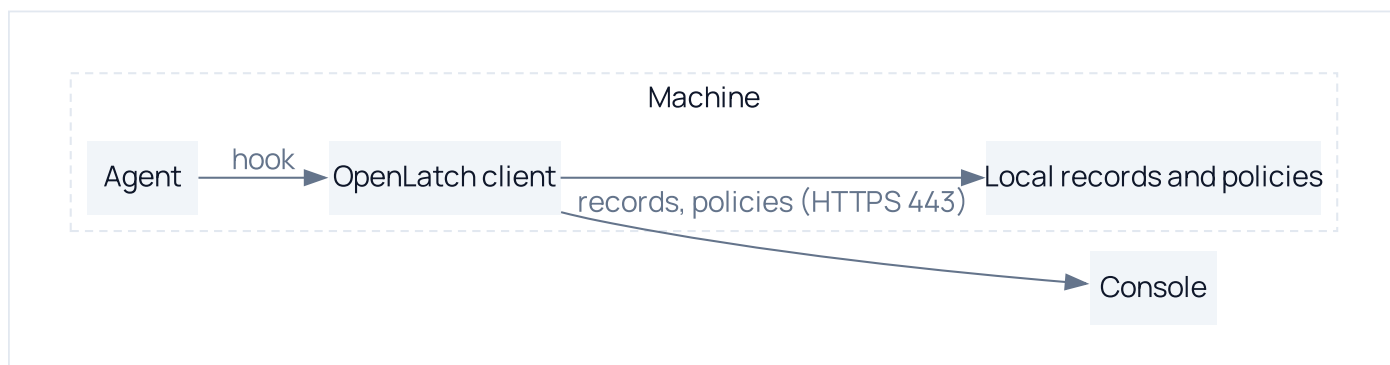


Figure 1. Deployment view: the client on each machine, and the console as a service.

The console

The console is delivered as a service at `app.openlatch.ai`. There is nothing to install and nothing to host. Your organization's data is stored separately from every other organization's.

The console holds four things you work with: the organization, its users and their roles, the agent record and the audit trail. The agent record lists every connected machine and the agents found on it, each with

an owner and a [control level](#): Enforced, Monitored or Known. The audit trail is append-only and exportable, with a retention window you set. [Configure the organization](#) is the procedure for all of this.

Policies are authored in the console, and from there they reach each machine. [Write an Agent Intent Policy](#) covers authoring.

How an action is decided

One action produces one verdict, on the machine, before the action runs. The console is not on this path.

Step	What happens	What you can observe
1 · The agent offers the action	The agent is about to run a shell command or call a tool. It hands the proposed action to OpenLatch first, and waits	—
2 · OpenLatch decides on the machine	The decision is made against the policies the machine already holds. There is no network call and no model on this path	A verdict is reached whether or not the console is reachable
3 · The verdict returns to the agent	The verdict comes back in the form the agent understands, carrying its reason. Where several policies match, the most restrictive wins	On a Block, the developer sees the refusal and its reason in the agent, and the action does not run
4 · The result is recorded	The action and its verdict are written down on the machine and sent to the console afterwards. Sending is off the decision path, so a slow console changes when an action is seen, never what was decided	The machine, its agents and their decided actions appear in the console's agent record

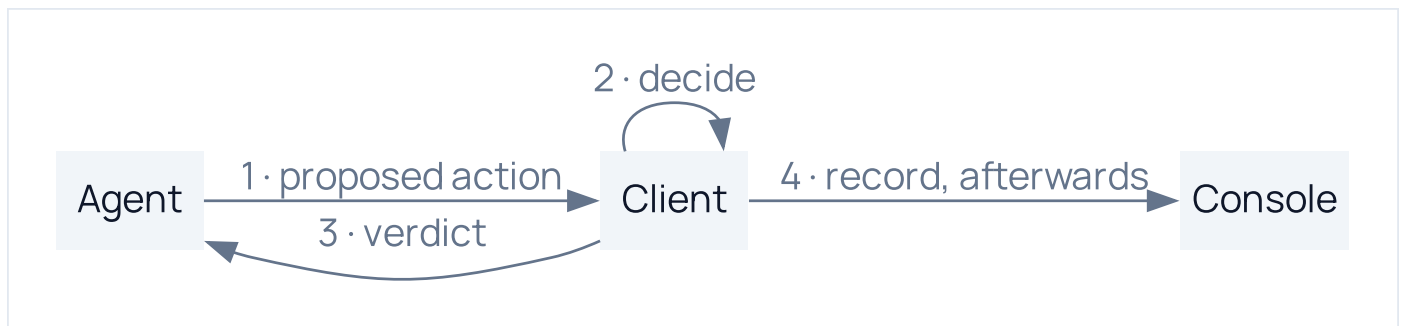


Figure 2. One action, one verdict: the decision is made on the machine before the action runs, and the record follows.

What crosses the machine’s boundary

Four flows cross the boundary of a machine running the client. Only two of them involve OpenLatch.

Flow	Direction and content
Policies	Down, from the console to the machine. The client checks at start-up and at regular intervals for policies newer than the ones it holds
Records	Up, from the machine to the console, over HTTPS and through your proxy, after the action has already been decided
Your agents’ model traffic	Unchanged. Your agents reach their model endpoints exactly as before, with their own credentials, and OpenLatch does not alter that traffic
Console sign-in	Into the console from a browser, through your identity provider

[Network requirements](#) lists every destination and covers proxies and TLS inspection.

When the network is unavailable

An unreachable console is a delivery problem, not an enforcement problem. The policies already on the machine stay in force, and the machine keeps deciding against them for as long as the outage lasts. Records that cannot be sent are held locally, up to a size cap, and go up when the console is reachable again. An outage long enough to pass the cap drops the oldest unsent records, and the client logs that it did.

Policies do not expire on their own. A policy that switched itself off after an outage would let anyone disable enforcement by unplugging a cable, so the machine keeps enforcing and the console shows that

the machine's policies have gone stale. A machine that has never received a policy allows the action and records that it had no policy: OpenLatch never blocks without policy. [Network requirements](#) carries the full table of degraded states.

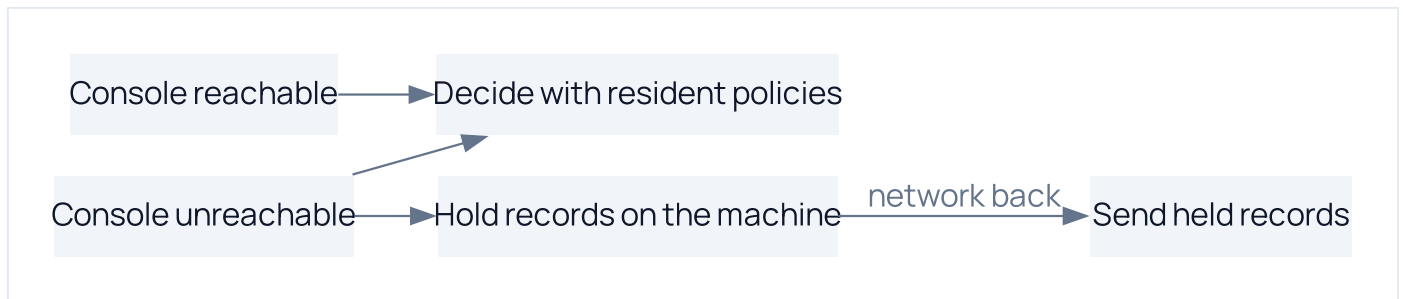


Figure 3. Deciding unplugged: the machine keeps deciding with the policies it holds, and the records wait for the network.

In short

- One client per machine that runs an agent, at user scope, with no administrator rights. It listens only on loopback and opens no inbound port.
- It keeps its configuration and local records in one directory, `~/ .openlatch` or `%APPDATA%\openlatch`, which `OPENLATCH_DIR` moves.
- Actions are decided on the machine, before they run, against policies the machine already holds. No network call sits on that path.
- The console is a service, with nothing to install or host. It holds the organization, its users and roles, the agent record and the audit trail, and policies reach each machine from it.
- An unreachable console does not stop enforcement. Policies stay in force, records are held and sent later, and a machine with no policy allows the action and records why.

Prerequisites

Fill in this page before you schedule the first rollout wave. Nothing in the deployment is hard to obtain. What stalls a rollout is discovering, on the morning of the first wave, that a decision sits somewhere else in the organization, or that the person who can approve an egress entry is away. The inventory below turns that discovery into a form somebody fills in the week before.

Fill in the readiness inventory in one working session, with everyone who holds an answer in the room. The checklist then turns those answers into rows grouped by phase of work. The last section states what the rollout costs in hours.

Readiness inventory

Each answer changes something downstream. The agent answers decide which machines belong in which wave. The endpoint answers decide the install channel. The network answers decide whether the route is discovered or handed over. The identity answers decide who administers the organization. Where the answer is genuinely none, write “none” rather than nothing, so a later reader can tell an answered question from an unasked one.

Agents

Item	Why it matters
Which agents are in use, and on which machines	The install detects the agents present on a machine and wires each one. Knowing the set in advance tells you which machines belong in which wave. Once an agent reports, the console shows its control level
Whether any agent runs somewhere other than a personal workstation	The client is installed once per machine that runs an agent, wherever that machine sits. A build host or a shared workstation running an agent is in scope on the same terms as a laptop

Endpoints

Item	Why it matters
Operating system mix and machine counts per operating system	The client installs and runs at user scope on every supported platform, with no administrator rights. The floors are macOS 13, Windows 10 1809 or Windows Server 2019, Ubuntu 20.04 and Debian 10, or later
Processor architectures in the fleet	Packages exist for macOS on Apple silicon and Intel, Linux x64 and arm64, and Windows x64. Windows on ARM64 and musl-based Linux distributions such as Alpine are not supported
The packaging or configuration channel your organization already operates	Waves 2 and 3 deliver the install through a channel that already reaches these machines. That channel also carries the wave credential from your secret store
Whether any wave machines are shared by several people	Install and local state are per user. Each user on a shared machine runs an install of their own

Network

Item	Why it matters
Proxy in the path: yes or no	Decides whether the route is discovered at install or handed to it. See Network requirements
Proxy scheme: HTTP CONNECT, SOCKS5 or proxy auto-config	All three are supported. A proxy auto-config file is evaluated by the operating system on Windows and macOS. On Linux, set the route explicitly instead
Proxy authentication: none, Basic, Kerberos (Negotiate) or NTLM	None, Basic and Negotiate are supported. NTLM is not supported, by design, so an NTLM-only estate plans a Negotiate or unauthenticated path for the client before the first wave
TLS inspection: yes or no, and where the interception certificate authority lives	Inspection needs no bypass. The interception root belongs in the operating system trust store, or is supplied as a PEM file at install
Whether the model endpoints your agents already reach stay reachable, and are exempt from response buffering and streaming inspection	Your agents reach those endpoints exactly as before. A device that buffers a streamed response makes agent output arrive in one burst

Identity

Item	Why it matters
The SAML 2.0 identity provider your organization uses	The console authenticates an organization through SAML 2.0. Microsoft Entra ID, Okta, PingOne and Active Directory Federation Services connect this way. Connecting it needs an administrator on both sides
Allowed email domains	Allowed domains and invitations together gate who can join the organization
Directory for owner lookup: Microsoft Entra ID, Okta, Google Workspace, PingOne or Active Directory	A read-only lookup resolves an agent's owner to a person. It never creates or removes users. This connection is optional
Who holds the owner and admin roles in the console	Console roles run owner, admin, editor, viewer. Invitees start as viewer, and identity settings need admin

Secrets

Item	Why it matters
Where the per-wave OPENLATCH_API_KEY is held, and how it reaches a machine without appearing in a script	One key per rollout wave, created in the console and revoked there when the wave completes. Managed installs read it from your secret store as an environment variable. It is never written into a script, a package or a command line

Pilot

Item	Why it matters
The ten wave-1 machines and their owners	Wave 1 runs the developer install path. Named owners let you confirm each machine against the console's agent record
How you will reach OpenLatch during the rollout	Agree the support route before the first wave rather than during it. See Operate

 Note

An NTLM-only proxy estate is the one answer in this inventory that can block a rollout outright. Ask the team that manages your proxy early, and read the proxy support matrix in [Network requirements](#) before the first wave is scheduled.

Consolidated checklist

Every row names where the value comes from and the page it blocks. The four phases are independent, so the console rows and the network rows run in parallel through the same week.

Prepare the console

#	Item	Where it comes from	Blocks
1	Organization created in the console	Your OpenLatch contact creates the organization and invites its owner. Accept the invitation from the invited address. Later sign-ins start at <code>app.openlatch.ai</code> with an email step whose browser check needs <code>challenges.cloudflare.com</code>	Configure the organization
2	SAML connected and tested	Your identity provider's admin console, for the entity ID, sign-on URL, and certificate or metadata URL. Entered in the console under Settings > SAML SSO	Configure the organization
3	Allowed email domains set	The same SAML settings as row 2	Configure the organization
4	Owner, admin, editor and viewer roles assigned	The console, under Settings > Users . Invitees start as viewer	Configure the organization
5	Directory chosen for owner lookup, optional	Your directory's admin console. The lookup is read-only	Configure the organization
6	Audit trail retention set	The console, under Settings > Audit Trail : 90, 180, 365 or 730 days, or a custom window, with auto-purge and export	Configure the organization
7	Webhook or Slack notification channel, optional	The console, under Settings > Notification Channels	Configure the organization
8	Wave credential created and stored	The console, under your account's API Keys , from a signed-in browser session. Stored in your organization's secret store	Configure the organization

Prepare the network

#	Item	Where it comes from	Blocks
9	Egress allowlist entry for <code>app.openlatch.ai</code> on TCP 443 approved	Your change process, against the table in Network requirements	Deploy the client
10	The model endpoints your agents already reach stay reachable, and are exempt from response buffering and streaming inspection	Your proxy or secure web gateway configuration	Deploy the client
11	Proxy scheme and credential path confirmed	Your proxy configuration, read against the support matrix	Deploy the client
12	Interception certificate authority available as a PEM file, if the operating system does not already trust it	Your certificate authority's own export	Deploy the client

Prepare the machines

#	Item	Where it comes from	Blocks
13	The agents you intend to cover are installed and signed in on those machines	The machines themselves, confirmed with the person who uses each one	Deploy the client
14	Install reviewed for the managed route	The managed route in Deploy the client . It reads <code>OPENLATCH_API_KEY</code> from your secret store and never carries it inline	Deploy the client

Prepare the rollout

#	Item	Where it comes from	Blocks
15	Wave-1 machine list with owners	Your endpoint inventory, cross-checked with the agent owners	Verify and roll out
16	Developers briefed that some agents ask them to confirm the connection once	Your own rollout communication, sent before the wave starts	Verify and roll out
17	Support route agreed for the rollout	Operate	Verify and roll out
18	Rollback owner named	Your own decision: the person who can pause a wave and remove the client from a machine	Verify and roll out

Effort and timing

The client rollout runs in three waves of two weeks each. Wave 1 installs on about ten machines through the developer path. Wave 2 takes about fifty machines through the channel you already operate. Wave 3 extends to the full perimeter and ends in the decision to run it as normal operations.

Effort is front-loaded into the first week, when the organization is set up and the route is proven on one machine of each operating system in the fleet. After that, each wave repeats a procedure that already worked.

Phase of work	Wave 1	Wave 2	Wave 3	Typical effort
Prepare the console	Connect SAML, set allowed domains and audit retention, invite people, create the wave credential	Rotate the wave credential	Hand over the operating routine	A day in the first week, then a few hours a week
Prepare the network	Approve the egress entry and confirm the proxy route	Review the managed route	No further change	Half a day in the first week
Prepare the machines	Prove the install on one machine per operating system, then run ten machines through the developer path	Carry the install in the channel you already operate and run the wave	Extend the same channel to the remaining machines	Minutes per machine after the first
Confirm and operate	Confirm each installed machine appears in the console's agent record with an owner	Weekly checkpoint and a managed-route review	Weekly checkpoint, then the decision to run it as normal operations	A weekly checkpoint through the rollout

In short

- Complete the readiness inventory in one working session before the first wave is scheduled. Every blank cell is a decision somebody will make under time pressure later.
- One egress entry is mandatory: `app.openlatch.ai` on TCP 443, outbound. Everything else in the path is your agents' own traffic, which is unchanged.
- An NTLM-only proxy estate blocks the rollout. Plan a Negotiate or unauthenticated path for the client.
- No administrator rights are needed on a machine. Install and local state are per user, so a machine shared by several people needs an install per user.

- One `OPENLATCH_API_KEY` per rollout wave, held in your secret store and revoked in the console when the wave completes. It never appears in a script, a package or a command line.

Network requirements

Use this page as the change request for your network team. The client needs one outbound destination, reached through your proxy, and opens nothing inbound.

One outbound dependency

The client runs on the machine that runs the agent, inside the agent's own lifecycle. It reaches OpenLatch for three things: to collect the policies it decides against, to deliver the record of what it decided, and to authenticate. All three go to one destination over one port. The optional product telemetry and crash reports go to that same destination, so OpenLatch adds no other destination, whether you leave them on or turn them off.

Property	Value
Direction	Outbound only. The client opens no listening port on a network interface, and nothing on the machine becomes reachable from the network
Destination	<code>app.openlatch.ai</code> , on TCP 443
Transport	HTTPS. TLS is verified against the operating system trust store, plus an optional certificate bundle merged on top of it. Verification cannot be disabled
Through your proxy	Yes: HTTP and HTTPS CONNECT, SOCKS5, and proxy auto-config where the operating system supports it, with Basic and Kerberos (Negotiate) authentication
IP allowlisting	Not supported. There is no static address range, so allowlist by fully qualified domain name and SNI
Certificates	None issued by OpenLatch, and none required from you

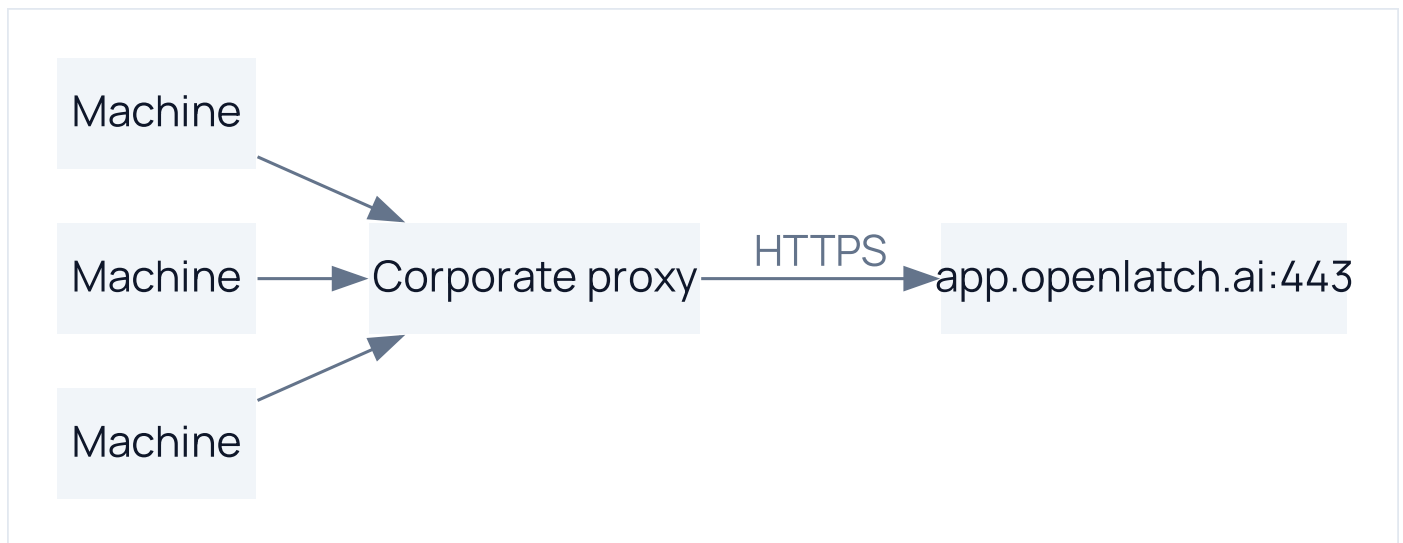


Figure 1. A fleet of machines behind a corporate proxy: one outbound destination, nothing inbound.

Egress requirements

This is the table to paste into a change request. The first row is what OpenLatch needs to run, and it is the only destination a machine running the client needs from OpenLatch. The second is needed only while you install or update the client; an estate that mirrors packages internally points it at its own mirror instead. The third is needed only when you install with one of the two scripts. The fourth concerns browsers only.

Destination	Port	Direction	Protocol	Purpose	Allowlisting
<code>app.openlatch.ai</code>	TCP 443	Outbound	HTTPS	Policies, decision records, authentication, and the optional product telemetry and crash reports	By name and SNI. There is no static address range
Your package registry: <code>registry.npmjs.org</code> , unless you point the client at a mirror	TCP 443	Outbound	HTTPS	Installing the client, and downloading an update when you allow one	By name, or by the name of your internal mirror. Needed only during an install or an update
<code>openlatch.ai</code>	TCP 443	Outbound	HTTPS	Downloading the install script, <code>install.sh</code> or <code>install.ps1</code> , and the install report it sends	By name. Needed only when installing with a script
<code>challenges.cloudflare.com</code> , from the browser of anyone who signs in to the console	TCP 443	Outbound	HTTPS	The check the email step runs at sign-in	By name, for browsers only. No machine running the

Destination	Port	Direction	Protocol	Purpose	Allowlisting
					client
					contacts it

 Note

On an estate that denies egress by default, the second and third rows are the ones that are forgotten. The install then fails before the client has ever reached the console, and the failure looks like a broken product rather than a closed route.

The administrator’s browser

Sign-in to the console starts with an email step, and that step runs a short check in the browser that the visitor is a person. The check is served from `challenges.cloudflare.com`, so the browsers your administrators sign in with must reach that name on TCP 443. A browser that cannot reach it fails the check every time. This requirement applies to browsers only: `app.openlatch.ai` stays the one destination a machine running the client needs.

Your agents’ own traffic

Your agents reach their model endpoints exactly as they did before OpenLatch was installed. OpenLatch neither terminates nor re-signs that traffic, and it adds no destination of its own to it. Those endpoints must stay reachable, and they must be exempt from response buffering and streaming inspection, so that a streamed response is not held back on its way to the person waiting for it.

A device that buffers the response, or scans it as a whole body, holds the stream until it completes and then releases it in one burst. The request still succeeds; the developer sees an agent that appears to stall and then prints everything at once.

For proxy products that steer by application rather than by destination, the process name is `openlatch` on macOS and Linux and `openlatch.exe` on Windows. Use that name in a custom application definition.

Proxy support matrix

Answer per control. A “No” row is a design decision rather than a gap, and each one names the path to take instead.

Network control	Supported	How it is handled	Notes
Unauthenticated HTTP or HTTPS CONNECT proxy	Yes	Discovered at install, or handed to the install	The common corporate shape. Whatever route works is recorded on the machine
Basic authentication	Yes	The credential is prompted for on a challenge and kept in the operating system credential store	Never written to a file in plain text
Kerberos (Negotiate)	Yes	Uses the logged-on identity, through the operating system's own mechanism	The service principal defaults to the proxy host's HTTP service and can be overridden at install. A missing or expired ticket is reported with its remedy
NTLM	No	Not supported, by design	Microsoft deprecated NTLM, and a client that quietly downgrades to it works against what it was deployed for. Use Negotiate, or an unauthenticated path for the client
Proxy auto-config (PAC or WPAD)	Yes, on Windows and macOS	The URL is handed to the operating system's own resolver	Refused on Linux, where no auto-config resolver is available to the client. Set the route explicitly there
SOCKS5	Yes	A socks5: // or socks5h: // route	Name resolution stays local with socks5 and moves to the proxy with socks5h
Transparent proxy	Yes	No configuration on the machine; the traffic is intercepted in the path	Pair it with the TLS inspection row where the device re-signs the connection

Network control	Supported	How it is handled	Notes
TLS inspection with a corporate certificate authority	Yes	The interception root goes in the operating system trust store, or is supplied as a PEM bundle at install	The bundle is merged on top of the operating system roots, never a replacement, and verification is never disabled
mTLS client certificates to the proxy	No	—	The client presents no client certificate. Use one of the supported authentication schemes
IP-based allowlisting of OpenLatch egress	No	—	There is no static address range. Allowlist <code>app.openlatch.ai</code> by name and SNI

Configure the route

How the route is found

On most corporate machines the install finds the route without being told. It probes what is already configured, then walks the operating system's own settings, including proxy auto-config on Windows and macOS. It asks a question only when none of those reach the console, once, on a terminal. An unattended install that cannot find a route stops with a report naming every candidate it tried, rather than silently going direct.

Whatever route made the install work is recorded on the machine. This matters: the client starts with the user's session and does not inherit the installing shell's variables, so a route that lived only in an environment variable would be lost at the next restart.

Hand the route to the install

Where discovery cannot work, or where a managed install runs with no terminal to ask on, hand the route to the install directly. Supply the certificate bundle at the same time if the operating system does not already trust the interception root.

Hold the enrollment back with `OPENLATCH_NO_INIT` and run it as a second step, so that the flags reach the command that performs it. Export the variable before the script runs, so that the shell that reads the script carries it too.

On macOS and Linux:

```
export OPENLATCH_NO_INIT=1
curl -LsSf https://openlatch.ai/install.sh | sh
~/local/bin/openlatch init --proxy http://proxy.corp:8080 --ca-bundle /etc/ssl/corp-root.pem
```

On Windows:

```
$env:OPENLATCH_NO_INIT = "1"
powershell -ExecutionPolicy Bypass -c "irm https://openlatch.ai/install.ps1 | iex"
& "$env:LOCALAPPDATA\openlatch\bin\openlatch.exe" init --proxy http://proxy.corp:8080 --ca-bundle C:\corp\corp-root.pem
```

On a machine that already has Node.js, `npx @openlatch/client init` takes the same flags in a single step:

```
npx @openlatch/client init --proxy http://proxy.corp:8080 --ca-bundle /etc/ssl/corp-root.pem
```

A bypass list can be supplied alongside the route, in the usual grammar. Traffic that never leaves the machine does not go to the proxy and needs no bypass entry.

Where policy forbids a machine from ever reaching out without the proxy, the client can be configured to refuse a direct connection when no route resolves. It then reports a route failure instead of quietly going around your controls. [Client configuration](#) lists the settings.

Credential handling

A proxy credential lives in the operating system's credential store: Credential Manager on Windows, Keychain on macOS, Secret Service on Linux, and an encrypted file on a headless host that has none. It is never written to a configuration file in plain text. Every place a proxy URL is printed or logged shows it as `user:*****@host`. A proxy URL that carries a password on the command line is refused, because a command line is readable by any process on the machine.

TLS inspection

The connection to the console works under inspection, and needs no bypass. The client makes ordinary request-and-response HTTPS calls, verifies against the operating system trust store, and pins no certificate. Put the interception root in the trust store where your estate already does that, or supply a PEM copy of it at install. The bundle is merged on top of the operating system roots, so a machine keeps trusting everything it trusted before.

Your agents' own model endpoints are the exception, and the request there is narrow: exempt them from response buffering and streaming inspection, and leave the rest of your inspection policy alone.

Caution

Never disable certificate verification to work around an inspecting proxy. The client offers no switch for it. A failed handshake is reported with the remedy attached: trust the interception root on the machine, or supply it at install.

Offline and degraded behavior

The client refreshes its policies from the console in the background. Every decision is made on the machine, against the policies already there, so no network call sits on the decision path. That is what makes a verdict survive a broken route.

Where the machine holds policy, it keeps enforcing whatever the network does. Where it holds none, it allows the action and stamps the record to say the decision was made without policy. Nothing fails silently.

Situation	What the machine does	What you see
The console is unreachable	Keeps enforcing the policies it holds and retries in the background. Records are held on the machine	The machine stops checking in. <code>openlatch</code> status reports it as disconnected, with policies present
The policies on the machine have not been refreshed for some time	Keeps enforcing them. Policies do not expire	A staleness warning. Staleness is reported, never acted on
The wave credential is revoked or rotated, and the console rejects it	Stops collecting updates, keeps enforcing what it holds, and resumes on its own once a valid credential is in place	An authentication warning on the machine, and no further check-ins in the console
Your organization has published no policy yet	Keeps whatever it already holds in force	The machine reports as connected, with nothing to collect
An update arrives damaged or in a form the machine does not recognize	Rejects it and keeps the previous policies. A damaged update never takes effect	An integrity warning on the machine, with the previous policies still in force
The local copy of the policies is altered on the machine	Discards it. The machine runs without policy until the next successful refresh	Every verdict is stamped to show it was decided without policy, and the machine reports the loss
No policy has ever reached the machine	Returns Allow: a machine never blocks without policy	Every verdict is stamped to show it was decided without policy
The client is not running	The action proceeds and is not held up	<code>openlatch</code> status reports the client as not running, and the console shows no recent activity for the machine

Situation	What the machine does	What you see
Records cannot be delivered	Retries, then holds them on the machine and sends them when the console is reachable again	A gap in the console that closes when the machine reconnects. The hold is bounded: past its size cap the oldest unsent records are dropped, and the daemon log records an OL-1204 warning
You set every policy to Monitor from the console	Each machine applies the change on its next refresh, and records what it would have done	The change, in the console, with no access to a machine needed. Decisions keep being recorded

Verify the route

Run these commands on the first machine before the wave, and again on any machine that reports a connection problem.

```

openlatch status
openlatch doctor
openlatch system proxy test

```

Command	What it proves
openlatch status	Whether the machine is connected to the console, whether policies are present, and which agents are wired. It answers even when the route is broken, which is exactly when you need it
openlatch doctor	The whole machine, with a cause and a remedy against each finding. It reports the hop through your proxy separately from the connection to the console, so a dead proxy and an unreachable console never look like the same failure
openlatch system proxy test	The route itself, probed one hop at a time. It stops at the hop that fails and names it

[Deploy the client](#) covers installing it, and [Verify and roll out](#) covers full verification of a machine.

In short

- Approve one entry for the machines: `app.openlatch.ai` on TCP 443, outbound. There are no inbound ports.
- The browsers that sign in to the console also need `challenges.cloudflare.com` on TCP 443. No machine running the client contacts it.
- Allowlist by name and SNI, not by address. There is no static address range.
- Your agents reach their model endpoints as before. Keep those endpoints reachable and exempt from response buffering and streaming inspection.
- TLS inspection needs no bypass for the console connection. Basic, Kerberos, SOCKS5, transparent and auto-config proxies are supported. NTLM is not, so plan a Negotiate or unauthenticated path.
- A broken route degrades delivery, not enforcement. The machine keeps deciding with the policies it holds, records wait until the console is reachable, and both states show in `openlatch status`.

Configure the organization

Set up the console for your organization: the account, single sign-on, users and roles, owner lookup, retention and notifications, and the credential the client uses to enroll. Everything here happens in the browser, takes about half an hour, and touches no machine. Installing the client is [Deploy the client](#).

Before you start, have the [readiness inventory](#) filled in, and make sure your browser reaches `challenges.cloudflare.com`.

Sign in and finish the organization

OpenLatch creates the organization for you, names the address you gave your OpenLatch contact as its owner, and sends that address an invitation. The welcome wizard is where you finish setting it up.

- 1 Accept the invitation.** Open the link in the invitation email. It takes you to your organization's own sign-in page and admits the invited address as the owner. Complete it with any method the page offers: a passkey, a magic link, an email address and password, or a social sign-in.

Expected: the console opens on your organization, and your role reads owner.

- 2 Sign in from `app.openlatch.ai`.** Every later sign-in starts with one step: your email address. That step runs a short check in your browser that the visitor is a person; it usually asks nothing, and at most asks for one click. The console then finds the organization your address belongs to and opens its sign-in page with the address filled in.

An address that no organization knows is told so. A check that fails every time usually means the browser cannot reach `challenges.cloudflare.com`; see [Network requirements](#).

- 3 Work through the welcome wizard.** It runs in four steps: name the organization, invite teammates, connect an agent, and a summary of what you completed and what you skipped. Use the first step to set the legal or business-unit name you want on the audit trail. Connecting an agent is [Deploy the client](#); skipping it here costs nothing.

Expected: the wizard closes on the console, with no agents enrolled yet.

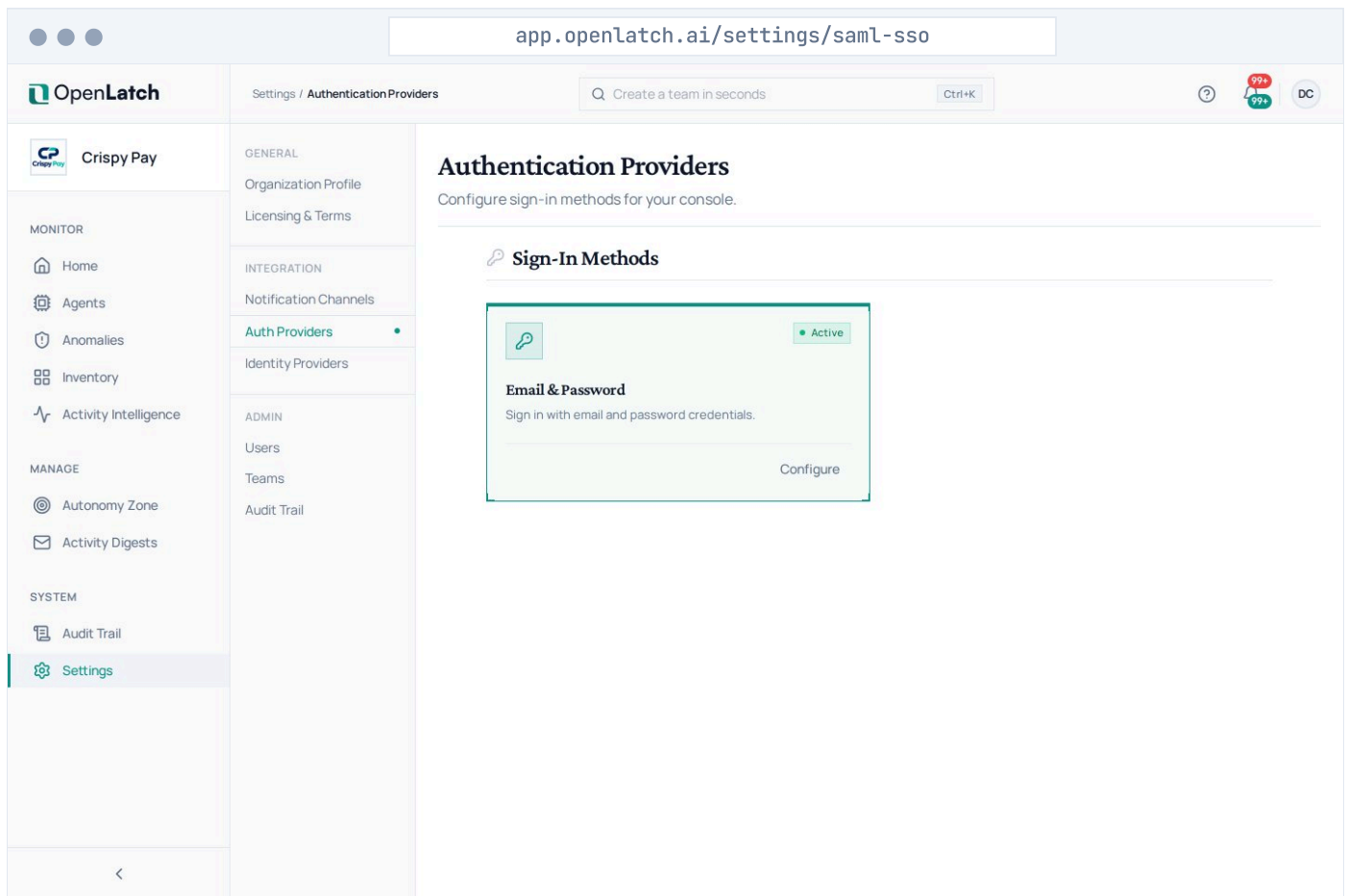
Connect single sign-on

Single sign-on uses SAML 2.0, configured per organization. Microsoft Entra ID, Okta, PingOne and Active Directory Federation Services all connect through the same screen. Identity settings need the admin role or higher.

- 1 **Open Settings > SAML SSO and add a provider.** The page shows the service-provider details that whoever administers your identity provider will need. Have your identity provider's metadata at hand.
- 2 **Enter the identity provider details.** The console asks for the identity provider's entity ID, its sign-on URL, its signing certificate, and the email domains allowed to sign in. You can supply a metadata URL instead of pasting the certificate. Pick your identity provider from the list first, and the page shows a setup guide written for that product.

The assertion your identity provider sends must carry the user's email address and display name; the setup guide names the exact fields.
- 3 **Save, then run the connection test, then sign in.** The test runs against the saved configuration, not what is still on screen. It checks that the identity provider's metadata URL or sign-on URL answers; it does not perform a SAML sign-in. Then sign in through SAML yourself, in a private browser window, before you tell anyone else to.

Expected: the test reports a pass, and your own SAML sign-in reaches the console.
- 4 **Decide who may join.** Two gates control membership. An allowed email domain lets anyone with an address in that domain register into the organization. An invitation admits one named person. Keep the allowed domains narrow, and rely on invitations for everyone outside them.



The SAML SSO settings, with the service-provider details and the values your identity provider supplies.

[Set up single sign-on](#) covers the SAML settings and the other sign-in methods in more detail.

Invite the team and assign roles

Four roles run in a strict order of privilege. Every invitee starts as a viewer, so a role is granted deliberately rather than removed after the fact.

Role	What it can do
owner	Everything an admin can do, plus ownership of the organization itself. Keep it with the person accountable for the deployment
admin	Identity settings, policies, client credentials, retention and notification channels. Grant it to the few people who run the deployment
editor	Policies and agent ownership overrides
viewer	Reads agents, executions, decisions and the audit trail. Enough for everyone else, including a developer checking that their own machine enrolled

- 1 **Invite by email address** under **Settings > Users**. Each invitation produces a registration link tied to that address. Invite the people running the rollout, not the whole engineering organization.

Expected: each invitee appears in the user list as pending, with the viewer role.

- 2 **Raise the roles that need raising.** Grant admin only where identity settings, credentials or retention must change. Grant editor to whoever maintains policies and the agent record. Leave the rest at viewer.

The screenshot shows the 'Users' settings page in the OpenLatch application. The page is titled 'app.openlatch.ai/settings/users'. On the left, there is a sidebar with the OpenLatch logo and a navigation menu. The main content area is titled 'Users' and includes a sub-header 'Manage team members and invitations.' Below this, there is a search bar and several filter buttons: '+ Status', '+ Team', '+ Email', '+ Created', and '+ Last Login'. A table lists the users, with columns for Name, Email, Teams, Role, Status, Last Login, and Created. The users listed are: Docs Capture Owner (Owner, Active), Rohan Kapoor (Policy Editor, Active), Camille Lefèvre (Policy Editor, Active), Marcus Washington (Policy Editor, Active), Ingrid Solberg (Admin, Active), User Documentation (Admin, Active), and Luc Delsalle (Owner, Active). At the bottom of the table, it says 'Showing 1 to 7 of 7 users'.

The Users settings: each member and the role they hold.

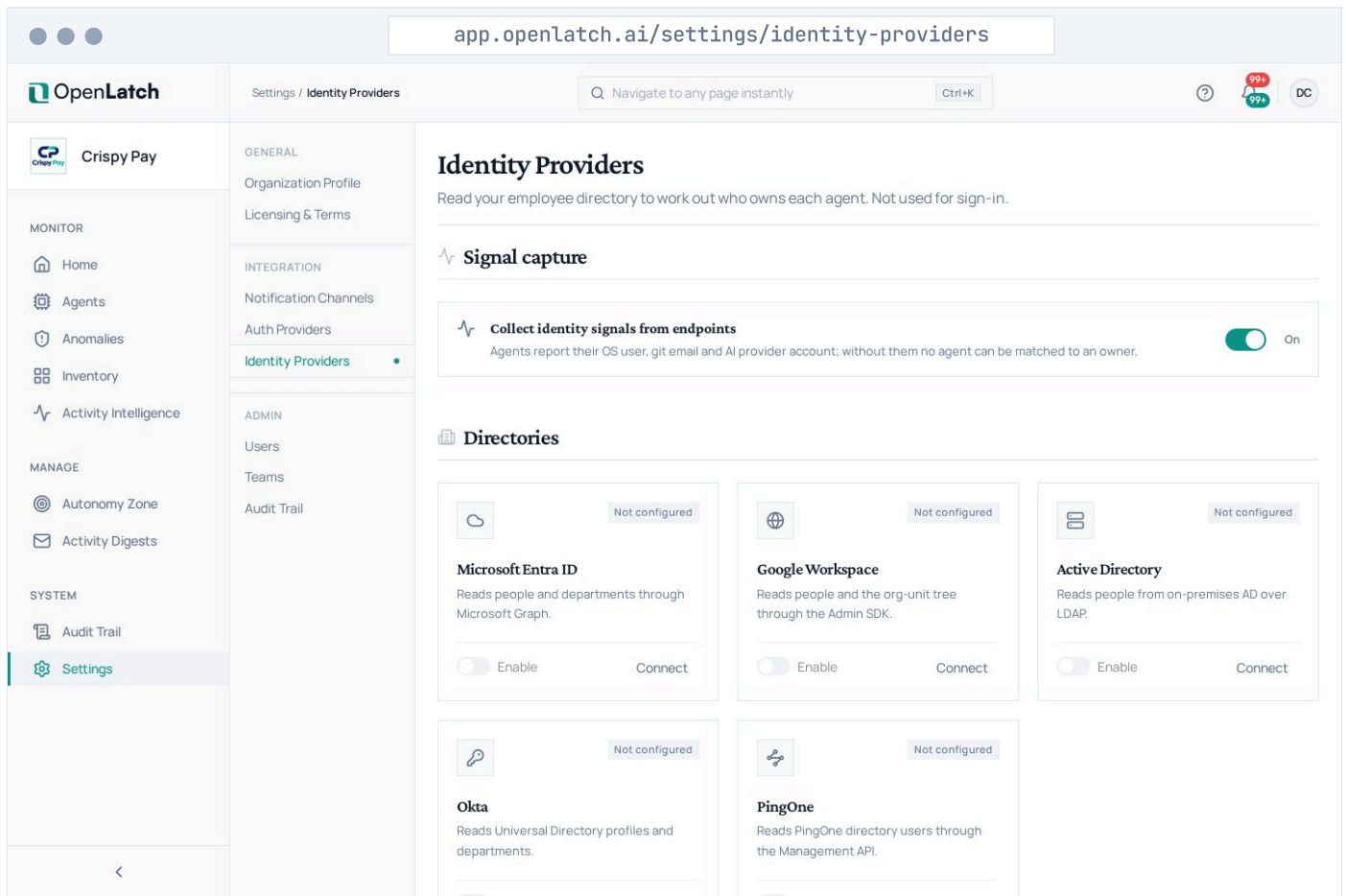
Connect a directory for owner lookup

This step is optional, and you can do it after the first machines are enrolled. A directory connection resolves the owner of each connected agent, so the agent record shows a person rather than an operating system account name. It is read-only and is not used for sign-in.

Ownership is matched from two sides, and both must be present. The machines report the signals that identify who is working, and the directory turns those signals into a named person. The signal side is a switch on the same page, **Collect identity signals from endpoints**, on by default. With it off, no agent can be matched to an owner.

- 1 Open **Settings > Identity Providers** and connect one directory: Microsoft Entra ID, Google Workspace, Active Directory, Okta or PingOne. Each is a read-only source; none creates users, and none copies the directory into OpenLatch.
- 2 Resolve one person you know is in the directory. An organization with no directory connected is a supported state, and an agent whose owner cannot be resolved is recorded as unattributed rather

than guessed.



Identity Providers: the switch that lets machines report who is working, and the directories that turn it into a named person.

Set retention and notifications

Set these two before the rollout starts; a security review will ask about both.

- 1 **Set the audit trail retention** under **Settings > Audit Trail**. The audit trail is append-only and hash-chained. Choose a retention window, decide on auto-purge, and set the export format your evidence process expects.

Setting	Values
Retention	90, 180, 365 or 730 days, or a custom window. The console recommends 365
Auto-purge	On or off. With it on, records past the retention window are removed every 24 hours
Export	CSV or JSON

- 2 **Add a notification channel** under **Settings > Notification Channels**. Point it at the channel the rollout already works in, such as a Slack channel or an HTTPS webhook, and send a test.

Expected: the test delivery arrives.

The screenshot displays the 'Audit Trail' settings page in the OpenLatch console. The page is titled 'app.openlatch.ai/settings/audit-trail'. The left sidebar shows the 'Settings' menu item selected. The main content area is divided into sections: 'Retention Policy' with a 'Retention Period' of 365 days (recommended), 'Auto-Purge' which is enabled, 'Export Preferences' where CSV is the chosen default format, and 'Hash Chain Integrity' with a 'Verify Integrity' button. The bottom right corner has 'Cancel' and 'Save Configuration' buttons.

The Audit Trail settings: retention window, auto-purge and export format.

Create the client credential

The client authenticates to the console with an API key. You create it from a signed-in browser session, on your account's **API Keys** page, and revoke it from the same page.

- 1 **Create one key per rollout wave**, named after its wave. A key per wave means a key that has to be revoked affects one wave only.

Expected: the key value is shown once, in full, at creation. Afterwards the list shows its name, a masked value, when it was created and when it was last used.

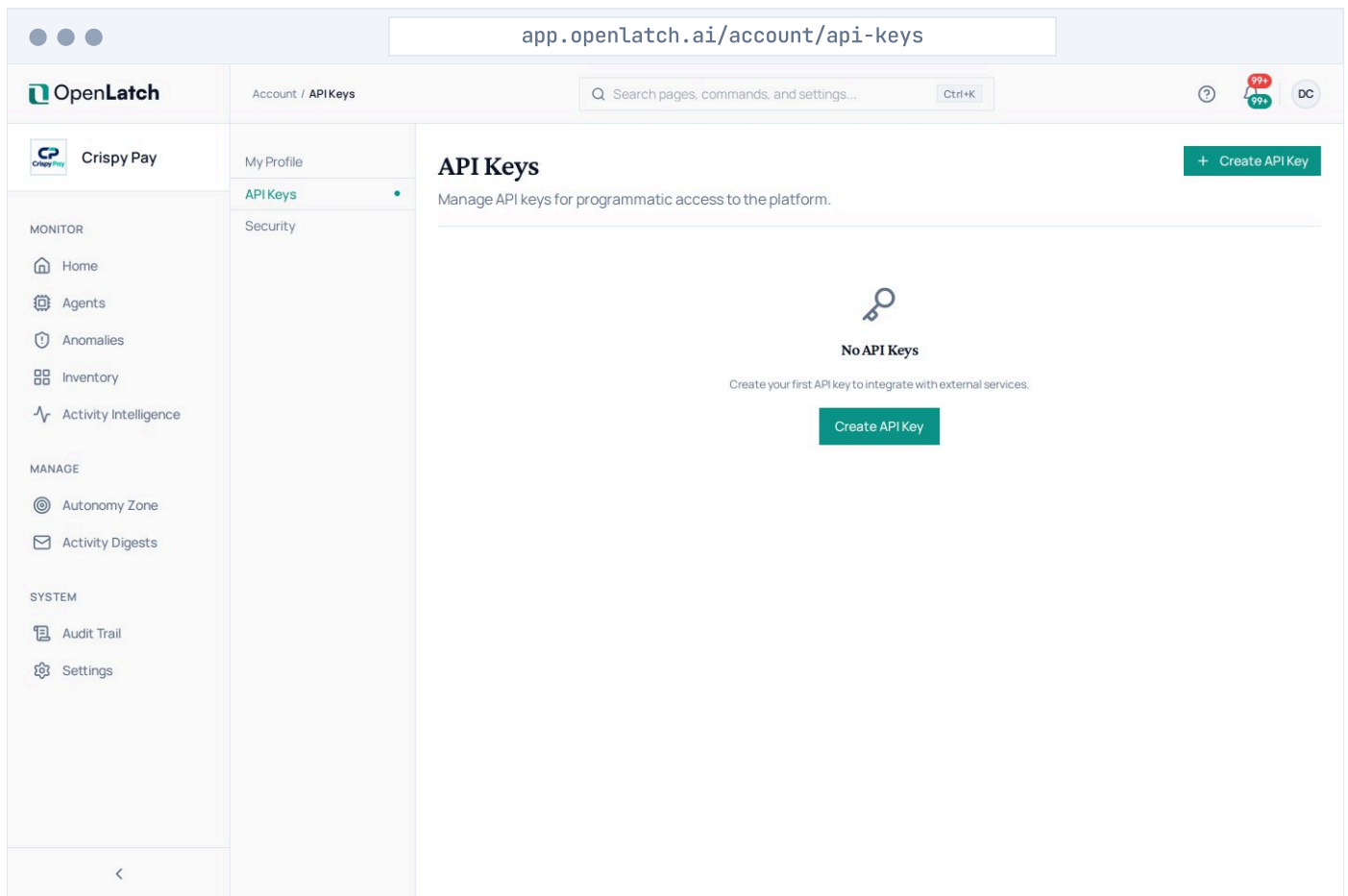
- 2 **Store it in your organization's secret store**. The managed install reads the key from the environment as `OPENLATCH_API_KEY`. It is never written into a script, a package or a command line.

- 3 **Keep it until the wave's machines hold a replacement**. Each machine stores the key it was enrolled with and authenticates with it, so revoking a wave key disconnects every machine of that wave. Before you revoke one, give its machines a new key: set it as `OPENLATCH_API_KEY` and run `openlatch init --yes` again on each machine. The last-used date tells you whether a key is still being presented before you revoke it.

Expected: the key is listed as revoked, and anything still presenting it, an install or a machine, is refused.

Caution

A key on a command line reaches shell history, process listings and configuration-management logs. The client refuses a credential passed that way. Supply it through the environment, from your secret store.



API Keys: each key listed with a masked value after creation.

[Manage API keys](#) covers keys in more detail.

In short

- OpenLatch creates the organization and invites its owner; the welcome wizard is where you name it.
- SAML 2.0 is configured per organization and checked by a test that names the stage that failed. Allowed email domains and invitations are the two gates on membership.
- Roles run owner, admin, editor, viewer, and every invitee starts as viewer. A directory connection is optional, read-only, and resolves an agent's owner.
- Set the audit trail retention, with auto-purge, before the rollout starts, and point notifications at the channel already in use.
- Create one client credential per rollout wave, supply it through the secret store as `OPENLATCH_API_KEY`, and revoke it when the wave completes.

Deploy the client

Install the client on every machine that runs an agent. The client is the decision point in the agent's execution path: installing it connects every agent already present on the machine, and from then on each covered action is decided on the machine, before it runs, even when the console is unreachable.

Before you start, check these items:

#	Item	Where it comes from	Blocks
1	The organization exists, and the people running the pilot can sign in	Configure the organization	Every route
2	One API key per rollout wave, held in your secret store	Your account's API Keys page	The managed routes only
3	At least one agent installed on the target machine and signed in to its own account	The agent's own installer	Every route
4	Egress approved, with streamed model responses exempt from buffering and inspection	Network requirements	Every route
5	The proxy scheme in force on the target subnet: direct, Basic, Negotiate, PAC, or TLS inspection with its root certificate	The team that manages your proxy	The managed routes

An agent that is absent when the install runs is not connected. Install the agents first, or run the install again afterwards; running it again is idempotent, so a repeat is never destructive.

Choose an install route

Four routes reach the same end state. Pick by where the credential comes from and how many machines the route must reach, not by operating system. Most estates use two: the developer path for the pilot, and one managed route for the rest.

Route	Best for	Reach	Credential
Developer path	Wave 1: a pilot team enrolling itself	About ten machines	Browser sign-in
Endpoint management script, user context	Estates already managed by Intune or Jamf	The full estate	Wave key from the secret store
Configuration management or a pipeline step	Linux fleets and mixed estates with existing automation, such as Ansible	The full estate	Wave key from a vault lookup
Container image	Build machines and continuous integration runners	One enrollment per job	Runtime secret, mounted per job

Three commands install the client. Which one you run depends on what the machine already has, not on the size of the estate:

Machine	Command
macOS or Linux, no runtime required	The <code>install.sh</code> script, below
Windows, no runtime required	The <code>install.ps1</code> script, below
Any machine that already has Node.js	<code>npx @openlatch/client init</code> . Node.js is what npx needs; the client is a native binary and needs no runtime

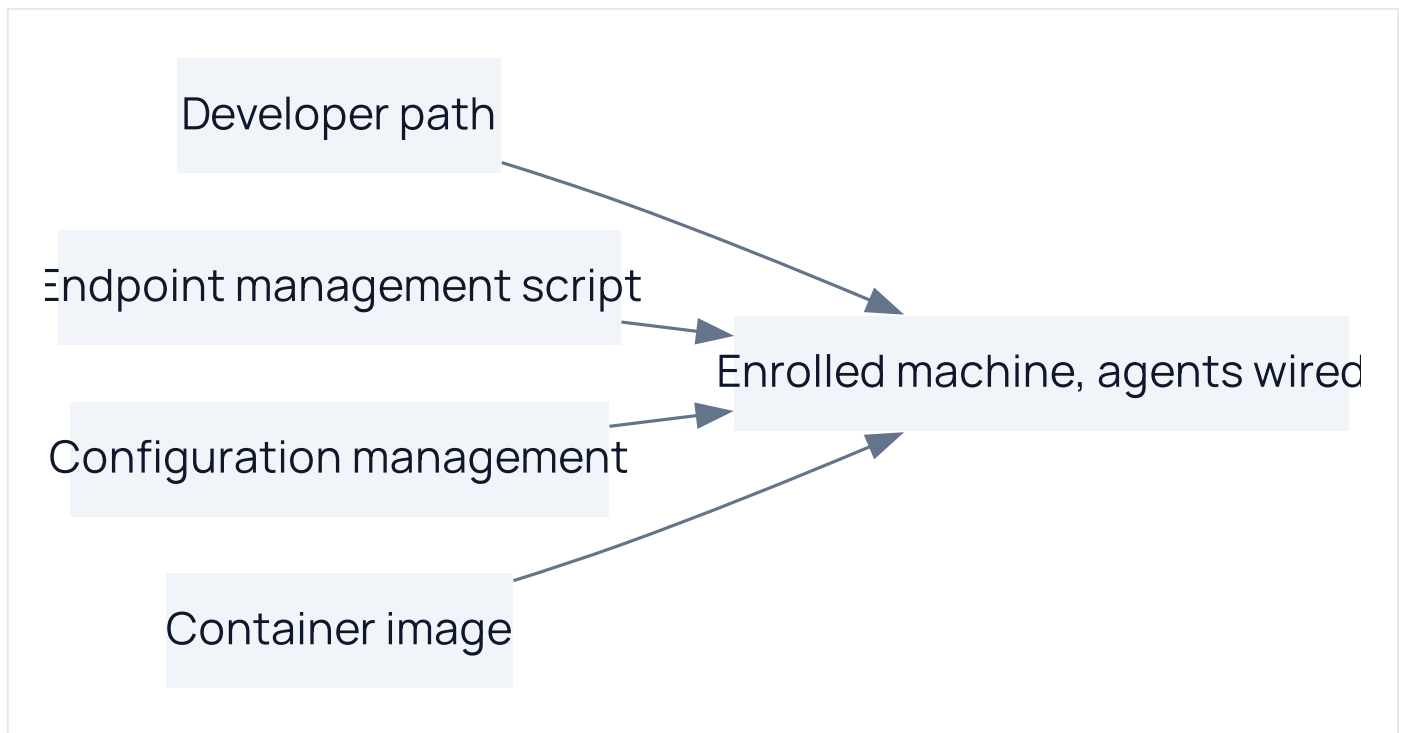


Figure 1. Four install routes, one end state: an enrolled machine with every agent wired.

Install a machine

One command enrolls a machine. The install scripts carry the client, so the machine needs no runtime of its own. Run the line for the platform in front of you.

On macOS and Linux:

```
curl -LsSf https://openlatch.ai/install.sh | sh
```

On Windows:

```
powershell -ExecutionPolicy Bypass -c "irm https://openlatch.ai/install.ps1 | iex"
```

On a machine that already has Node.js:



```
npx @openlatch/client init
```

All three reach the same end state, so a team that already runs one of them keeps it. The scripts install the client under the user's own home directory, add that directory to the PATH, and say that the current shell will not see it until it is restarted. They then run `openlatch init` by its full path, so the enrollment finishes in the same run. Where your estate does not reach the public package registry, `OPENLATCH_NPM_REGISTRY` points both the script and the client at your mirror.

What the install does

Step	What happens	Expected
Check the route out	The egress check runs first, so a blocked route fails before anything on the machine changes. Nothing is left half-installed	The route out is reported, with the source that set it
Connect each agent	Every agent installed on the machine is connected, through that agent's own configuration convention. Existing content in those files is preserved	One line per agent found, and an explicit line where none was found
Authenticate	The order is OPENLATCH_API_KEY from the environment, then a credential already in the operating system's credential store, then browser sign-in. A network error keeps a stored credential; only a rejection triggers a new sign-in	The account and organization printed back, or a browser window on an interactive terminal
Ask about product telemetry	Telemetry is opt-out, and the choice is offered before any agent configuration is written. It carries command names, durations and error information, never content	A single prompt, and the choice in the closing report
Start with the user's session	The client registers with the operating system's own service manager, at user scope. A failure here is not fatal, because a headless session and a restricted scheduler are both survivable	The report states that the client will start again after a logout or a reboot
Prove it, then report	The install confirms that the client is running, that each connected agent answers, and that the console can be reached	A closing report naming anything that needs attention, with the remedy beside it

Note

Some agents ask the developer to confirm the connection once, the first time they run after installation. Until then the agent is connected on paper and not in practice. `openlatch status` shows which agents are still waiting, and [Control levels](#) explains how each agent receives a verdict. A machine is not finished until the developer has opened the agent once.

Confirm the install

```
openlatch status
openlatch doctor
```

A machine is installed when `openlatch doctor` reports no problem and the machine appears in the console's agent record with its agents and an owner. [Verify and roll out](#) covers that check machine by machine and across a wave.

Behind a corporate proxy

Every outbound call the client makes travels one egress route. Loopback traffic on the machine never travels it. The route is resolved setting by setting, so a flag can pin the proxy URL while the certificate bundle comes from elsewhere.

Rank	Source	Use it for
1	Flags on the <code>openlatch init</code> command	Pinning the route at enrollment time
2	<code>OPENLATCH_PROXY</code> and its sibling variables	The only channel allowed to carry a proxy credential
3	The client's own configuration, written at install	What the client reads when it starts with the session, where no shell variables exist
4	The standard <code>https_proxy</code> family of variables	An estate that already exports them for every tool
5	Operating system discovery	Zero-configuration enrollment on a managed desktop
6	Direct	The last resort. It can be refused, so that an unresolved route fails instead of going direct

Whatever made the install work is written into the client's own configuration, with the source it came from. The client starts with the user's session and inherits none of the installing shell's variables, so a route that lived only in an environment variable would leave the machine going direct after the session ended.


```
openlatch init --proxy http://proxy.corp:8080 --ca-bundle /etc/ssl/corp-root.pem
openlatch init --proxy http://proxy.corp:8080 --proxy-auth negotiate --proxy-spn
HTTP/proxy.corp
```

The certificate bundle is merged on top of the operating system trust store, never replacing it, and verification is never disabled. Negotiate uses the logged-on identity, with the service principal name defaulting to HTTP/ followed by the proxy host. NTLM is refused by design, so an NTLM-only estate needs a proxy-side change before enrollment.

A credential passed inside `--proxy` is refused, because command-line arguments are readable by every process on the host. Supply it through `OPENLATCH_PROXY` or answer the prompt on a terminal. Either way the password moves into the operating system credential store, never lands in a file on disk, and is masked in every log line.

Caution

On a proxied host, loopback addresses must be in the proxy bypass list of the shell that launches your agents. Some agents receive that entry automatically at install. Where an agent has no place to receive it, export `NO_PROXY=127.0.0.1,localhost` in the shell that launches it. Without it, the agent's traffic to the client leaves for the corporate proxy instead of staying on the machine, and the proxy answers with an error.

Managed installs for fleets

Caution

Use one API key per rollout wave, read from your secret store and revoked in the console when the wave completes. Never write the key into a script, a package or a command line. Every example below puts `OPENLATCH_API_KEY` into the environment the install runs in. Where the management platform can hand a value to a script, as Jamf does through parameter 4, use that channel; where it cannot, as on Intune, the script fetches the key from the secret store as its first action. Treat the body of a deployment script as readable: a management platform holds it in plain text and may write it to logs on the device.

A managed install runs the same script with three additions:

Addition	Effect
OPENLATCH_INSTALL_YES	Makes the install unattended
OPENLATCH_NO_INIT	Holds back the enrollment the script would otherwise run at the end, so that your script runs <code>openlatch init --yes</code> itself. With <code>--yes</code> the enrollment never prompts: it fails and names what it needed
OPENLATCH_API_KEY in the environment	The credential arrives from your secret store rather than through a browser

Run `openlatch doctor` as the last line of every script and collect its output, so that a machine that installed but is not fully connected stays visible instead of counting as done.

Endpoint management script, user context (Windows)

```
# Platform script, user context. Runs silently and cannot prompt.
$ErrorActionPreference = "Stop"

# The platform has no field that sets an environment variable, so the script
# fetches the wave key itself, as its first action. Replace this call with the
# one your secret store provides.
$env:OPENLATCH_API_KEY = Get-WaveKeyFromSecretStore
if (-not $env:OPENLATCH_API_KEY) {
    Write-Output "the wave key could not be read from the secret store"; exit 1
}

$env:OPENLATCH_INSTALL_YES = "1"
$env:OPENLATCH_NO_INIT     = "1"

powershell -ExecutionPolicy Bypass -c "irm https://openlatch.ai/install.ps1 | iex"
if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

$openlatch = "$env:LOCALAPPDATA\openlatch\bin\openlatch.exe"
& $openlatch init --yes
if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }
& $openlatch doctor
exit $LASTEXITCODE
```

`$ErrorActionPreference` does not stop the script when a native command fails, so the script checks `$LASTEXITCODE` after each one and returns it, and Intune reports the failure. Intune runs scripts silently and does not support interactive installation, which is why the install runs unattended and `--yes` is mandatory. The management extension fetches assignments on its own schedule, so plan a wave in days rather than minutes. A user-context script needs a signed-in user, and platform scripts do not reach devices that are only registered with Microsoft Entra ID; to reach personally registered devices, use the Win32 app route with an install behavior of User, which the no-elevation design makes possible.

No Intune setting defines an environment variable for a platform script. That is why the secret-store call is the script's first action. Passing the key as an argument is not an option: the client refuses a credential in its own arguments.

Endpoint management script, user context (macOS)

Jamf passes its first custom value in parameter 4, which keeps the key out of the script body. The policy needs no packaging step of its own, because the install registers itself under the user's home directory. Scope the policy to a user-level trigger so the enrollment runs as the developer rather than as root.

```
#!/bin/zsh
set -eu

# Parameter 4 carries the wave key, or replace this line with a secret-store call.
export OPENLATCH_API_KEY="${4:?wave key was not supplied}"

export OPENLATCH_INSTALL_YES=1
export OPENLATCH_NO_INIT=1

curl -LsSf https://openlatch.ai/install.sh | sh

"$HOME/.local/bin/openlatch" init --yes
"$HOME/.local/bin/openlatch" doctor
```

Configuration management or a pipeline step

```

- name: Install the OpenLatch client
  ansible.builtin.shell: curl -LsSf https://openlatch.ai/install.sh | sh
  args:
    creates: "{{ ansible_env.HOME }}/.local/bin/openlatch"
  environment:
    OPENLATCH_INSTALL_YES: "1"
    OPENLATCH_NO_INIT: "1"

- name: Enroll the machine
  ansible.builtin.command: "{{ ansible_env.HOME }}/.local/bin/openlatch init --yes"
  environment:
    # Or a vault lookup, so the value never lands in the play or in a fact.
    OPENLATCH_API_KEY: "{{ lookup('env', 'OPENLATCH_API_KEY') }}"

- name: Report what the machine looks like
  ansible.builtin.command: "{{ ansible_env.HOME }}/.local/bin/openlatch doctor"
  changed_when: false

```

Run the enrollment against a sample of the wave before the full run, and read the report task's output on those machines. A proxy scheme the install cannot discover shows up there rather than on every machine at once.

Container image and continuous integration

```

FROM debian:bookworm-slim
RUN apt-get update \
  && apt-get install -y --no-install-recommends ca-certificates curl \
  && rm -rf /var/lib/apt/lists/*

ENV OPENLATCH_INSTALL_YES=1 \
  OPENLATCH_NO_INIT=1 \
  OPENLATCH_NO_MODIFY_PATH=1 \
  OPENLATCH_INSTALL_DIR=/usr/local/bin

RUN curl -LsSf https://openlatch.ai/install.sh | sh
COPY entrypoint.sh /usr/local/bin/entrypoint.sh
ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]

```

entrypoint.sh

```
#!/bin/sh
set -eu
: "${OPENLATCH_API_KEY:?the runtime secret was not mounted}"
openlatch init --yes --no-persistence
exec "$@"
```

```
- name: Enroll the runner
  env:
    OPENLATCH_API_KEY: ${ secrets.OPENLATCH_API_KEY }
    OPENLATCH_INSTALL_YES: "1"
    OPENLATCH_NO_INIT: "1"
  run: |
    curl -Lsf https://openlatch.ai/install.sh | sh
    "$HOME/.local/bin/openlatch" init --yes --no-persistence
    "$HOME/.local/bin/openlatch" status
```

The base image carries nothing for the client: the script brings the client with it. The credential is mounted at run time, so no image layer holds it. `OPENLATCH_NO_MODIFY_PATH` stops the install writing a `PATH` line into a shell profile no interactive shell will read, and `OPENLATCH_INSTALL_DIR` puts the client where the image's `PATH` already looks. `--no-persistence` skips the service manager, because a container has no user session to start into. Product telemetry is off in continuous integration without configuration. Ephemeral runners enroll on every job, and because a repeat install is safe, a retried job is safe.

What the client puts on a machine

The client installs and runs at user scope. No administrator rights are required, nothing is written to a system directory, and no service is registered at machine scope.

Item	Detail
One installed unit	The OpenLatch client, installed once per machine that runs an agent
Where it lands	The scripts put the client in <code>~/ .local/bin</code> on macOS and Linux, and <code>%LOCALAPPDATA%\openlatch\bin</code> on Windows, then add that directory to the PATH. <code>OPENLATCH_INSTALL_DIR</code> moves it. This is deliberately not the state directory below
Files	The client keeps its configuration and local records in one directory: <code>~/ .openlatch</code> on macOS and Linux, <code>%APPDATA%\openlatch</code> on Windows. <code>OPENLATCH_DIR</code> moves it. Anything sensitive inside it is readable by its owner only
Agent configuration	Each connected agent's own configuration file gains an entry pointing at the client. Existing content is preserved, and what was there before is recorded so it can be restored
Startup	The client starts with the user's session, under the operating system's own service manager
Credentials	The API key, and any proxy credential, live in the operating system credential store
Network posture	Listens only on the loopback interface. No inbound port is opened

`openlatch uninstall` disconnects every agent it finds on the machine, stops the client, and restores what it changed in each agent's configuration. `openlatch uninstall --purge` also deletes the state directory and the stored credential. A purge is not a removal of the installed files: they sit outside the state directory, so a purge leaves them, along with the PATH line the script added. [Operate](#) covers removal in full.

In short

- Four install routes reach one end state. Pick by where the credential comes from and how many machines the route must reach, not by operating system.
- The install scripts enroll a machine on macOS, Windows and Linux with no runtime required; `npx @openlatch/client init` reaches the same end state where Node.js is present. Either way the install detects the agents present and connects each one, and running it again is safe.

- Some agents ask the developer to confirm the connection once. A machine is not finished until that has happened.
- A machine is installed when openlatch doctor reports no problem and the machine appears in the console's agent record with its agents and an owner.
- One API key per wave, read from your secret store at run time, never in a script, a package or a command line.

Verify and roll out

A machine is deployed when `openlatch doctor` reports no problem and the console holds its record. This page proves both, then widens the deployment in three waves.

Verify a machine

Run both commands on every machine you install, in this order.

```
openlatch status
openlatch doctor
```

`openlatch status` answers one question: is this machine covered right now? A good result shows the machine connected to the console, policies present on the machine, and every agent on the machine wired.

`openlatch doctor` answers the harder question: is that report true? It prints a cause and a remedy against each thing it finds. A good result is that it finds no problem. This is the acceptance step for a machine, so count a machine with an unresolved finding as a failed machine in the wave report. When it reports something, follow the remedy it names and run it again, until it is clean.

What you are confirming	What a good result looks like
The machine is connected	It has reached the console and holds a current record there
The policies are present	Policies are on the machine. This is what lets it keep deciding when the console is unreachable
The agents are wired	Every agent installed on the machine is wired, and none is still waiting for its developer to confirm the connection

Caution

A machine can be running and covering nothing, and from the outside it looks identical to a machine that works: the client is up, the console holds a record, and one agent on the machine was never wired. Do not move to the next rollout step behind an install that `openlatch doctor` has not cleared.

Build a diagnostic archive

When a machine has a problem you cannot resolve from the remedy it prints, build a diagnostic archive.

```
openlatch doctor --rescue
```

The command produces one redacted ZIP archive holding the diagnosis, the client's configuration and a recent window of the machine's local records.

Option	What it does
<code>--since 24h</code>	Bounds the window of local records the archive includes
<code>--yes</code>	Skips the confirmation, for an unattended run

The archive lets a problem be investigated from the machine's own evidence rather than from a description of the symptom. It is redacted for that purpose, so send the archive rather than the directory itself.

Note

Open a support case with the archive attached, the machine's operating system, and the time the problem happened with its time zone. Contact your OpenLatch technical lead through the channel agreed with your organization. Send security reports to security@openlatch.ai.

Confirm in the console

A clean report on the machine is half the check. The other half is the console holding a record for the machine, with the agents it detected and the person its actions are attributed to.

1 **Open the agent record.** Sign in at `app.openlatch.ai` and open **Agents**.

Expected: one row per agent that has reported.

2 **Find the machine.** Search by hostname or by the operating system user the install ran as.

Expected: the machine appears within 24 hours of install, and within minutes on a healthy network.

3 **Confirm the agents on the machine.** Open an agent and read its details, including its control level.

Expected: every agent installed on that machine is listed, each with its control level. A missing agent is an install problem, not a console problem: go back to the machine checks above.

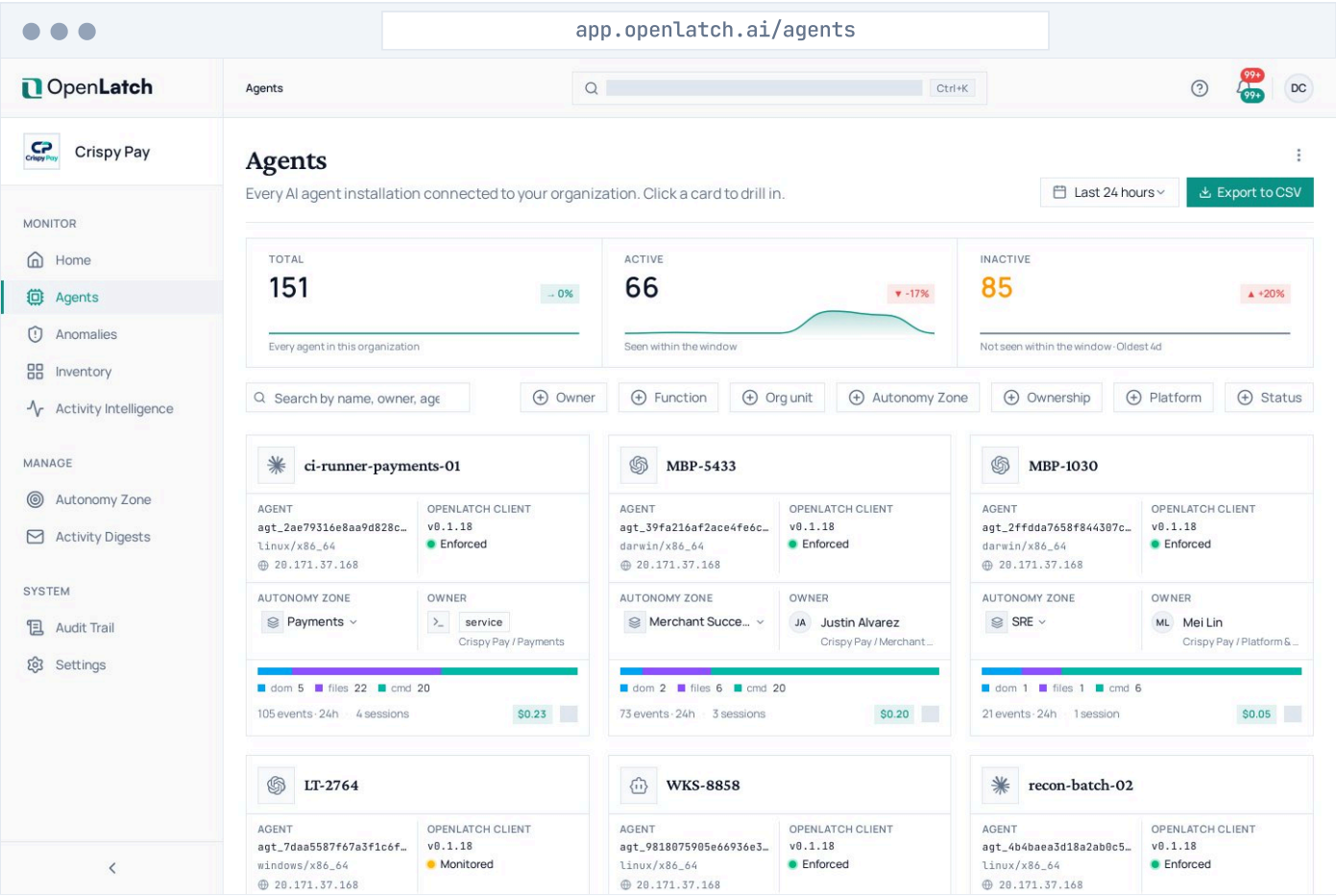
4 **Confirm the resolved owner.** The agent's **Identity** tab names the person its actions are attributed to, with their role and where they sit in the organization. With a directory connected, the owner resolves through it by read-only lookup; without one, OpenLatch works from what the machine reports.

Two controls sit beside the name: one explains how that owner was arrived at, and the other overrides it by hand. Correcting ownership is the editor role's job.

Expected: a named person rather than an empty field.

5 **Confirm records are arriving.** Read the last-seen time on the agent while a developer works on that machine.

Expected: the time moves. A record that stops moving points back at the route out of the machine.



Agents: one row per agent that has reported, with its control level and owner.

The screenshot displays the OpenLATCH web interface. The top navigation bar shows the URL `app.openlatch.ai/agents/ci-runner-payments-01/iden...`. The left sidebar contains a menu with sections: MONITOR (Home, Agents, Anomalies, Inventory, Activity Intelligence), MANAGE (Autonomy Zone, Activity Digests), and SYSTEM (Audit Trail, Settings). The 'Agents' section is highlighted. The main content area is titled 'ci-runner-payments-01' and shows the 'Identity' tab. This tab displays the agent's ownership, listing 'Jenkins CI' as the primary owner and 'Camille Lefèvre' as the accountable person. It also shows a list of agents owned by Camille Lefèvre, including 'ci-runner-payments-01'. The interface includes various status indicators and buttons like 'Override' and 'Why this owner?'.

An agent's Identity tab: the person its actions are attributed to, and where they sit in the organization.

Roll out in waves

Three waves of two weeks each, each one proving something the next depends on. Wave 1 proves the install on the developer path. Wave 2 proves the managed route. Wave 3 proves the perimeter. Issue one client credential per wave and revoke it in the console when the wave completes.

Attribute	Wave 1	Wave 2	Wave 3
Scope	About ten developer machines	About fifty machines	The full perimeter
Install route	Developer path, in the developer's own terminal	Managed route, unattended	Managed route, unattended
Duration	Two weeks	Two weeks	Two weeks
Activity	Enroll each machine, verify it, chase the agents still waiting on their developer, and write down what people hit	Package the install, prove it on one machine of each operating system, then run it unattended and read the report it collects	Run the same package across the perimeter, and chase the machines that never report
Entry criteria	The organization is configured, and a credential is issued for the wave	Wave 1 met its exit criteria, and the managed script runs cleanly on one machine of each operating system	The managed route is proven on every operating system in the estate, and the proxy scheme is confirmed on every subnet in scope
Exit criteria	Every enrolled machine reports to the console within 24 hours of install; openlatch doctor reports no problem on each; no agent is left waiting on its developer; the proxy route is confirmed on every operating system in the wave	The same four, plus every machine installed without a developer opening a terminal	The same four, plus the decision to run it as normal operations
Rollback	openlatch uninstall on the affected machine	The same, delivered through the tool that installed it	The same, delivered through the tool that installed it

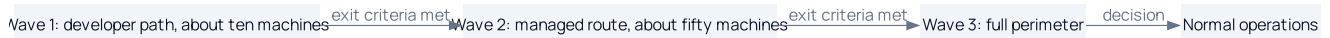


Figure 1. Three waves, what each one proves, and the gate between them.

Problems you are most likely to hit

Symptom	Likely cause	Do this
The machine reports, but one agent on it is not wired	That agent was not installed when the install ran, or its configuration lives at a customized path	Install the agent, then run <code>openlatch init</code> again on the machine. The client is already there, so this needs no download. Where the path is customized, set it in the environment the enrollment runs under
An agent is listed as waiting	Some agents ask the developer to confirm the connection once, the first time they run after installation	Have the developer open the agent once and confirm. Where the estate locks agent configuration centrally, deliver the connection through that managed channel instead
Policies never arrive on the machine	The machine has not reached the console since it was installed, or its credential was revoked or rotated	Confirm the route out first, per Network requirements . Where the credential was rejected, issue one for the wave and run the install again
The machine is connected, but its agents behave as though nothing is there	The agents' own model endpoints are blocked at the proxy, or their streamed responses are buffered	Exempt those endpoints from response buffering and streaming inspection. They are the agents' own traffic and must stay reachable as before
The machine never appears in the console	The route out is blocked, or the enrollment never completed	Run <code>openlatch doctor</code> on the machine. It names what failed, which tells a proxy problem from an install that stopped early

In short

- An install is verified when `openlatch doctor` reports no problem and the console shows the machine's agents, each with a control level and a resolved owner.
- `openlatch status` reports whether the machine is covered. `openlatch doctor` proves it, with a cause and a remedy against each finding.
- `openlatch doctor --rescue` builds one redacted archive with the diagnosis, the configuration and a recent window of local records. Open every support case with it.
- An agent still waiting for its developer to confirm the connection is connected on paper and not in practice. Chase those before closing a wave.
- Three waves of two weeks: about ten developer machines, about fifty on the managed route, then the perimeter. One client credential per wave, revoked when the wave completes.

Operate

Use this page once machines are enrolled: to keep the client current, to work with the records it keeps on each machine, and to remove it.

Updates

The client can check for a new version and update itself. The download is integrity-checked before anything on the machine is replaced: a SHA-512 digest is compared with the digest the package registry publishes for that release, so a corrupted or truncated download is never applied. The check detects corruption and truncation; it does not protect against a compromised registry. Leave automatic updating on for installs a developer runs themselves, where it keeps a machine current without anyone scheduling it.

A managed estate usually wants the version under its own control. Turn automatic updating off and leave the check on, so that a machine still reports that a newer version is available while it waits for your maintenance window. Then deliver the new version the way the client was first delivered, through the tooling that already manages the machine.

Variable	Set it to
OPENLATCH_AUTO_UPDATE	<code>false</code> stops the client from updating itself. Set it wherever you want the version pinned
OPENLATCH_UPDATE_CHECK	Leave it <code>true</code> , so <code>openlatch status</code> still reports that a newer version is available

 Note

Replacing the files on disk does not by itself move the running client to the new version. The machine reports the mismatch until the client restarts, so end the window with `openlatch restart`, or with the sign-out and sign-in your estate already schedules.

Local records

The client keeps its configuration and its local records in one directory: `~/ .openlatch` on macOS and Linux, `%APPDATA%\openlatch` on Windows. `OPENLATCH_DIR` moves it. An estate that moves it should point its own collection and removal scripts at the same value.

Records are written with credentials masked, and file paths are masked the same way. [Data handling and security](#) lists what is captured, what is masked and how long each kind is kept. Records that have not reached the console yet are held in that directory and sent once the console is reachable again. The hold is bounded: past its size cap, 100 MB by default, the oldest unsent records are dropped and the daemon log records an 0L-1204 warning. See the outbox and fallback settings in [Configuration](#).

Open a support case

Run `openlatch doctor` on the affected machine first. It reports what it finds in plain terms. Then build the redacted archive with `openlatch doctor --rescue`: it holds the diagnosis, the configuration and a recent window of the local records. Send that archive, never the directory itself, with the operating system and the time of the problem with its time zone.

Contact your OpenLatch technical lead through the channel agreed with your organization. Send security reports to security@openlatch.ai.

Remove the client

To remove the client from a machine:



```
openlatch uninstall
```

To remove the client and the records it kept locally:



```
openlatch uninstall --purge
```

`openlatch uninstall` unwires the client from every agent it finds on the machine and stops it. Each agent is left as it was before the install, and settings OpenLatch did not write are not touched. The

configuration and the local records stay on disk unless you ask for them to go.

`--purge` also deletes that directory and the credential held in the operating system's credential store. It refuses any directory that is not the client's own, and it resolves symbolic links before it deletes, so a link cannot redirect the delete elsewhere. An estate that moves the directory should keep the same check in its own removal script.

Neither command deletes the client's own files. They were installed outside the state directory, so a purge leaves them on the machine, along with the PATH line the install added. To take the client off a machine completely, delete the two files the install placed, `openlatch` and `openlatch-hook` in `~/ .local/bin` on macOS and Linux, or `openlatch.exe` and `openlatch-hook.exe` in `%LOCALAPPDATA%\openlatch\bin` on Windows. Leave the directory and anything else in it: other software may use it. On Windows, the directory belongs to the client, so you can also remove it and the PATH entry the install added; on macOS and Linux, keep the PATH line if other tools rely on `~/ .local/bin`. There is no uninstall script, so an estate that wants this done for it does it through the tooling that installed the client.

Removal is per machine. Nothing has to be undone in the console for a machine that has been uninstalled, and a machine that comes back is installed the same way as the first time.

In short

- Leave automatic updating on for developer installs. Pin it with `OPENLATCH_AUTO_UPDATE=false` on a managed estate, and keep the check on so a machine still reports what it is missing.
- A new version takes effect when the client restarts: end the maintenance window with `openlatch restart` or a sign-out and sign-in.
- Configuration and local records sit in one directory. A support case opens with the redacted archive from `openlatch doctor --rescue`.
- `openlatch uninstall` leaves each agent as it was. `--purge` also removes the directory and the stored credential, and leaves the two installed files, which a full removal deletes separately.

Data handling and security

Hand this page to a security review. It sets out what leaves a machine running the client, what stays on it, how long each record is kept, and how access to the console is controlled.

OpenLatch controls a misbehaving agent: one that loops or spends budget on work nobody asked for, one that has been injected through its context and acts on it through its tools, and one that breaks a policy while otherwise behaving as designed. Those failures produce real incidents, and they are what the control loop catches before the action executes. OpenLatch does not claim resistance to an agent built to evade it.

Principles

Principle	What it means
The decision is made on the machine	Every verdict is computed on the machine, before the action runs. The console is not on the decision path
OpenLatch sits in no network path	The client is not a gateway and inspects none of your organization's traffic. Your agents reach their model endpoints exactly as before
Least data	Values are masked and strings truncated before anything leaves the machine, and metadata is sent wherever content is not needed
No credential leaves the machine	The keys your agents use stay on the machine. Proxy passwords are held by the operating system, never in a configuration file and never in a log

Data inventory

Read the second column first.

Data element	Leaves the machine	Masking and limits	Where it is held	Retention
Action metadata (tool name, session, verdict, timing)	Yes	None applied. Metadata carries no free text	Console	180 days
Shell commands and tool arguments	Yes	Credentials masked with labelled placeholders. Each string capped at 4 KB	Console, and in hashed form in usage rollups	180 days; rollups 30 days
File paths	Yes	Masked the same way, and hashed with a per-organization key in usage rollups	Console, and in hashed form in usage rollups	180 days; rollups 30 days
File contents	Configurable	Filtered by default, or hash-only. Inline capture capped at 64 KB	Console	180 days
Prompts	Metadata only	Occurrence, timing, session and length. Prompt text is not sent unless your organization enables it	Console	180 days
Model requests and responses	No	Content is never sent. Token counts, the model used and the cache split are recorded	Console	90 days
User identity	Yes	Operating system user and git email, to attribute an action to a person	Console	180 days, with the record

Data element	Leaves the machine	Masking and limits	Where it is held	Retention
Machine identity	Yes	Operating system, architecture, local and public IP address, software versions, an install ID	Console	180 days, with the record
Model account identifier	Yes	An account ID, or a marker showing that a shared key is in use. Never a token and never a key	Console	180 days, with the record
Policies	Downstream only	Sent to the machine, never uploaded from it	On the machine	Until a newer set replaces them
Verdicts and anomalies	Yes	None applied	Console	180 days

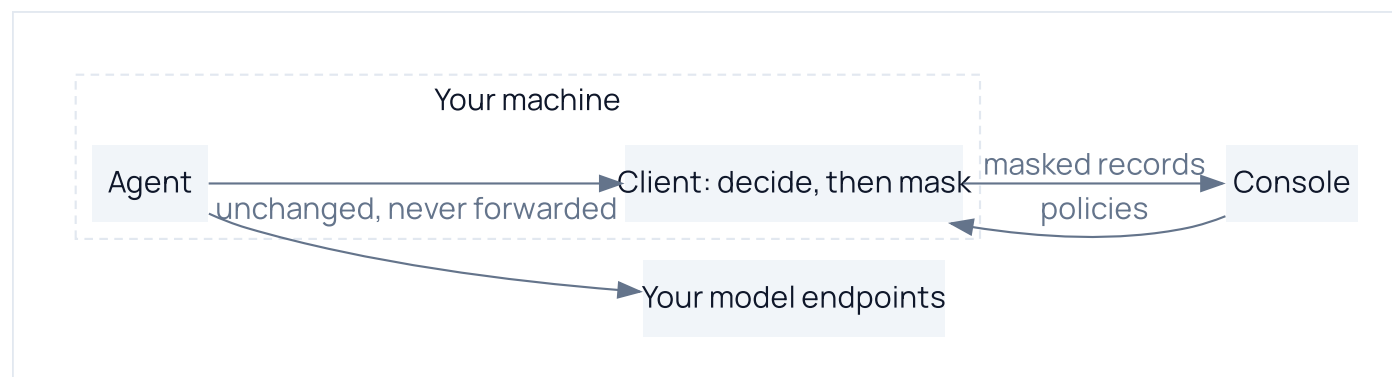


Figure 1. Data flow and trust boundaries: masked records go up, policies come down, and model traffic never passes through OpenLatch.

Masking, and what stays on the machine

Masking runs in the client, on the machine, before a record is queued for delivery. It walks the payload and replaces credential-shaped values with a labelled placeholder that keeps the recognizable public prefix and drops the secret, in the form `[AWS_KEY:AKIA**]`. Only string values are masked; field names never are. Masking runs before truncation, so cutting a string at the 4 KB cap cannot expose the tail of a partly

masked secret. It runs after the verdict, so redacted text never changes which policies match. Your organization can add its own credential shapes to the client's configuration, and they apply alongside the built-in ones.

 **Caution**

The client keeps local records of actions on the machine in plain text, in `~/ .openlatch` on macOS and Linux or `%APPDATA%\openlatch` on Windows. They hold masked command text and file paths, and are protected by the user's own file permissions and by whatever disk encryption your estate applies. Credentials are the exception: they are held by the operating system's credential store where one is available, and otherwise in a file readable only by its owner.

Retention

Data	Window	Configurable	Export
Action records	180 days	No	—
Decision records and anomalies	180 days	No	CSV
Usage rollups	30 days	No	Console
Hashed archives	90 days	No	Console
Cost and token records	90 days	No	Console
Audit trail	90, 180, 365 or 730 days, or a custom window	Yes, with auto-purge	CSV or JSON

The audit trail is the one window you set yourself, in the console, with auto-purge; [Configure the organization](#) shows where. Every other window above is fixed. The policies held on the machine have no retention window: they are replaced when the client fetches a newer set, and removed when the client is uninstalled with `--purge`.

Hosting

The console is a service hosted in the United States, in the US-East region. Each organization's data is kept apart from every other organization's by the database itself: row-level security is enforced on every tenant table, with no runtime role that bypasses it. A list of subprocessors is available on request.

In transit, the client and the console speak TLS validated against the operating system trust store, with the option to trust an additional certificate authority where your estate terminates TLS with its own, and no option anywhere that disables verification. At rest, stored secrets are encrypted.

Identity and access

The console supports SAML 2.0 single sign-on per organization. You supply the identity provider's entity ID, its sign-on URL, and a signing certificate or a metadata URL, then set the allowed email domains. A test runs the exchange and reports the result before the connection goes into service. Microsoft Entra ID, Okta, PingOne and Active Directory Federation Services connect this way.

Two gates control who joins. Allowed email domains restrict which addresses the console accepts at all, and invitations control who is admitted. A per-organization SAML connection is the only identity source that admits a member without an invitation, so a social sign-in is never a way around provisioning.

Role	What it can do
owner	Everything an admin can do, plus ownership of the organization itself
admin	Identity settings, policies, client credentials and retention settings
editor	Policies and agent ownership overrides
viewer	Read access to agents, executions, decisions and the audit trail. Invitees start here

A client credential inherits the role of the person who issued it. One issued by an editor cannot change an identity setting, and revoking the credential or demoting the person both close the same door.

The audit trail is append-only and hash-chained, so a removed or edited row breaks the chain. Every policy change writes a row naming the actor, with the state before and after. The trail exports as CSV or JSON, its retention window is configurable with auto-purge, and the export itself writes an audit event.

Protecting the client and its policies

Client updates are integrity-checked before anything is replaced: the SHA-512 digest of the download must match the digest the package registry publishes for that release. That check detects corruption and truncation; it does not protect against a compromised registry.

The policies held on the machine are checked before use, and again on every load, not only when they are fetched. A set that fails the check is discarded and reported, and the machine runs without policy until a newer set arrives. That state is stamped on every record rather than left silent, and the same holds where no policy has ever been fetched: the action is allowed and the record says why. OpenLatch never blocks an action because policy is missing.

Secret material is held by the operating system's credential store where one is available, and otherwise in a file readable only by its owner. A credential passed on a command line is refused, because a command line is readable by other processes on the machine.

Product telemetry and crash reports

Product telemetry is about OpenLatch software, not about your organization's agents. It carries command names, durations, the agent, error codes and volumes, and never command text, file paths, file contents or prompts. Consent is asked during installation, can be changed later, and is off by default in continuous integration. Crash reports carry a scrubbed failure message and nothing else, and are sent unless you turn them off.

Both travel to `app.openlatch.ai` with the traffic the client needs to run, so a network rule cannot stop them without cutting the client off from the console. The switches are on the client instead: decline telemetry at installation or run `openlatch system telemetry disable`, and turn crash reports off with `OPENLATCH_CRASH_REPORTING=0`. With them off, the client behaves exactly as it does with them on.

[Privacy and telemetry](#) explains both in more detail.

Report a vulnerability

Send security reports to security@openlatch.ai. OpenLatch acknowledges a report within two business days and works the finding with the reporter from there.

In short

- Action metadata, masked commands and file paths, verdicts and anomalies leave the machine. Prompt text stays on the machine unless your organization enables it, and model requests and responses are never sent.
- One directory holds the configuration and the local records, in plain text, protected by the user's file permissions and your disk encryption. Credentials are held by the operating system, or in an owner-only file.
- Action and decision records are kept 180 days, usage rollups 30 days, hashed archives and cost records 90 days. The audit trail is the one window you set, with auto-purge, and it exports as CSV or JSON.
- The console is hosted in the United States (US-East), with each organization's data isolated by the database. SAML2.0 per organization, four roles from owner to viewer, client credentials that inherit their issuer's role, and an append-only, hash-chained audit trail.
- Updates are integrity-checked before they are installed, and the policies on the machine are checked before use and on every load. A machine with no usable policy allows the action and says so on the record.

Deployment FAQ

Use this page to answer the questions a deployment raises. Where another page treats a question in full, the answer points to it.

Installation

Question	Answer
What do you need from us to start?	Three things: an organization in the console with one administrator who can invite everyone else, outbound HTTPS on 443 to <code>app.openlatch.ai</code> , and the perimeter, meaning which machines are in the first wave. The install scripts bring the client with them, so a machine needs no runtime of its own. Prerequisites turns that into a checklist
Does it need administrator rights on the machine?	No. The client installs and runs at user scope, loads no driver and opens no inbound port
Can we deploy it with our existing management tooling?	Yes. Because the client runs in user context, it is delivered as a user-scope script rather than a device-scope installer, with the credential passed from your secret store through the environment and never written into a script, a package or a command line. Create one key per rollout wave and revoke it when the wave completes. See Deploy the client
Which agents does it cover?	The install detects the agents present on a machine and wires each one it finds. Control levels lists the agents wired today, plane by plane. Every agent appears in the console with its control level, Enforced, Monitored or Known, which tells you how much control you have over it
Does the developer have to do anything?	Usually not. Some agents ask the developer to confirm the connection once, the first time they run after installation, and <code>openlatch status</code> shows which agents are still waiting. See Verify and roll out

Network and proxy

Question	Answer
What does OpenLatch need to reach, and does it open anything?	One destination, outbound: <code>app.openlatch.ai</code> on TCP 443 over HTTPS, allowlisted by name because there is no static address range. Nothing inbound: the client listens only on the loopback interface. See Network requirements
Our only path out is an authenticated proxy. What is supported?	HTTP and HTTPS CONNECT, SOCKS5, transparent proxies, and PAC or WPAD where the operating system's own resolver evaluates it. Basic and Kerberos (Negotiate) authentication are supported. NTLM is not, by design; Negotiate is the route on those estates. Proxy credentials live in the operating system's credential store
Does TLS inspection break it?	No, provided the interception certificate authority is trusted. Install it in the operating system trust store, or point the client at it, in which case it is merged on top of the system store. Verification is never disabled, and OpenLatch issues no certificate of its own
Does our agents' own traffic change?	No. Your agents reach their model endpoints exactly as before. Those endpoints must stay reachable and be exempt from response buffering and streaming inspection
What happens when a machine is offline, or the console is unreachable?	Policies stay in force on the machine, because the decision is made locally. Records are held on the machine and sent when the console is reachable again. See Architecture

Data and security

Question	Answer
Is this a proxy or a gateway in front of our agents?	Neither. OpenLatch runs inside the agent's own lifecycle, and the decision is made on the machine before the action runs. Nothing of ours sits in your network path, and OpenLatch terminates no TLS
What leaves the machine? Our prompts? Our source code?	Prompts are captured as metadata by default: that one occurred, when, in which session and how long it was. Prompt text is not sent unless your organization turns that on, and the requests and responses exchanged with a model are never sent to OpenLatch. What does leave is action metadata with credentials and file paths masked, plus the identity signals needed to attribute an action; file contents are filtered by default. OpenLatch connects to none of your repositories. See Data handling and security
Where is the console hosted?	In the United States, in the US-East region, with each organization's data isolated from every other organization's. Verdicts are produced on your own machines, so where records are stored does not affect enforcement. A list of subprocessors is available on request
Who can sign in to the console, and who can see what?	Single sign-on through SAML 2.0 is configured per organization, and access inside the console follows four roles. Every administrative change is written to an append-only audit trail that you can export. See Configure the organization
Can someone turn it off on their own machine?	OpenLatch addresses a misbehaving agent: one that loops, wastes budget, is injected and acts through its tools, or breaks a policy while otherwise working normally. The client installs in the user's own scope, so a person with control of their machine can remove it. What that costs them is coverage, not secrecy: coverage is reported per machine, and a machine that stops reporting shows up as a gap rather than passing as covered

Operations

Question	Answer
Does it slow the agent down?	The decision is made on the machine, with no network call and no model on the path, so a covered action does not wait on a round trip. OpenLatch publishes no latency figure. Its claim is that every covered action reaches the policy decision point, and the verdict an explicit policy produced is the verdict that is enforced
How do we confirm a machine is really connected?	<code>openlatch status</code> reports whether the machine is connected, whether policies are present and whether the agents are wired, and <code>openlatch doctor</code> proves each of those. The machine's agents then appear in the console's agent record with an owner. See Verify and roll out
How do updates work, and can we pin them?	The client can update itself, and each download is integrity-checked before anything is replaced. Leave that on for installs a developer runs. A managed estate turns it off, keeps the check on, and delivers the new version in its own maintenance window. See Operate
How do we get notifications and records into our own tooling?	Notifications go to email, a chat channel such as Slack or Microsoft Teams, or a webhook you own, and the audit trail exports as CSV or JSON on the retention you set. See Set up notification channels
How do we remove it?	<code>openlatch uninstall</code> unwires the client from every agent on the machine and leaves each one as it was. Adding <code>--purge</code> also removes the local directory and the stored credential. See Operate
How do we open a support case?	Contact your OpenLatch technical lead through the channel agreed with your organization. Run <code>openlatch doctor --rescue</code> on the affected machine and attach the redacted archive it produces, with the operating system and the time of the problem with its time zone. Security reports go to security@openlatch.ai

Terms and egress

Use this page to look up a term from the Deploy section, or to copy the egress rule into a change request.

Terms and acronyms

Term	Meaning
Agent Autonomy Zone	The operating envelope an agent works inside, expressed as intent policies across five dimensions: Quality, Economics, Security, Safety and Compliance. See Autonomy Zone
Agent Intent Policy (AIP)	The form in which your organization's intent is authored in the console, and compiled to the deterministic rules that decide an action
Verdicts	Allow, Ask, Block or Optimize, in this order. Allow lets the action proceed. Ask lets it proceed and flags it for asynchronous review. Block stops it before it executes. Optimize changes the path: in Enforce a lever acts, and in Monitor it measures what it would have done. See Verdicts
Client	The OpenLatch software installed once on each machine that runs an agent. It decides a covered action, on the machine, before the action runs
Console	The OpenLatch web application at <code>app.openlatch.ai</code> , where you set up the organization, author policies and see what your agents did
Control level	How much control OpenLatch has over one agent. Enforced: the verdict reaches the agent and is obeyed. Monitored: events flow, but no verdict can be delivered. Known: the agent is identified and classified, with the limits of control stated. See Control levels
IdP	Identity provider: your organization's directory and authentication service
MDM	Mobile device management: the tooling that delivers software and settings to managed machines
PAC / WPAD	Proxy auto-config and Web Proxy Auto-Discovery: a script that tells a client which proxy to use, and the mechanism that locates it
SAML2.0	Security Assertion Markup Language, version 2.0: the single sign-on protocol the console supports per organization
SNI	Server Name Indication: the TLS extension that carries the destination hostname in clear, which allowlisting by name reads

Term	Meaning
TLS inspection	A proxy that terminates and re-issues TLS to inspect traffic. Supported when the interception certificate authority is trusted
Wave	One stage of a rollout. The Deploy section runs three waves of growing size

Egress requirements

Hand this table to whoever manages your egress. A machine running the client needs exactly one destination from OpenLatch, and the optional product telemetry and crash reports travel to it too. The second row is needed only while you install or update the client. The third is for the browsers that sign in to the console; no machine running the client contacts it.

Destination	Port	Direction	Protocol	How to allow it
<code>app.openlatch.ai</code>	TCP 443	Outbound	HTTPS	By name. There is no static address range. This entry also carries the optional product telemetry and crash reports
Your package registry: <code>registry.npmjs.org</code> , unless you point the client at a mirror	TCP 443	Outbound	HTTPS	By name, or use your internal mirror. Needed for an install or an update, not to run
<code>challenges.cloudflare.com</code> , from the browser of anyone who signs in to the console	TCP 443	Outbound	HTTPS	By name, for browsers only. It serves the check the email step runs at sign-in

Your agents reach their model endpoints exactly as before. Those endpoints must stay reachable, and exempt from response buffering and streaming inspection.

No inbound port is required, on the machine or anywhere else. OpenLatch issues no certificates and requires none. [Network requirements](#) is the full treatment, including proxies, authentication and TLS inspection.

AIP reference

Use this page to read or review an Agent Intent Policy (AIP): what it is made of, which dimension of the Autonomy Zone it covers, which mode it runs in, and which verdict it returns for a covered action.

An intent policy states what your organization means, rather than every surface an agent might use. OpenLatch writes each intent policy as an AIP and compiles it to deterministic rules that the client applies before the action runs.

Intent atoms

An AIP is a set of one or more **intent atoms**. One clause of your source document becomes one atom, and every atom has the same fixed fields.

Field	Contents
who	The agents the atom applies to. When the source does not say, the atom states that it was inferred
does what	The action, such as a deployment or a file deletion
to what	The target of the action, such as a production namespace
only if	The conditions under which the action is acceptable
unless	The exceptions to the atom
class	Whether the atom can be enforced: enforceable, monitor-only or out-of-scope

From these fields OpenLatch generates a **sentence** and a set of **test cases**. You never edit the compiled rules.

You see	What it is for
The sentence	What you ratify, with one Yes or No
The source	The clause of your document the atom comes from, quoted so you can check it was read correctly
The test cases	Concrete situations with the verdict each one would get, shown read-only beside the sentence so your answer is informed

A No sends the atom back for correction: you say what you meant, the fields change, and the atom returns with a new sentence and new cases for a fresh answer. Changing an atom always cancels the ratification given to its earlier version.

An atom's `class` is a property of the atom, not a setting. A `monitor-only` atom describes something that can be observed but not stopped before it happens, so it stays in Monitor even when its policy is in Enforce.

Dimensions

Every atom belongs to one of the five dimensions of the Autonomy Zone. A policy covers the union of its atoms' dimensions.

Dimension	The question it answers	Example intent policies
Quality	Is the work up to standard?	Code is tested before it ships; the mandated process is followed
Economics	Is the cost justified?	No runaway loops or pointless retries; the model matches the task
Security	Can it touch this?	No credential leaves the corporate boundary; only approved tools and servers
Safety	Is the consequence acceptable?	No production access without approval; no irreversible or destructive operations
Compliance	Is it lawful and by the book?	Internal policies and standards; contractual obligations

The mission is not a dimension: each agent brings its own mission, and the zone brings the rules it must follow.

Modes

A policy runs in one of three modes. The mode belongs to the policy, never to an atom, and is set in each Autonomy Zone that holds the policy.

Mode	What happens on a match
Disabled	Nothing. The policy and its history are kept, but it is not evaluated
Monitor	The match is recorded as an anomaly and flagged; the action proceeds
Enforce	The verdict is applied before the action runs, wherever an adapter covers that action, and the record states whether it was actually enforced

When an agent's zones hold the same policy more than once, it runs once, in the strictest mode: Enforce over Monitor over Disabled.

Policies earn Enforce on evidence: run a policy in Monitor, review its anomalies, and move it to Enforce once it is accurate. OpenLatch never widens an agent's authority on its own.

Verdicts

Every covered action that matches a policy in Enforce receives one of four verdicts before it runs. A verdict is deterministic because an explicit policy produced it.

Verdict	What happens to the action
Allow	The action proceeds unchanged. In Monitor, it may be allowed and flagged
Ask	The action is allowed and flagged for review by default
Block	The action is prevented before it runs. The agent gets the reason and can try something else; the session continues
Optimize	The action takes a different path, through one of the levers below

Ask

Ask allows the action and flags it for asynchronous review by default. The review decision, with the scope the reviewer picks, travels to the client in the next policy bundle, so the next similar action is decided on the machine without a person.

Ask holds an action and waits for approval only on a per-atom opt-in for an irreversible action, such as dropping a production database.

Codex CLI has no native ask prompt. On Codex CLI, Ask surfaces to the agent as a message with the rule's reason, and the call proceeds.

Ask prevents recurrence, not the first occurrence. Write anything that must never happen unreviewed as Block, or mark the action irreversible on its atom.

Optimize levers

Optimize changes the path of an action instead of allowing or blocking it. The mode decides what a lever does: in Enforce a lever acts, and in Monitor a lever only measures what it would have done.

Lever	What it changes	Status
loop_stop	Stops an agent that repeats the same call, and tells it why	Acts in Enforce
steer	Refuses the action and tells the agent what to do instead, such as pushing a branch rather than the main branch	Acts in Enforce
narrow_output	Returns a filtered subset of a long command output to the agent, and keeps the full output on disk	Acts in Enforce, on Claude Code
effort_clamp	Lowers the reasoning effort of a model request	Measures in Monitor and in Enforce; acts in Enforce only where the customer opts in
context_edit	Tightens the context-clearing settings of a model request	Measures in Monitor and in Enforce; acts in Enforce only where the customer opts in
prefix_guard	Keeps the cached prompt prefix of a model request stable	Measures in Monitor and in Enforce; acts in Enforce only where the customer opts in
substitute	Swaps a command for an equivalent your organization approved	Not shipped: it has no actuator and only measures in Monitor

In Enforce, `loop_stop`, `steer` and `narrow_output` act (`narrow_output` on Claude Code). `effort_clamp`, `context_edit` and `prefix_guard` only measure, even in Enforce, unless the customer opts in. `substitute` is not shipped, so it only measures in Monitor. In Monitor, every lever only measures.

The evidence record says which applies: applied in Enforce, measured in Monitor.

What a verdict covers

Every covered action reaches the policy decision point, and its verdict is enforced. An action outside a covered surface, such as a script the agent wrote earlier and runs later, is not decided by the client. Each

agent's control level states which coverage applies to it.

Client configuration

Use this page to change how the client behaves on a machine: where it keeps its state, how it reaches the platform, and which of its parts run.

`openlatch init` writes the configuration for you. Edit it by hand only for a setting no command covers, then run `openlatch restart` to apply it: the daemon reads its configuration once, when it starts.

Where settings come from

The client reads each setting from the first source that sets it, in this order:

1. A flag on the command line.
2. An environment variable (`OPENLATCH_*`).
3. The configuration file, `config.toml` in the state directory: `~/ .openlatch` on macOS and Linux, `%APPDATA%\openlatch` on Windows.
4. The built-in default.

The proxy route is the one exception: it resolves key by key, and below the configuration file it also honours the standard `https_proxy` family of variables, then the operating system's own proxy settings, before it goes direct.

The supervised daemon does not inherit your shell's variables. Anything it needs, such as the platform address, `openlatch init` writes into `config.toml` so the daemon uses it too.

Environment variables

State and connection

Variable	What it sets	Default
OPENLATCH_DIR	The state directory: configuration, credential, logs and outbox	~/ .openlatch on macOS and Linux, %APPDATA%\openlatch on Windows
OPENLATCH_PORT	The daemon's port. Set, openlatch init uses exactly this port; unset, it looks for a free one from 7443 to 7543	7443
OPENLATCH_API_URL	The platform address the client sends events to and fetches policy from	https://app.openlatch.ai
OPENLATCH_API_KEY	The enrolment credential. openlatch init stores it in the operating system's keychain, or in an encrypted file when no keychain is usable	—
OPENLATCH_SKIP_KEYRING	Treats the keychain as unavailable, so the environment variable wins	unset
OPENLATCH_LOG	Log level: error, warn, info, debug or trace	info

Proxy

Variable	What it sets	Default
OPENLATCH_PROXY	The proxy URL for every outbound request. It is the one setting that may carry a credential; <code>openlatch init</code> moves the password to the keychain instead of writing it to disk	unset
OPENLATCH_NO_PROXY	Extra destinations that skip the proxy. Loopback addresses always skip it	unset
OPENLATCH_PROXY_MODE	auto, manual or direct	auto
OPENLATCH_PROXY_AUTH	auto, none, basic or negotiate (Kerberos). NTLM is not supported	auto
OPENLATCH_PROXY_PAC_URL	An explicit proxy auto-config URL, used on macOS and Windows. Refused on Linux with 0L-1225	unset
OPENLATCH_PROXY_SPN	The Kerberos service name of the proxy	HTTP/<proxy-host>
OPENLATCH_CA_BUNDLE	A PEM bundle added to the operating system's trusted certificates, for a proxy that inspects TLS traffic	unset

Parts of the client

Variable	What it sets	Default
OPENLATCH_POLICY_ENABLED	Local policy evaluation. false turns enforcement off completely on this machine; it is not a Monitor mode (0L-1215)	true
OPENLATCH_MODEL_RELAY_ENABLED	The model relay, the local listener that records each agent's model calls. The daemon points an agent at it only once a test request proves it reaches the provider	true
OPENLATCH_MODEL_RELAY_PORT	The model relay's port	7600
OPENLATCH_INVENTORY_ENABLED	Watching agent configuration files (the configuration plane)	true
OPENLATCH_CLINE_PLUGIN_DELIVERY	Delivering the OpenLatch plugin to Cline.app and the VS Code Cline extension. The same as plugin_delivery in [cline]	true

Delivery to the platform

Variable	What it sets	Default
OPENLATCH_CLOUD_BATCH_MAX_EVENTS	Events sent per batch, from 1 to 100	50
OPENLATCH_CLOUD_BATCH_MAX_WAIT_MS	How long the first event of a batch waits before the batch is sent, in milliseconds	5000
OPENLATCH_CLOUD_OUTBOX_MAX_BYTES	The most unsent events the outbox holds, in bytes, before the oldest are dropped. 0 removes the cap	104857600
OPENLATCH_CLOUD_FALLBACK_MAX_BYTES	The most unreplayed events the fallback log holds, in bytes, before the oldest are skipped. It caps that window, not the file: during a long outage the file on disk keeps growing until the daemon replays it. Read by the daemon only; the hook keeps its own 50 MB limit. 0 removes the cap	52428800

Sending events to the platform cannot be turned off. To send them elsewhere, change `OPENLATCH_API_URL`.

Updates and telemetry

Variable	What it sets	Default
OPENLATCH_AUTO_UPDATE	Automatic updates: the daemon checks for a release regularly and installs it once no hook is running. true or 1 turns it on; any other value turns it off. The same as auto_update in [update]	true
OPENLATCH_UPDATE_CHECK	The version check at start-up. false or 0 turns it off. The same as check in [update]	true
OPENLATCH_NPM_REGISTRY	The registry openlatch update downloads from	https://registry.npmjs.org
OPENLATCH_TELEMETRY_DISABLED	1 turns usage telemetry off	unset
DO_NOT_TRACK	1 turns usage telemetry off, as in other tools that follow this convention	unset
OPENLATCH_CRASH_REPORTING	0, false, no or off stops crash reports, independently of usage telemetry	on

Automatic updates never run on CI runners.

Configuration file

config.toml in the state directory holds the settings the commands write, grouped in sections. The sections you are most likely to read or edit:

Section	Key	What it sets
[policy]	enabled	Local policy evaluation, as OPENLATCH_POLICY_ENABLED
[model_relay]	enabled	The model relay, as OPENLATCH_MODEL_RELAY_ENABLED. openlatch system model-relay enable and disable write it
[cline]	plugin_delivery	Delivering the OpenLatch plugin to Cline hosts
[update]	auto_update · check	Automatic updates and the start-up version check
[proxy]	see below	The route every outbound request takes

The [proxy] section

openlatch init finds the route on its own where it can and writes it here. openlatch system proxy set, discover and clear change it; prefer them to editing the file.

```
[proxy]
mode = "auto"
url = "http://proxy.corp:8080"
auth = "auto"
no_proxy = ".corp.example,10.0.0.0/8"
ca_bundle = "/etc/ssl/corp-root.pem"
allow_direct = true
```

Key	What it sets
mode	auto looks for a working route, manual uses exactly what is set and is never replaced, direct never uses a proxy
url	The proxy, as http://, https://, socks5:// or socks5h://. It never holds a username or password
username	The account to authenticate as. The password lives in the keychain, never in this file
auth	auto picks a scheme when the daemon starts; basic, negotiate and none are explicit
no_proxy	Extra destinations that skip the proxy, such as example.com, .example.com or 10.0.0.0/8
ca_bundle	A PEM bundle added to the trusted certificates, never a replacement for them
allow_direct	false refuses a direct connection when no proxy can be used (0L-1227)
spn	The Kerberos service name of the proxy
http1_only	Forces HTTP/1.1, for an inspecting proxy that mishandles HTTP/2
pac_url	An explicit proxy auto-config URL. Refused on Linux (0L-1225)

Run `openlatch system proxy test` after any change to check the route.

What the client writes into agent settings

While the model relay is on, the daemon points each detected agent's model calls at it, and removes that wiring when you stop the daemon, switch the relay off or uninstall. A value you set yourself is never overwritten: `openlatch doctor` reports it instead (0L-RELAY-SETTINGS).

Agent	Where the wiring goes
Claude Code	Environment keys in its settings: the relay address, an install identifier, and loopback entries in <code>NO_PROXY</code>
Codex CLI	An <code>openlatch</code> provider entry in <code>~/.codex/config.toml</code>
Cursor	The relay address in the IDE's user settings, and a shell wrapper for the Cursor CLI that a new terminal picks up
Cline	One local relay endpoint per configured provider, in Cline's own settings files

Wiring Claude Code to the model relay sets its `ANTHROPIC_BASE_URL`, which turns off Claude Code Remote Control while the relay is on. On a machine behind a proxy, also export `NO_PROXY=127.0.0.1,localhost` in the shell that starts Codex CLI, so its model calls reach the local relay rather than the corporate proxy.

Run `openlatch system model-relay status` to see which agents' model calls go through the relay.

Glossary

Use this page when a term in the console, the client output or these docs needs a precise meaning.
Product names, verdicts and modes keep their English form in every language.

Autonomy Zone and policies

Term	Meaning
Agent Autonomy Zone	The operating envelope an organization defines for its AI agents: what an agent may do, under which conditions, and with which consequences accepted. Short form: Autonomy Zone
Dimension	One of the five sides of the Autonomy Zone: Quality, Economics, Security, Safety and Compliance
Effective zone	The one zone an agent resolves to. A unit of your organization without its own zone inherits the nearest one above it
Intent policy	A rule that states what your organization means, rather than every surface an agent might use
Agent Intent Policy (AIP)	The form an intent policy takes in OpenLatch: a set of intent atoms, compiled to deterministic rules. See the AIP reference
Intent atom	One clause of a policy, held in fixed fields (who, does what, to what, only if, unless) from which a sentence and test cases are generated
Ratification	The policy owner's Yes or No on an atom's sentence, with its test cases shown beside it
Mode	How a policy runs in a zone: Disabled, Monitor or Enforce
Disabled	A mode in which the policy is kept but not evaluated
Monitor	A mode in which a match is recorded and flagged, and the action proceeds
Enforce	A mode in which the verdict is applied before the action runs, wherever an adapter covers it

Decisions

Term	Meaning
Verdict	The decision a policy returns for a covered action, before it runs: Allow, Ask, Block or Optimize
Allow	The action proceeds unchanged
Ask	The action is allowed and flagged for review by default. It is held for approval only on a per-atom opt-in for an irreversible action
Block	The action is prevented before it runs, and the agent is told why
Optimize	The action takes a different path, through an Optimize lever
Optimize lever	One way Optimize changes an action. In Enforce a lever acts where it is shipped; in Monitor it only measures what it would have done
Anomaly	An agent action that violates an intent policy of one of the zone's dimensions
Deterministic	A verdict produced by an explicit policy, so the same action in the same context always gets the same verdict

Agents and coverage

Term	Meaning
Sanctioned agent	An agent your organization has approved and connected to OpenLatch
Control level	How much OpenLatch controls an agent: Enforced, Monitored or Known
Enforced	A control level: an OpenLatch adapter sits in the agent's execution path and applies verdicts
Monitored	A control level: the agent's actions are visible and its anomalies recorded, but verdicts are not applied
Known	A control level: the agent is identified and classified, with the limits of control stated
Command plane	The actions an agent takes through its tools: commands, file changes, tool and server calls
Request plane	The requests an agent sends to its model provider
Execution graph	The record of what an agent did in a session: model calls, tool calls, commands, file changes, data access and cost, in order
Evidence record	What OpenLatch keeps for each decision: the action, the policy, the mode, the verdict and whether it was enforced

The client

Term	Meaning
Client	The OpenLatch program installed on each machine where agents run
Daemon	The client's background process, which receives each agent event and decides it
Hook	The entry point an agent calls before and after each action, which hands the action to the daemon
Policy bundle	The integrity-checked file of rules the client fetches from the platform and keeps on disk, so it keeps deciding with the network unplugged
Model relay	A local listener the client points each agent's model calls at, so the request plane is recorded
Outbox	The local queue where the client keeps events until the platform has received them
Supervisor	The operating-system service (launchd, systemd or Task Scheduler) that starts the daemon at login and restarts it
Configuration plane	The agent configuration files the client watches for changes
Error code	A code such as OL-1002 that the client prints with a problem. Each one has its own page, listed in the CLI reference

The platform

Term	Meaning
Platform	The OpenLatch service where you define zones and policies and review what agents did, reached in the console at <code>app.openlatch.ai</code>
Console	The platform's web interface
Organization	Your company's account on the platform, which holds its units, zones, policies and agents
API key	A credential that lets the client or a script act for your organization on the platform

CLI reference

Every command and subcommand of the OpenLatch client.

openlatch init

`openlatch setup`

Installs the hooks of the agents found on this machine, starts the daemon and checks that it answers. Safe to run again: it frees the client's ports first, then confirms that the daemon it started is the one serving.

openlatch status

Shows the daemon's state, its uptime and event counts, the health of each part of the client, and the license line: its state, hosts, fair use and contract end.

openlatch start

Starts the daemon. Refuses with OL-1501 when a daemon already runs on this machine.

openlatch stop

Stops the daemon. It also removes the agents' model relay wiring, and says so.

openlatch restart

Stops the daemon and starts it again, for example to apply a configuration change.

openlatch logs

Shows the client's event log.

openlatch doctor

Diagnoses the client's configuration and connectivity, and gives the cause and the remedy of every problem it finds. `--fix` applies the repairs it can, `--restore` undoes them, and `--rescue` writes a diagnostic file for support.

openlatch uninstall

Removes the agents' hooks and stops the daemon. `--agent` limits it to the agents you name.

openlatch docs

Opens the documentation in your browser.

openlatch system

Groups the commands that inspect or control one part of the client: `aip`, `auth`, `hooks`, `model-relay`, `proxy`, `supervision`, `telemetry` and `inventory`. Every remedy `openlatch doctor` gives names the exact system command to run.

openlatch system aip

Fetches and inspects the policy bundle resident on this machine, which holds the Autonomy Zone rules the client enforces.

openlatch system aip refresh

```
openlatch system aip update
```

Makes the running daemon fetch the policy bundle now, and waits up to 15 seconds for the result: changed, unchanged, pending, refused or offline, with the facts of the bundle in use. A pending result exits with code 7; a refusal or an offline daemon exits with code 1.

openlatch system auth

Groups the commands that connect the client to the platform.

openlatch system auth login

Signs the client in to the platform through your browser and stores the credential in the operating system's keychain.

openlatch system auth logout

Signs the client out and removes the stored credential.

openlatch system auth status

Shows whether the client is signed in, and where its credential comes from.

openlatch system auth switch

Moves this machine to another of your organizations.

openlatch system hooks

Groups the commands that manage agent hooks. `system hooks install` is the same as `openlatch init`.

openlatch system hooks install

Installs the agents' hooks; the same as `openlatch init`.

openlatch system hooks uninstall

Removes the agents' hooks; the same as `openlatch uninstall`.

openlatch system hooks status

Shows the hooks installed for each agent, in detail.

openlatch system model-relay

Groups the commands that inspect or switch the model relay, the local listener that records each agent's model calls.

openlatch system model-relay status

Reports whether the model relay is up, its port, and whether each agent's model calls go through it. An agent whose hooks work but whose model calls are not recorded reads as enforced, not measured.

openlatch system model-relay enable

Turns the model relay on in the configuration, then offers to restart the daemon to apply it.

openlatch system model-relay disable

Turns the model relay off in the configuration, then offers to restart the daemon to apply it.

openlatch system model-relay explain

Prints a model relay record by its id, read from local storage. Its content never leaves the machine.

openlatch system proxy

Groups the commands that inspect, discover, set or test the proxy every outbound request takes.

openlatch system proxy status

Shows the route outbound requests take, for the CLI and, when it differs, for the daemon.

openlatch system proxy discover

Runs the operating system's proxy discovery again, tests each candidate and keeps the one that works.

openlatch system proxy set

Sets the proxy by hand. A proxy set this way is never replaced by discovery or automatic repair.

openlatch system proxy clear

Removes the proxy and its credential, so requests go directly to their destination.

openlatch system proxy test

Tests the route step by step: the proxy connection, TLS, HTTP and a streamed response.

openlatch system supervision

Groups the commands that manage the operating-system supervisor (launchd, systemd or Task Scheduler) that starts the daemon at login and restarts it. Everything stays at user scope.

openlatch system supervision install

Installs the operating-system supervisor, so the daemon starts at login and after a reboot.

openlatch system supervision uninstall

Removes the operating-system supervisor. The daemon keeps running but no longer starts on its own.

openlatch system supervision status

Shows whether the supervisor is installed and running, and which one this system uses.

openlatch system supervision enable

Turns supervision back on after disable, and installs the supervisor again.

openlatch system supervision disable

Removes the supervisor and records that supervision is off on this machine.

openlatch system telemetry

Groups the commands that manage usage telemetry, which you can turn off.

openlatch system telemetry status

Shows whether usage telemetry is on, which setting decides it, and whether this build includes it.

openlatch system telemetry enable

Turns usage telemetry on.

openlatch system telemetry disable

Turns usage telemetry off.

openlatch system telemetry purge

Turns usage telemetry off and empties the queue of unsent events in a running daemon.

openlatch system telemetry debug

Shows how to run any command so it prints the telemetry events it would send.

openlatch system inventory

Groups the commands that inspect the configuration plane: the agent configuration sources the client watches, their change events, rescans and the watcher's health.

openlatch system inventory list

Lists the configuration sources the client watches, from the local audit log.

openlatch system inventory log

Shows the configuration change events in the local audit log.

openlatch system inventory rescan

Rescans every watched configuration source now.

openlatch system inventory status

Reports the configuration watcher's health: whether it is on, whether its list of sources loaded, and how many sources it holds.

openlatch system inventory inspect

Shows one configuration source in detail: its current fingerprint and its pending alerts.

openlatch system inventory projects

Lists the project folders the client knows from the configuration files it watches.

openlatch system inventory ack

Acknowledges a pending configuration alert, so the agent is no longer reminded of it.

openlatch update

Checks for a new client version. On a terminal, it then asks whether to install it, and installs it if you accept. Without a terminal (a script, a pipe, a management tool), it only reports the release. -- yes installs without asking; -- check only checks.

Error codes

Every code the client can print, with what happened and what to do.

- [OL-1002 – Event is too large](#)
- [OL-1003 – Duplicate event ignored](#)
- [OL-1100 – Custom pattern does not compile](#)
- [OL-1200 – Platform is unreachable](#)
- [OL-1201 – Platform rejected the credential](#)
- [OL-1202 – Platform rate limit reached](#)
- [OL-1204 – Outbox write failed](#)
- [OL-1205 – Outbox only partly sent](#)
- [OL-1206 – Platform channel is congested](#)
- [OL-1207 – Outbox entry set aside](#)
- [OL-1208 – Retained event expired](#)
- [OL-1210 – Policy bundle fetch failed](#)
- [OL-1211 – Policy bundle failed verification](#)
- [OL-1212 – Policy bundle cannot be used](#)
- [OL-1213 – Policy bundle refresh is overdue](#)
- [OL-1214 – One rule in the bundle was set aside](#)
- [OL-1215 – Local policy is switched off](#)
- [OL-1216 – New version cannot read the policy cache](#)
- [OL-1220 – No route to the platform](#)

- [OL-1221 – Proxy is unreachable](#)
- [OL-1222 – Proxy rejected the credential](#)
- [OL-1223 – Proxy scheme is not supported](#)
- [OL-1224 – TLS certificate not trusted](#)
- [OL-1225 – Proxy auto-config is unavailable](#)
- [OL-1226 – Proxy configuration is invalid](#)
- [OL-1227 – Direct connection not allowed](#)
- [OL-1228 – Proxy buffers streamed responses](#)
- [OL-1229 – No Kerberos ticket for the proxy](#)
- [OL-1240 – Evaluation input row is malformed](#)
- [OL-1241 – Evaluation bundle was rejected](#)
- [OL-1242 – No bundle to evaluate against](#)
- [OL-1243 – Session state does not match the bundle](#)
- [OL-1244 – Evaluation failed on one row](#)
- [OL-1250 – Platform paused ingest for licensing](#)
- [OL-1300 – Configuration value is invalid](#)
- [OL-1301 – Required configuration field is missing](#)
- [OL-1302 – Configuration changed but not applied](#)
- [OL-1400 – No supported agent found](#)
- [OL-1401 – Agent settings cannot be written](#)
- [OL-1402 – Agent settings file cannot be parsed](#)
- [OL-1403 – Other hooks found in agent settings](#)
- [OL-1404 – Hook binary cannot be installed](#)
- [OL-1405 – Hooks installed but not running](#)
- [OL-1406 – Agent TOML config cannot be parsed](#)
- [OL-1407 – Selected agent cannot take hooks](#)
- [OL-1408 – Expected Cline component is missing](#)
- [OL-1409 – Cline component state is unknown](#)
- [OL-1410 – Cline actions cannot be refused](#)
- [OL-1411 – Cline.app model calls not captured](#)
- [OL-1412 – OpenLatch plugin not loaded in Cline](#)

- [OL-1413 – Cursor hooks file has an unknown format](#)
- [OL-1414 – Cursor install cannot be confirmed](#)
- [OL-1415 – Cursor hook cannot reach the daemon](#)
- [OL-1416 – Some agents left without a daemon](#)
- [OL-1417 – Agent plugin waiting for its first load](#)
- [OL-1418 – Another file holds the plugin path](#)
- [OL-1419 – Agent version below the supported floor](#)
- [OL-1420 – Agent component state is unknown](#)
- [OL-1421 – Agent plugin cannot reach the daemon](#)
- [OL-1422 – Agent plugin did not load](#)
- [OL-1423 – Agent relay not yet proven](#)
- [OL-1424 – Agent surface not covered by the relay](#)
- [OL-1425 – Agent command-line tool bypasses the relay](#)
- [OL-1500 – Port already in use](#)
- [OL-1501 – Daemon already running](#)
- [OL-1502 – Daemon did not become healthy](#)
- [OL-1503 – Newer client version available](#)
- [OL-1504 – Update could not be applied](#)
- [OL-1505 – Built from source, cannot self-update](#)
- [OL-1506 – Daemon did not answer the update](#)
- [OL-1507 – Daemon could not be stopped](#)
- [OL-1510 – Part of the daemon restarted](#)
- [OL-1511 – Part of the daemon keeps failing](#)
- [OL-1512 – Daemon is degraded](#)
- [OL-1513 – Nothing restarts the daemon](#)
- [OL-1600 – No credential found](#)
- [OL-1601 – Credential expired](#)
- [OL-1602 – Keychain is unavailable](#)
- [OL-1603 – Keychain denied access](#)
- [OL-1604 – Encrypted credential file failed](#)
- [OL-1605 – Sign-in timed out](#)

- [OL-1606 – Sign-in failed](#)
- [OL-1607 – Key revocation failed at sign-out](#)
- [OL-1608 – Organization must be chosen](#)
- [OL-1609 – Organization picker left unanswered](#)
- [OL-1610 – Organization not found](#)
- [OL-1611 – Organization name is ambiguous](#)
- [OL-1612 – API key belongs to another organization](#)
- [OL-1613 – Organization could not be confirmed](#)
- [OL-1614 – Machine is enrolled in another organization](#)
- [OL-1615 – Organization switch not applied](#)
- [OL-1616 – Switch refused: the key comes from the environment](#)
- [OL-1617 – Sign-in code expired](#)
- [OL-1618 – Account has no organization](#)
- [OL-1619 – Organization is unavailable](#)
- [OL-1620 – Too many sign-ins in progress](#)
- [OL-1700 – Telemetry consent file is corrupt](#)
- [OL-1701 – Telemetry consent file cannot be written](#)
- [OL-1702 – Telemetry batch not sent](#)
- [OL-1703 – Telemetry could not start](#)
- [OL-1704 – Telemetry hand-off refused](#)
- [OL-1800 – Doctor repair journal is corrupt](#)
- [OL-1801 – Nothing to restore](#)
- [OL-1802 – Diagnostic file cannot be written](#)
- [OL-1803 – Diagnostic file is incomplete](#)
- [OL-1900 – Hook integrity key unavailable](#)
- [OL-1901 – Hook state file is corrupt](#)
- [OL-1902 – Hook state file cannot be written](#)
- [OL-1903 – Hook marker version not supported](#)
- [OL-1904 – Hook entry could not be processed](#)
- [OL-1905 – Older hook marker upgraded](#)
- [OL-1906 – Changed hooks were repaired](#)

- [OL-1910 – Settings file cannot be replaced](#)
- [OL-1911 – Settings file link cannot be resolved](#)
- [OL-1950 – Supervision not supported on this system](#)
- [OL-1951 – Supervisor could not be installed](#)
- [OL-1952 – Supervisor could not be registered](#)
- [OL-1953 – Supervisor refused a command](#)
- [OL-2000 – Configuration watcher could not start](#)
- [OL-2001 – Configuration source list not found](#)
- [OL-2002 – Configuration source list is invalid](#)
- [OL-2003 – Watched path cannot be expanded](#)
- [OL-2004 – One path cannot be watched](#)
- [OL-2005 – Configuration file could not be fingerprinted](#)
- [OL-2006 – Configuration rescan was partial](#)
- [OL-2007 – Session start details unavailable](#)
- [OL-2008 – Configuration watching is off](#)
- [OL-9999 – Unexpected internal error](#)
- [OL-RELAY-CA – Model relay certificate not trusted](#)
- [OL-RELAY-CURSOR-UNPROVEN – Cursor relay not yet proven](#)
- [OL-RELAY-CURSOR-UNSUPPORTED – Cursor surface not covered by the relay](#)
- [OL-RELAY-DOWN – Model relay is not running](#)
- [OL-RELAY-FOREIGN – Another process holds the relay port](#)
- [OL-RELAY-IO – Model relay input or output error](#)
- [OL-RELAY-MISCONFIG – Relay setting is overridden](#)
- [OL-RELAY-NEXTSTART – Relay applies at the editor's next start](#)
- [OL-RELAY-NOFIND – No local record for this id](#)
- [OL-RELAY-PORT – Model relay port is in use](#)
- [OL-RELAY-PORTS – No relay port for a provider](#)
- [OL-RELAY-PREFLIGHT – Model relay could not reach the provider](#)
- [OL-RELAY-PROVIDER – Agent config holds another provider entry](#)
- [OL-RELAY-SERVE – Model relay stopped on an error](#)
- [OL-RELAY-SETTINGS – Agent settings hold your own proxy](#)

- OL-RELAY-STATEFILE – Agent state file cannot be edited safely.
- OL-RELAY-TRUSTPROOF – Relay trust not yet proven here
- OL-RELAY-UNCOVERED – Provider not carried by the relay.

Changelog

Product

All products



48 entries

Atom: [all](#) · [client](#) · [platform](#)

October 13, 2026 Client

Client 0.6.18

Choose your organization at sign-in, and move a machine to another organization.

- Signing in asks which organization to use when your account belongs to several; `--org` or `OPENLATCH_ORG` names it without asking.
- New command `openlatch system auth switch` moves this machine to another of your organizations.
- On Windows, `openlatch doctor` reports an agent's command-line tool whose machine-wide install bypasses the relay (OL-1425).
- Groundwork for a coding agent that OpenLatch does not yet support. Nothing changes for the agents already wired.

October 12, 2026 Client

Client 0.6.17

Cursor returns to coverage once its windows reload.

- A held Cursor is covered again once its windows reload.
- The relay records each Cursor call once, and classifies the traffic of a coding agent that OpenLatch does not yet support.

October 12, 2026 Client

Client 0.6.16

The client also connects the Codex desktop app, with no command-line tool and no new `openlatch init`.

- The client connects the Codex desktop app on a machine with no Codex command-line tool, without running `openlatch init` again.
 - `openlatch doctor` no longer reports a false finding for a coding agent that OpenLatch does not yet support.
-

October 12, 2026 Platform

One policy in several Autonomy Zones

Attach an Agent Intent Policy to several zones, each with its own mode.

October 11, 2026 Client

Client 0.6.15

Cursor and Cline command-line tools reach the relay on Windows with no setup.

- On Windows, the Cursor and Cline command-line tools route their model traffic through the relay with no setup.
 - On Windows, the relay's certificate check no longer reads a file that is still open.
-

October 11, 2026 Client

Client 0.6.14

Groundwork for a coding agent that OpenLatch does not yet support.

- Relay groundwork for a coding agent that OpenLatch does not yet support. Nothing changes for the agents already wired.
-

October 10, 2026 Client

Client 0.6.13

Groundwork for a coding agent that OpenLatch does not yet support, and a Windows relay fix.

- Hook and inventory groundwork for a coding agent that OpenLatch does not yet support. Nothing changes for the agents already wired.
 - On Windows, the relay's certificate is checked in the user's own context, and background processes start without opening a console window.
-

October 10, 2026 Client

Client 0.6.12

More accurate model usage and PowerShell reading.

- `openlatch doctor --fix` reports the state it leaves behind.
 - Usage that an interrupted model stream already reported is kept.
 - A model is marked unknown only when its usage is estimated.
 - An API key is told apart from a subscription token by its shape.
 - A PowerShell parameter written as `-Param:vaLue` is read the same way as `-Param vaLue`.
-

October 9, 2026 Client

Client 0.6.11

PowerShell commands are read more accurately.

- A URL in a PowerShell command is now read as a URL, not as a file path.
-

October 8, 2026 Client

Client 0.6.10

Fixes for enrolment and for reading PowerShell commands.

- The daemon's idle health check now sends the API key.
 - `openlatch init` now saves an exported `OPENLATCH_API_URL` to the client configuration.
 - A file path is classified where it resolves, not as it is spelled.
 - PowerShell commands are split into words by PowerShell's own rules.
-

October 8, 2026 Platform

Autonomy Zones on your organization tree

Enable an Autonomy Zone on any unit, and see which agents each zone governs and why.

October 8, 2026 Platform

Autonomy Zones and Agent Intent Policies

Define what each agent may do as Agent Intent Policies, and run each one in Disabled, Monitor or Enforce mode.

October 8, 2026 Platform

Public REST API

Automate the console's operations with organization API keys.

October 7, 2026 Client

Client 0.6.9

Live configuration reports, and fixes for Cline, Cursor and macOS.

- Configuration changes on a host are reported as they happen.
 - A Claude Code teammate session is linked to the session that leads it.
 - The skills, rules and MCP servers that Codex CLI, Cursor and Cline load are inventoried as each agent actually loads them.
 - Cursor reports name the MCP server on every call.
 - Updating Cline.app no longer interrupts a conversation.
 - A Claude Code session started from a plan records that plan as its first prompt.
 - On macOS, the client no longer asks for access to the Desktop, Downloads and Documents folders.
 - `openlatch init` no longer describes usage telemetry as anonymous.
 - Native Windows paths are classified correctly on file reads and writes.
-

October 6, 2026 Platform

Database-enforced tenant isolation

The database itself keeps each organization's data apart.

October 6, 2026 Client

Client 0.6.8

Policy bundle and relay fixes.

- A cached policy bundle is revalidated correctly when the platform reports it unchanged.
 - Usage from sessions that no hook registered is no longer recorded.
-

October 5, 2026 Client

Client 0.6.7

Path-aware evaluation and Cline fixes.

- An action's effects are correlated with the paths it targets.
 - The Cline plugin applies to every tool, not only the shell.
 - `openlatch init` waits for the first Cline plugin delivery instead of failing.
 - Each action is decided once, not again after it runs.
 - Codex CLI side threads are filed under their parent session.
 - Usage is filed under the right installation.
-

October 5, 2026 Platform

Policies that read what an action does

Intent policies match what a command does, such as reading secret material or reaching a network host, not how it is spelled.

October 2, 2026 Client

Client 0.6.6

The Cursor IDE's model calls go through the relay.

- The Cursor IDE's model calls are routed through the relay.
 - A fresh `openlatch init` no longer fails on components that are still starting.
 - `openlatch doctor` shows one status line for every agent and recognizes the Cline.app hub.
 - A shell process substitution is no longer read as a redirection target.
-

October 1, 2026 Client

Client 0.6.5

One shared tool vocabulary, and Ask verdicts delivered per agent.

- Each agent's tools map onto one shared vocabulary; a tool with no mapping is still recorded under its own name.
 - Policy evaluation reads one classification catalogue from the policy bundle.
 - Ask verdicts reach each agent in the form it supports, with an authored verdict where an agent has no ask of its own.
-

October 1, 2026 Client

Client 0.6.4

Cursor support.

- Policies decide on Cursor actions, and the verdict reaches Cursor in its own format.
 - Cursor's model traffic goes through the relay and is recorded as usage, never as content.
 - Cursor's wiring is checked, repaired and inventoried like every other agent's.
 - `openlatch uninstall` names the agents a partial uninstall leaves wired.
-

September 30, 2026 Client

Client 0.6.3

The Cline plugin reaches Cline.app and VS Code.

- The Cline plugin is delivered to Cline.app and to VS Code.
 - The client checks that the plugin loaded on each host.
-

September 29, 2026 Platform

License and usage in the console

Read what your organization is licensed for against what it consumes, on one page.

September 29, 2026 Platform

Agent ownership

Every agent resolves to an owner from your directory, and its business function can drive a policy.

September 29, 2026 Platform

Agents you can recognize

Agents carry a readable name and show the sessions and projects they bring together.

September 29, 2026 Platform

Execution graph

See everything an agent did as one graph, filter it, and start a rule from a command.

September 29, 2026 Platform

Agent economics

See each agent's spend, forecast and sessions, with ranked savings opportunities.

September 29, 2026 Client

Client 0.6.2

Cline reporting, self-repair and a complete purge.

- Cline sessions, prompts and tool calls are reported to the platform.
 - Relay wiring, hook trust and the Cline plugin files are checked for tampering and repaired.
 - The client reports when a policy bundle is waiting to be delivered.
 - `openlatch uninstall` sends an uninstall event, and `openlatch uninstall --purge` removes all client state.
-

September 29, 2026 Client

Client 0.6.1

Cline through the relay, and relay certificate management.

- Cline's model traffic is captured through the relay, including the Cline desktop app.
 - The relay proves its certificate is trusted before an agent is pointed at it.
 - The relay's certificate can be reported, rotated, released and removed.
 - Only detected agents are reported in host facts.
 - Codex CLI hooks are trusted on Windows.
-

September 29, 2026 Platform

Policies that keep deciding offline

Each client holds its policy bundle locally, so enforcement continues with the network unplugged.

September 24, 2026 Client

Client 0.6.0

A step-by-step `openlatch update`, with one breaking change to its JSON output.

- Breaking: `openlatch update --check --json` no longer returns `min_supported_client`. `current`, `latest` and `severity` are unchanged.
 - `openlatch update` shows its progress step by step, and the daemon comes back after it.
 - Cline hooks, rules and workflows are reported by the configuration inventory.
-

September 23, 2026 Client

Client 0.5.8

Cline prompt capture.

- Prompts are captured on the Cline CLI and the Cline desktop app.
 - Agent types that are not wired are no longer reported.
-

September 23, 2026 Client

Client 0.5.7

Terminal output fix on Windows.

- Colors and progress display correctly on Windows for every command, not only `openlatch init`.
-

September 23, 2026 Client

Client 0.5.6

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.
-

September 22, 2026 Client

Client 0.5.4

Daemon supervision and telemetry delivery fixes.

- On Windows, the scheduled task that keeps the daemon running keeps its version marker.
 - The daemon's first telemetry batch is sent over HTTPS.
-

September 22, 2026 Client

Client 0.5.3

Telemetry through the platform, and Linux binaries for older hosts.

- Telemetry and crash reports go through the platform's own ingestion endpoint.
 - Linux binaries start on slim and older hosts.
-

September 21, 2026 Client

Client 0.5.2

Startup and event delivery fixes.

- On Windows, `openlatch` runs as a console program, and its binaries start on a fresh host.
 - Events still queued at shutdown are sent.
 - The daemon notices a new login without a restart.
 - Event delivery recovers after every reconnection to the platform.
-

September 20, 2026 Client

Client 0.5.1

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.
-

September 8, 2026 Client

Client 0.4.0

Subsystem commands move under `openlatch` system.

- Breaking: `openlatch auth`, `hooks`, `model-relay`, `proxy`, `supervision`, `telemetry` and `inventory` are now `openlatch system <name>`, with no aliases.
 - Breaking: the `[boundary]` configuration section is now `[model_relay]`, `OPENLATCH_BOUNDARY_*` variables are now `OPENLATCH_MODEL_RELAY_*`, and `OL-BND-*` error codes are now `OL-RELAY-*`.
 - `openlatch doctor` reports the live state of each hook.
 - Proxy diagnostics report the proxy the client actually uses.
-

September 6, 2026 Client

Client 0.3.3

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.
-

August 28, 2026 Client

Client 0.2.2

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.
-

August 19, 2026 Client

Client 0.1.18

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.
-

April 23, 2026 Client

Client 0.1.14

Default API address fix.

- The default API address no longer repeats `/api`.
-

April 23, 2026 Client

Client 0.1.13

The npm package resolves with `npx @openlatch/client init`.

- The npm package gains a `client` entry point, so `npx @openlatch/client init` resolves.
 - Help output no longer prints a banner.
-

April 23, 2026 Client

Client 0.1.12

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.
-

April 23, 2026 Client

Client 0.1.8

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.
-

April 23, 2026 Client

Client 0.1.7

The npm package ships the hook entry point.

- The npm package installs the `openlatch-hook` entry point.
-

April 23, 2026 Client

Client 0.1.6

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.
-

OL-1002 — Event is too large

What happened

Event body exceeds the 1 MB size limit.

What to do

Run `openlatch doctor` to check the client. If the error repeats, create a diagnostic file with `openlatch doctor --rescue` and send it to support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1003 — Duplicate event ignored

What happened

Event was deduplicated within the TTL window (informational).

What to do

Nothing to do. If you see it often, run `openlatch doctor` to check that each agent's hooks are installed once.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1100 — Custom pattern does not compile

What happened

A custom regex pattern in `config.toml` failed to compile.

What to do

Correct the pattern in `~/openlatch/config.toml`, then run `openlatch restart` to load it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1200 — Platform is unreachable

What happened

Cloud API endpoint is unreachable (network error or DNS failure).

What to do

Run `openlatch system proxy test` to check the route from this machine to the platform.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1201 — Platform rejected the credential

What happened

Cloud returned 401/403 — API key is invalid or revoked.

What to do

Sign in again with `openlatch system auth login`, or set a valid key in `OPENLATCH_API_KEY` and run `openlatch init`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1202 — Platform rate limit reached

What happened

Cloud returned 429 – rate limit exceeded; respect Retry-After header.

What to do

Nothing to do if it clears on its own. If it persists, run `openlatch status` and share the output with support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1204 — Outbox write failed

What happened

Durable outbox write failure (disk full, permission denied, I/O error). The daemon continues serving events — the outbox is a recovery aid, not a correctness mechanism — but the event is lost for cloud replay.

What to do

Free disk space or fix the permissions on `~/ .openlatch`, then run `openlatch doctor` to confirm the outbox is writable.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1205 — Outbox only partly sent

What happened

Outbox drain completed with some entries still pending (cloud transient failure mid-replay, or I/O error reading the spool). The remaining entries survive on disk and are retried on the next drain signal.

What to do

Nothing to do if the next attempt succeeds. If entries keep waiting, run `openlatch doctor`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1206 — Platform channel is congested

What happened

Cloud forwarding channel is under sustained congestion — emergency mode engaged. Live hook events were being dropped faster than the worker could drain; replay is paused and the fallback queue may be evicted to restore headroom. Cleared automatically when drops cease for two consecutive health ticks.

What to do

Run `openlatch status` to watch the event counts, and `openlatch system proxy test` if the connection to the platform is slow.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1207 — Outbox entry set aside

What happened

An outbox entry failed repeatedly within the current daemon lifetime and was quarantined so the queue can drain past it. The entry is dropped from the outbox without ever reaching the cloud – only emitted after the per-id attempt count crosses `OUTBOX_MAX_ATTEMPTS`.

What to do

Run `openlatch doctor --rescue` and send the diagnostic file to support so the entry can be examined.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1208 — Retained event expired

What happened

A retained compatibility event reached its explicit durable expiry before it could be delivered. This is terminal expiry, not auth or transport.

What to do

Run `openlatch doctor` to check the connection to the platform, so events are delivered before they expire.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1210 — Policy bundle fetch failed

What happened

A poll for the policy bundle failed (network error, 5xx, or 401/403). The previously activated bundle is retained and keeps enforcing — fail-static, never fail-open. Warn level for transient failures, error level for auth failures.

What to do

Run `openlatch doctor` to find the cause, then `openlatch system aip refresh` to fetch the bundle again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1211 — Policy bundle failed verification

What happened

A fetched or on-disk bundle failed verification and was NOT activated: digest mismatch against the ETag, an `organization_id` that is not ours, a 200 carrying no ETag header (the digest is not inside the body, so an unverifiable body must never activate), or a cached `bundle.json` whose bytes no longer match `bundle.meta.json`. The previous bundle stays resident; the stored validator is deliberately NOT advanced, so the next poll re-requests and re-attempts activation.

What to do

Run `openlatch system aip refresh` to fetch a fresh copy. If the error repeats, run `openlatch doctor --rescue` and send the file to support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1212 — Policy bundle cannot be used

What happened

A policy bundle could not be read: it is malformed, it uses a format version this client does not know, or it carries a signature this client cannot check.

What to do

Update the client with `openlatch update`, then run `openlatch system aip refresh`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1213 — Policy bundle refresh is overdue

What happened

No successful poll within `stale_warn_after_secs`. Warning only — the resident bundle keeps enforcing. Measured from the last successful poll (connectivity), not from `built_at` (policy age), and a 304 counts as a successful poll.

What to do

Check the machine's connection to the platform with `openlatch doctor`. Once it is restored, run `openlatch system aip refresh` to fetch the latest bundle at once.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1214 — One rule in the bundle was set aside

What happened

One rule inside an otherwise-good bundle was acted on individually at projection time — dropped, or held in a weaker mode — and the rest of the bundle activated normally. Deliberately NOT OL-1212, which means **the bundle was rejected and the previous one retained** — reusing it would invert its meaning for anyone alerting on it.

What to do

Update the client with `openlatch update` so it supports every rule in the bundle, then run `openlatch system aip refresh`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1215 — Local policy is switched off

What happened

The policy plane is switched off in config, so no rule is evaluated and every action is allowed.

What to do

Remove `enabled = false` from the `[policy]` section of `~/.openlatch/config.toml` (or unset `OPENLATCH_POLICY_ENABLED`), then run `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1216 — New version cannot read the policy cache

What happened

A candidate executable cannot load the exact resident policy cache and therefore may not replace the evaluator that can.

What to do

Run `openlatch system aip refresh`, then try the update again with `openlatch update`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1220 — No route to the platform

What happened

The platform could not be reached at all: no proxy is configured, and the direct connection to the platform failed at the TCP/DNS level. Distinct from OL-1221 below — that code fires when a **proxy** is in the path and unreachable, whereas this one means egress never had a proxy hop to blame. Check outbound DNS resolution and firewall rules for the host running the daemon.

What to do

Check DNS resolution and the firewall rules for this machine, then run `openlatch system proxy test` to confirm the route.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1221 — Proxy is unreachable

What happened

A proxy is configured, but the proxy itself did not answer: the TCP connect to the proxy host failed, or an HTTPS CONNECT through it timed out or was refused. The destination is never reached because the client cannot even establish the tunnel to it. Verify the proxy host and port are correct and reachable from this machine before suspecting the destination.

What to do

Check the proxy host and port with `openlatch system proxy status`, then run `openlatch system proxy test`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1222 — Proxy rejected the credential

What happened

The proxy answered `407 Proxy Authentication Required` and either the credentials the client offered were rejected, or it had none to offer. Distinct from `OL-1201/ERR_CLOUD_AUTH_FAILED` — that is the **platform** rejecting our API key; this is the proxy's own gate, tripped before the request ever reaches the platform. A credential that was valid yesterday and a corporate proxy that rotated its password look identical on the wire, so the operator needs to reconfirm the proxy credential specifically, not the platform API key.

What to do

Set the current proxy credential with `openlatch system proxy set`, then run `openlatch system proxy test`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1223 — Proxy scheme is not supported

What happened

The configured proxy URL names a scheme this build cannot speak: anything outside `http/https/socks5/socks5h`, or `negotiate` requested against a build compiled without the Kerberos feature. Fails fast at proxy resolution rather than silently falling through to a plain connect — a silent downgrade would route traffic around the very proxy the operator configured. Fix the scheme in the proxy URL, or rebuild with the feature that scheme requires.

What to do

Set a proxy URL that starts with `http`, `https`, `socks5` or `socks5h` with `openlatch system proxy set`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1224 — TLS certificate not trusted

What happened

The TLS handshake to the destination failed certificate verification. The most common cause on an enterprise network is a TLS-inspecting proxy or firewall re-signing outbound traffic with an interception CA that is not in the OS trust store and was not supplied via `ca_bundle` — from the client's point of view that looks identical to a real machine-in-the-middle, which is exactly why it refuses the handshake instead of guessing. The fix is almost always to point `ca_bundle` at the interception root, never to disable verification.

What to do

Point `OPENLATCH_CA_BUNDLE` (or `ca_bundle` in the `[proxy]` section) at your network's root certificate, then run `openlatch system proxy test`. Never turn verification off.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1225 — Proxy auto-config is unavailable

What happened

A PAC (Proxy Auto-Config) or WPAD source is configured, but the OS could not evaluate it: the PAC URL did not resolve or serve a script, or the platform's PAC engine is unavailable on this OS. Egress falls back to whatever static proxy or direct behaviour the config declares beneath the PAC entry, so this code is informational unless that fallback also fails.

What to do

Run `openlatch system proxy discover` to find a working route, or set one by hand with `openlatch system proxy set`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1226 — Proxy configuration is invalid

What happened

The proxy configuration itself is malformed: an unparsable proxy URL, a scheme missing a credential field it requires, or `allow_direct` and the proxy settings contradicting each other. Unlike every other code in this block — each describing a runtime network condition that never blocks startup — this one DOES fail startup: a config the client cannot even parse is an authoring bug, not a transient network state, and letting the daemon start with it would only move the failure to the first request.

What to do

Correct the proxy with `openlatch system proxy set`, or remove it with `openlatch system proxy clear`, then run `openlatch start`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1227 — Direct connection not allowed

What happened

`allow_direct = false` and no usable proxy could be found for this destination. Egress refuses the request rather than silently falling back to a direct connection — on a network where policy requires all egress to transit a proxy, a silent direct fallback is an exfiltration path, not a convenience. Fix the proxy configuration, or set `allow_direct = true` if direct egress from this host is actually acceptable.

What to do

Fix the proxy with `openlatch system proxy discover` or `openlatch system proxy set`. Set `allow_direct = true` only if direct connections from this machine are acceptable.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1228 — Proxy buffers streamed responses

What happened

The proxy buffers a streaming response instead of passing bytes through as they arrive: the `CONNECT` tunnel itself works, but a Server-Sent Events stream piles up behind the proxy's internal buffer and lands in one burst instead of streaming. This is a proxy-behaviour diagnosis, not a connectivity failure — the request still succeeds — so it is surfaced as a warning rather than failing the call. Look for a response-buffering or content-inspection setting on the proxy for the streaming endpoint.

What to do

Ask your network team to turn off response buffering or content inspection for the platform on the proxy, then run `openlatch system proxy test`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1229 — No Kerberos ticket for the proxy

What happened

negotiate (Kerberos/SPNEGO) was selected for proxy auth, but no usable ticket could be minted: no Kerberos credential cache, an expired TGT, or the proxy's SPN not resolving from this host. Distinct from OL-1222/ERR_PROXY_AUTH_FAILED — the proxy was never even offered a token to accept or reject. The fix is host-side: run `kinit`, or confirm the machine is domain-joined so Windows can obtain a ticket automatically.

What to do

Run `kinit`, or check that the machine is joined to the domain, then run `openlatch system proxy test`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1240 — Evaluation input row is malformed

What happened

A row frame that did not deserialise into a request row: an event that is not an object, a now_ms that is not an integer, a state_in that is not a register file. ****One row, never the stream**** — the frame is answered with an error response and the run continues, so a replay reports "all but four events evaluated" instead of dying midway.

What to do

Correct the row named in the output and run the evaluation again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1241 — Evaluation bundle was rejected

What happened

Stage 1 of the bundle parse failed, so **nothing loaded**: the envelope did not deserialise, or its `client_floor` is above what this engine implements. Written as a `bundle_error` frame, which is a different outcome from a `bundle_ack` carrying a non-empty `skipped` — that one means the bundle IS loaded and one artifact is not.

What to do

Update the client with `openlatch update`, then run the evaluation again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1242 — No bundle to evaluate against

What happened

A row arrived with no bundle to evaluate it against: no header frame has been accepted and the row carries no inline bundle of its own. Answered per row, because the next header frame may well fix it.

What to do

Send the policy bundle before the rows, or include a bundle with the row.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1243 — Session state does not match the bundle

What happened

The row's `state_in` disagrees with the bundle's `state_layout`. ****Malformed, never padded**** — two implementations cannot agree about which counter is which if one of them silently invents a slot.

What to do

Supply the session state the client returned for the previous row of the same session.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1244 — Evaluation failed on one row

What happened

The engine panicked while evaluating one row.

What to do

Run `openlatch doctor --rescue` and send the diagnostic file to support with the row that failed.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1250 — Platform paused ingest for licensing

What happened

The platform refused ingest on **licensing** grounds: a 402, or a 503 whose body names a site-licence code.

What to do

Follow the licensing link shown by `openlatch status`, or contact your OpenLatch representative if none is shown.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1300 — Configuration value is invalid

What happened

The configuration file contains an invalid value.

What to do

Run `openlatch doctor` to find the value, correct it in `~/.openlatch/config.toml`, then run `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1301 — Required configuration field is missing

What happened

A required configuration field is absent and has no default.

What to do

Run `openlatch init` to write a complete configuration again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1302 — Configuration changed but not applied

What happened

The file on disk has changed since the running daemon read it, so the process is still serving the previous values. Nothing is wrong with the file — `openlatch system proxy set` and its siblings write it correctly and say so — and nothing is wrong with the daemon. The two are simply out of step, and only a restart closes the gap.

What to do

Run `openlatch restart` to apply the new configuration.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1400 — No supported agent found

What happened

No supported AI agent was detected on this machine.

What to do

Install a supported agent, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1401 — Agent settings cannot be written

What happened

Cannot read or write the agent's settings.json (permissions, I/O error).

What to do

Fix the permissions on the agent's settings file, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1402 — Agent settings file cannot be parsed

What happened

The settings.json file contains malformed JSONC that cannot be parsed.

What to do

Fix the syntax error in the agent's settings.json, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1403 — Other hooks found in agent settings

What happened

Existing non-OpenLatch hooks detected in settings.json (warning, non-blocking).

What to do

Nothing to do unless those hooks are unexpected. Run `openlatch system hooks status` to see every installed hook.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1404 — Hook binary cannot be installed

What happened

No `openlatch-hook` binary could be staged into `<ol_dir>/bin/`, so any hook command written now would resolve to a bare name and die with exit 127 on every agent tool call. Fatal for `init` — an install that cannot resolve its own hook binary is not a successful install — and reported (not applied) by `doctor -fix`.

What to do

Check the permissions and free space in `~/ .openlatch`, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1405 — Hooks installed but not running

What happened

The agent's hooks are installed and correct on disk, and the agent will nonetheless never run them — so nothing is enforced. Three ways to reach it, one code and three remedies: our hook binary does not resolve; the agent has not been told to trust the hook (Codex marks a new or changed hook command for review and skips it until the developer runs `/hooks`); or an administrative setting suppresses non-managed hooks wholesale.

What to do

Run `openlatch doctor`: it names which of the three causes it found and the step that fixes it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1406 — Agent TOML config cannot be parsed

What happened

An agent config file that is TOML — Codex CLI's `config.toml` — is present and is not valid TOML, so the model-relay provider table can be neither written nor reversed without destroying what is already in it.

What to do

Fix the syntax error in the agent's `config.toml`, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1407 — Selected agent cannot take hooks

What happened

The agent is installed on this machine and its hook surface cannot be written into at all, so `--agent` selected nothing this command can act on.

What to do

Run the command again without `--agent`, or name an agent this version supports.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1408 — Expected Cline component is missing

What happened

A Cline surface this build expected and did not find — the store root, the Hooks/ directory, plugins/, the MCP registry, or a Cline-lineage extension in any known editor root.

What to do

Run `openlatch doctor` to see which part is missing, install or update Cline, then run `openlatch init`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1409 — Cline component state is unknown

What happened

A Cline surface whose state could not be determined.

What to do

Run `openlatch doctor` to see which path failed, and fix its permissions.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1410 — Cline actions cannot be refused

What happened

Cline is installed, its ten hook scripts are on disk and capturing, and **nothing they observe can be refused**.

What to do

Run `openlatch doctor` to see why the plugin is not loaded, and follow the remedy it gives.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1411 — Cline.app model calls not captured

What happened

Cline.app's model traffic is not going through the relay: its hub runs without this install's capture (started before it, or from an older Cline.app), the hub port is held by something else, the hub is managed on a platform where that is unproven (D-47), the platform has no established install path so the app cannot be detected (A-8), or 06 §0 proved the app cannot be captured (§F). One condition, "Cline.app is not captured"; the reason is the detail line.

What to do

Close every Cline conversation and quit Cline.app, then open it again so its hub restarts through the relay.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1412 — OpenLatch plugin not loaded in Cline

What happened

A Cline host (Cline.app's hub, the VS Code extension) cannot or does not load the OpenLatch plugin, so a verdict cannot be delivered there and the host is monitor-only through the file hooks. One condition; the reason is the detail: host-too-old, delivery-off, probe-failed, pending-restart, runtime-missing, windows-launcher-unproven, not-managed, customer-override, not-detected-on-platform, windows-launcher-missing. Off is never a pass (cli-output-contract).

What to do

Run `openlatch doctor` for the reason. If plugin delivery is off, set `plugin_delivery = true` in the `[cline]` section of `~/.openlatch/config.toml` and run `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1413 — Cursor hooks file has an unknown format

What happened

Cursor's `hooks.json` is not a file this build can safely write into: it does not parse, or its root `version` is present and is not the integer 1 Cursor documents. Install refuses rather than writing into a format it does not know — a guess there could stop the whole file loading, the developer's own hooks included. The remedy is to fix or move the file and re-run `openlatch init`.

What to do

Fix the file or move it aside, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1414 — Cursor install cannot be confirmed

What happened

A Cursor surface (the IDE or the CLI) has an install location on this host that cannot be confirmed as Cursor: the application is present but its `resources/app/package.json` is missing, unreadable or names no Cursor product, or the CLI's `versions` directory exists but cannot be read. `doctor` reports it as degraded, naming the surface and what it could not read. The version is read from files only — `doctor` never runs Cursor, whose macOS binary launches the GUI.

What to do

Run `openlatch doctor` to see which file it could not read, then reinstall or update Cursor.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1415 — Cursor hook cannot reach the daemon

What happened

Cursor ran the OpenLatch hook, but the hook could not reach the daemon. Cursor's events reach the client only later, from the fallback log, and every Cursor action proceeds without a decision until this is fixed.

What to do

Run `openlatch start` to make sure the daemon runs, then `openlatch doctor`. Cursor's own hook errors are in its Hooks output channel.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1416 — Some agents left without a daemon

What happened

`openlatch uninstall --agent unwired` only some of the agents on this host and, as documented, still stopped the daemon and persisted `supervision mode = disabled` machine-wide (#325: a running daemon reads the removal as tampering and reinstalls it). Every agent it left out is still wired to a daemon that is no longer running, so each of its hook calls fails open to `fallback.jsonl` and nothing is enforced.

What to do

Run `openlatch start`, and `openlatch system supervision enable` if the daemon was supervised.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1417 — Agent plugin waiting for its first load

What happened

This code comes from the client's integration with OpenCode, an agent that is not yet supported. The client wrote the agent's plugin file and no load has been reported since. The agent loads plugins only when its server starts, so the check stays pending until then.

What to do

Restart the agent, then run `openLATCH doctor` to confirm the check has cleared.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1418 — Another file holds the plugin path

What happened

This code comes from the client's integration with OpenCode, an agent that is not yet supported. A file OpenLatch did not write sits where the client would write the agent's plugin, and the client never overwrites it.

What to do

Move that file elsewhere, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1419 — Agent version below the supported floor

What happened

This code comes from the client's integration with OpenCode, an agent that is not yet supported. The installed agent version is older than the minimum the client expects.

What to do

Upgrade the agent to a recent version, then run `openlatch doctor`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1420 — Agent component state is unknown

What happened

This code comes from the client's integration with OpenCode, an agent that is not yet supported. The agent is present, but the client could not establish the state of one of its components.

What to do

Run `openlatch doctor` to see which component it could not check.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1421 — Agent plugin cannot reach the daemon

What happened

This code comes from the client's integration with OpenCode, an agent that is not yet supported. The check that the agent's plugin can reach the daemon failed.

What to do

Run `openlatch doctor --fix`, then restart the agent.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1422 — Agent plugin did not load

What happened

This code comes from the client's integration with OpenCode, an agent that is not yet supported. The agent sent model calls through the relay, but its plugin never reported loading, so the plugin may have been skipped.

What to do

Restart the agent and check that it is not started with `--pure`, which skips every plugin.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1423 — Agent relay not yet proven

What happened

This code comes from the client's integration with OpenCode, an agent that is not yet supported. The model relay is wired for the agent, but no model call from it has been recorded yet.

What to do

Run the agent once from a shell that loads the OpenLatch environment (`env.sh`, or `env.ps1` in PowerShell).

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1424 — Agent surface not covered by the relay

What happened

This code comes from the client's integration with OpenCode, an agent that is not yet supported. One surface or transport of the agent cannot go through the model relay, so its model calls go directly to the provider. The doctor output names the reason.

What to do

Run `openlatch doctor` to see which surface is affected.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1425 — Agent command-line tool bypasses the relay

What happened

On Windows, the command-line tool of a wired agent, such as the Cursor or Cline CLI, resolves through the machine PATH to an executable other than the OpenLatch launcher. A machine-wide install shadows the launcher, so that tool's model calls go directly to the provider. The doctor only reads paths and file metadata: it never runs the executable and never changes the machine PATH.

What to do

Remove the executable that `openlatch doctor` names, or ask an administrator to move the machine PATH entry after the user Path entry. Then run `openlatch doctor` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1500 — Port already in use

What happened

The selected port is already in use by another process.

What to do

Free the port, or choose another one with `OPENLATCH_PORT`, then run `openlatch start`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1501 — Daemon already running

What happened

A daemon instance is already running on this machine.

What to do

Run `openlatch status` to see the running daemon, or `openlatch restart` to restart it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1502 — Daemon did not become healthy

What happened

Daemon process started but health check failed within timeout. Also what `openlatch update` reports when the daemon it restarted onto a new version did not come back, failed its post-restart health check, or was rolled back to the old one.

What to do

Run `openlatch doctor` to see why the daemon is unhealthy, then `openlatch start`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1503 — Newer client version available

What happened

A newer version of openlatch is available (warning, non-blocking).

What to do

Run `openlatch update` to install it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1504 — Update could not be applied

What happened

The update did not complete: the check or download failed, or the downloaded files failed an integrity or start-up check. Before the swap nothing on disk changes. If the swap itself failed, backup copies are left beside the binaries for the rollback.

What to do

Run `openlatch update` again. If it fails a second time, run `openlatch doctor --rescue` and send the file to support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1505 — Built from source, cannot self-update

What happened

This client was built from source with `cargo install`, so the built-in update cannot replace it.

What to do

Rebuild it from source to update it, or reinstall it with one of the commands on the [install page](#) so that the built-in update can manage it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1506 — Daemon did not answer the update

What happened

CLI expected to talk to a running daemon's `/admin/update` endpoint but could not reach it. Distinct from OL-1500 (the **user's** daemon failed to bind a port at startup). Also what `openlatch update` reports when the daemon it is following stops reporting progress: the same stage and byte count for two minutes. The update may still finish; run `openlatch status` to see which version is serving.

What to do

Run `openlatch status` to see which version is running.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1507 — Daemon could not be stopped

What happened

`openlatch stop/restart` could not terminate the daemon: it survived `graceful / shutdown`, `SIGTERM`, AND `SIGKILL` (effectively never — a process that survives `SIGKILL` is stuck in an uninterruptible kernel wait). The user must clean up manually. Distinct from OL-1300 (a `*config*` value error), which this path previously misused.

What to do

End the process with your operating system's task manager, or restart the machine, then run `openlatch start`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1510 — Part of the daemon restarted

What happened

A supervised in-process task panicked. The supervisor caught it via the task's `JoinHandle` and is restarting the subsystem after a backoff; the rest of the daemon keeps serving. Logged at `ERROR` on the first occurrence and at `DEBUG` for each consecutive one, so a subsystem panicking every backoff window cannot flood `daemon.log`.

What to do

Nothing to do if it recovers. If it repeats, run `openlatch doctor --rescue` and send the file to support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1511 — Part of the daemon keeps failing

What happened

A supervised task has failed on every restart for

`[crate::supervision::task::RESTART_LIMIT_WARN_AT]` consecutive attempts. The supervisor keeps retrying — it never gives up — but the streak is surfaced once so a permanently broken subsystem is visible without tailing DEBUG logs.

What to do

Run `openlatch doctor --rescue` and send the diagnostic file to support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1512 — Daemon is degraded

What happened

`/health` reports at least one `RestartPolicy` :: Always subsystem that is not running. The daemon answers requests but is not doing everything it advertises — status becomes degraded rather than ok.

What to do

Run `openlatch doctor` to see which part is down, then `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1513 — Nothing restarts the daemon

What happened

No OS supervisor is installed and nobody declined one, so nothing will restart the daemon if it dies or the machine reboots.

What to do

Run `openlatch system supervision install`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1600 — No credential found

What happened

No credentials found in keychain, env var, or encrypted file.

What to do

Run `openlatch system auth login`, or set `OPENLATCH_API_KEY` and run `openlatch init`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1601 — Credential expired

What happened

API key / token has expired or been revoked.

What to do

Run `openlatch system auth login` to sign in again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1602 — Keychain is unavailable

What happened

OS keychain service is unavailable (e.g., no Secret Service on headless Linux).

What to do

Nothing to do if `openlatch system auth status` shows you signed in. Otherwise run `openlatch system auth login`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1603 — Keychain denied access

What happened

OS keychain denied access (permission error).

What to do

Allow the client in your keychain settings, then run `openlatch system auth login`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1604 — Encrypted credential file failed

What happened

Encrypted file fallback failed (decrypt error, missing file, or corrupt data).

What to do

Run `openlatch system auth login` to store the credential again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1605 — Sign-in timed out

What happened

Auth login flow timed out after 5 minutes waiting for browser callback.

What to do

Run `openlatch system auth login` again and complete the sign-in in the browser that opens.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1606 — Sign-in failed

What happened

Auth login flow failed (callback server error, truncated request, browser launch error).

What to do

Run `openlatch system auth login` again. If no browser can open on this machine, set `OPENLATCH_API_KEY` and run `openlatch init`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1607 — Key revocation failed at sign-out

What happened

Server-side API key revocation failed during logout (warning only — local cleanup continues).

What to do

Revoke the key in the console at `app.openlatch.ai` if it should no longer work.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1608 — Organization must be chosen

What happened

Your account belongs to two or more organizations, nothing named one, and nobody was at a terminal to choose.

What to do

Name the organization with `--org` or `OPENLATCH_ORG`, or run the command again in a terminal and pick it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1609 — Organization picker left unanswered

What happened

The organization picker got no answer: the input ended, timed out or could not be read.

What to do

Run the command again and pick an organization, or name it with `--org` or `OPENLATCH_ORG`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1610 — Organization not found

What happened

The organization named by `--org` or `OPENLATCH_ORG` matches none of your organizations.

What to do

Check the organization's name, then run the command again with the exact name.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1611 — Organization name is ambiguous

What happened

The name given by `--org` or `OPENLATCH_ORG` matches two or more of your organizations.

What to do

Give a name that matches only one organization, or run the command in a terminal without naming one and pick it from the list.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1612 — API key belongs to another organization

What happened

OPENLATCH_API_KEY belongs to an organization other than the one named.

What to do

Use an API key of the named organization, or name the organization the key belongs to.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1613 — Organization could not be confirmed

What happened

An organization was named, and the platform could not be reached to confirm it.

What to do

Run `openlatch system proxy test` to check the route from this machine to the platform, then run the command again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1614 — Machine is enrolled in another organization

What happened

This machine is enrolled in another organization. Only `openlatch system auth switch` moves it.

What to do

Run `openlatch system auth switch` to move this machine to the organization you want.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1615 — Organization switch not applied

What happened

The daemon did not accept the organization switch. Nothing changed.

What to do

Run `openlatch doctor` to check the daemon, then run `openlatch system auth switch` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1616 — Switch refused: the key comes from the environment

What happened

`openlatch system auth switch` refused because the key comes from `OPENLATCH_API_KEY`.

What to do

Set `OPENLATCH_API_KEY` to a key of the organization you want and run `openlatch init`, or unset it and run `openlatch system auth switch`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1617 — Sign-in code expired

What happened

The sign-in code expired or was already used.

What to do

Run `openlatch system auth login` again and complete the sign-in in the browser that opens.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1618 — Account has no organization

What happened

Your account belongs to no organization that is available.

What to do

Ask an administrator of your organization to invite you, then sign in again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1619 — Organization is unavailable

What happened

The chosen organization is suspended or deleted, or its license has ended.

What to do

Choose another organization, or contact your organization's administrator.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1620 — Too many sign-ins in progress

What happened

Too many sign-ins are in progress for this account.

What to do

Wait a few minutes, then run `openlatch system auth login` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1700 — Telemetry consent file is corrupt

What happened

The telemetry.json consent file contains invalid JSON and cannot be parsed.

What to do

Record your choice again with `openlatch system telemetry enable` or `openlatch system telemetry disable`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1701 — Telemetry consent file cannot be written

What happened

Failed to write the telemetry.json consent file (permissions or I/O error).

What to do

Fix the permissions on `~/ .openlatch`, then record your choice again with `openlatch system telemetry disable` or `openlatch system telemetry enable`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1702 — Telemetry batch not sent

What happened

A telemetry batch POST failed (informational – events dropped, no retry).

What to do

Nothing to do. To stop sending usage telemetry, run `openlatch system telemetry disable`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1703 — Telemetry could not start

What happened

Telemetry subsystem failed to initialise (baked key missing, channel error).

What to do

Nothing to do. Run `openlatch system telemetry status` to see the current state.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1704 — Telemetry hand-off refused

What happened

The daemon refused a CLI process's telemetry hand-off (`POST /admin/telemetry/events`): the body was over the byte cap, or not a batch of telemetry events within the event cap. Nothing was relayed and nothing is retried — the events are dropped (l4). Only another process sees this, and the CLI ignores it; there is nothing for an operator to do.

What to do

Nothing to do.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1800 — Doctor repair journal is corrupt

What happened

`doctor --fix` journal file is present but cannot be parsed (`--restore` can't drive).

What to do

Run `openlatch doctor` to check the current state, then `openlatch init` to reinstall cleanly if needed.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1801 — Nothing to restore

What happened

`doctor --restore` was invoked with `no prior --fix` journal on disk.

What to do

Nothing to do. Run `openlatch doctor` to check the current state.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1802 — Diagnostic file cannot be written

What happened

`doctor --rescue` could not write the archive (disk full, permissions, etc.).

What to do

Free disk space or fix the permissions, then run `openlatch doctor --rescue` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1803 — Diagnostic file is incomplete

What happened

`doctor --rescue` produced an archive but one or more collectors errored.

What to do

Send the archive to support as it is; it says which parts are missing.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1900 — Hook integrity key unavailable

What happened

HMAC key unavailable — both OS keyring and file fallback failed.

What to do

Run `openlatch doctor` to check the keychain, then `openlatch init` to reinstall.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1901 — Hook state file is corrupt

What happened

The hook-state.json file exists but cannot be parsed (invalid JSON or unknown schema_version).

What to do

Run `openlatch init` to reinstall the hooks and rewrite the file.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1902 — Hook state file cannot be written

What happened

Atomic write to hook-state.json failed after settings.json was written.

What to do

Fix the permissions on ~/.openlatch, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1903 — Hook marker version not supported

What happened

The `_openlatch.v` field carries a version we don't understand.

What to do

Update the client with `openlatch update`, then run `openlatch init`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1904 — Hook entry could not be processed

What happened

JCS canonicalization rejected the hook entry (internal error; should not happen).

What to do

Run `openlatch doctor --rescue` and send the diagnostic file to support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1905 — Older hook marker upgraded

What happened

A legacy `_openLatch: true` boolean marker was found and will be upgraded (info-level).

What to do

Nothing to do.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1906 — Changed hooks were repaired

What happened

Hook entries were modified outside OpenLatch and the reconciler healed them.

What to do

Find the tool or script that edits the agent's settings file, then run `openlatch system hooks status` to confirm the hooks hold.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1910 — Settings file cannot be replaced

What happened

The client could not write the new version of an agent settings file in place of the old one, because of a permission or disk error. The original file is unchanged.

What to do

Fix the permissions on the agent's settings folder, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1911 — Settings file link cannot be resolved

What happened

An agent settings file is a symbolic link, and the client could not resolve where it points.

What to do

Check that the link's target exists and is readable, then run `openlatch init` again.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1950 — Supervision not supported on this system

What happened

The client cannot install an operating-system supervisor on this system, so nothing restarts the daemon automatically.

What to do

Start the daemon yourself with `openLatch start` after each reboot.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1951 — Supervisor could not be installed

What happened

The client could not install the operating-system supervisor (launchd, systemd or Task Scheduler): writing its files failed, or the supervisor refused the registration.

What to do

Read the operating-system error printed with the code, which names the cause: a folder permission, a `systemctl` failure or a missing user session bus on Linux, or a `schtasks` failure on Windows. Fix it, then run `openlatch system supervision install`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1952 — Supervisor could not be registered

What happened

On macOS, launchd refused to register the client's supervisor, for example because no user session is attached over SSH.

What to do

Run `openlatch system supervision install` from a desktop session on the machine.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-1953 — Supervisor refused a command

What happened

A start / stop / restart handed to the OS supervisor failed – the unit is registered but `systemctl` / `launchctl` / `schtasks` refused the control verb. Callers fall back to acting on the process directly, which is the pre-supervision behaviour, so this is a degraded path and not a hard failure.

What to do

Run `openlatch system supervision status` to check the supervisor.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2000 — Configuration watcher could not start

What happened

Config monitor init failed (manifest parse error or watcher setup error).

What to do

Run `openlatch system inventory status` to see the cause, then `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2001 — Configuration source list not found

What happened

Manifest file not found at configured path.

What to do

Run `openlatch init` to restore it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2002 — Configuration source list is invalid

What happened

Manifest schema validation failed.

What to do

Run `openlatch update`, then `openlatch init` to restore a valid list.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2003 — Watched path cannot be expanded

What happened

Path expansion failed (unknown variable, missing env var, OS-call failure).

What to do

Search `daemon.log`, in the `logs` folder of the client's state directory, for OL-2003: each entry names the path that could not be expanded. Fix the variable it uses, then run `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2004 — One path cannot be watched

What happened

Watcher init failed for a specific path (continues running for other paths).

What to do

Search `daemon.log`, in the `logs` folder of the client's state directory, for OL-2004: each entry names the path that cannot be watched. Check that path's permissions.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2005 — Configuration file could not be fingerprinted

What happened

JCS canonicalization or NFC normalization failed (fail-open: log + continue).

What to do

Nothing to do. If it repeats for the same file, run `openlatch system inventory inspect` on that source.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2006 — Configuration rescan was partial

What happened

Periodic rescan partial — some paths inaccessible; rescan completes for the rest.

What to do

Search `daemon.log`, in the `logs` folder of the client's state directory, for OL-2006: each entry names a path the rescan could not read. Check those paths' permissions.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2007 — Session start details unavailable

What happened

Session-start enrichment failed (logs warning, continues with un-enriched event).

What to do

Nothing to do. If it repeats, run `openlatch system inventory status`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-2008 — Configuration watching is off

What happened

The configuration-plane monitor is switched off in config, so config drift on this host is neither observed nor reported.

What to do

Unset `OPENLATCH_INVENTORY_ENABLED` (or set it to `true`), and set `enabled = true` under `[inventory_monitor]` in `config.toml` (or remove that line), then run `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-9999 — Unexpected internal error

What happened

Code assigned to all internal/unexpected errors routed through `bug_report()`.

What to do

Run `openlatch doctor --rescue` and send the diagnostic file to support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-CA — Model relay certificate not trusted

What happened

The relay's CA is not trusted by the user's own OS trust store — the install was skipped (no desktop session), refused (dialog cancelled), or the chain/store proof failed — so no proxy value was written and the agent goes direct (not captured). Remedy: `openlatch doctor --fix` from a desktop session, or the platform install command. Rendered under `PreflightFailed`.

What to do

Run `openlatch doctor --fix` from a desktop session on the machine.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-CURSOR-UNPROVEN — Cursor relay not yet proven

What happened

Cursor's relay is wired and no decrypted Cursor request has reached it since (doctor's `agents.cursor.relay.traffic = "unproven"`). One proof for the whole agent: the relay cannot tell the CLI's requests from the IDE's. Remedy: start one Cursor agent turn.

What to do

Start one Cursor agent turn, then run `openlatch doctor`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-CURSOR-UNSUPPORTED — Cursor surface not covered by the relay

What happened

A Cursor surface this build cannot point at the relay (`agents.cursor.relay.{cli,ide} = "unsupported"`): the IDE until its proof passes. Its model calls go direct and are not captured; the command plane still enforces. Not in `[ERR_MODEL_RELAY_CODES]` either.

What to do

Keep the client current with `open\latch update`; the surface is covered once a release adds it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-DOWN — Model relay is not running

What happened

Nothing is answering the model relay status endpoint on the pinned port (connection refused).

What to do

Run `openlatch restart`, then `openlatch system model-relay status`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-FOREIGN — Another process holds the relay port

What happened

The pinned model relay port is held by a process that is NOT the OpenLatch model relay (wrong/missing status signature). Reported by `openlatch doctor`: an agent wired at a foreign loopback listener is sending its provider credential to that process.

What to do

Stop the process that holds the port, then run `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-IO — Model relay input or output error

What happened

Generic model relay I/O error (used by the `From<std::io::Error>` bridge).

What to do

Run `openlatch system model-relay status`. If the error repeats, run `openlatch doctor --rescue` and send the file to support.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-MISCONFIG — Relay setting is overridden

What happened

The editor provably loaded a provider slot's relay URL and is using that provider, yet no traffic reaches the endpoint — something above the file (an organisation's remote configuration) overrides it.

What to do

Check the editor's remote or managed configuration for that provider's address, then run `openlatch doctor`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-NEXTSTART — Relay applies at the editor's next start

What happened

A provider slot is wired to its relay endpoint, and the editor that reads it has not been proven to use it yet.

What to do

Nothing to do. The check clears with the first model call through the relay after the editor restarts.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-NOFIND — No local record for this id

What happened

`model-relay explain <finding_id>` found no matching churn block in the bounded local retention store (never produced here, or aged/pruned out).

What to do

Check the id, then run the command on the machine that produced the record.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-PORT — Model relay port is in use

What happened

The pinned model relay loopback port is already in use. Loud, never a silent re-probe (D-25) — the agent config points at this exact port.

What to do

Free the port, then start the daemon again with `openlatch start`. To run without the model relay, turn it off with `openlatch system model-relay disable`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-PORTS — No relay port for a provider

What happened

A provider slot could not get an endpoint port: its port in the endpoint block (`[port + 1, port + 32]` above the main relay port) is held by another process, or every port in the block is already assigned.

What to do

Free a port just above the model relay port, then run `openlatch restart`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-PREFLIGHT — Model relay could not reach the provider

What happened

The model relay bound its port but a synthetic request could not complete the round trip to the provider through it, so the agent was deliberately left unwired (`model_relay::preflight`). Reported by `init` — which exits non-zero while leaving hooks, daemon and supervision fully installed — and by `doctor`. NOT a daemon-fatal condition: the daemon keeps serving and re-wires itself as soon as the probe passes.

What to do

Run `openlatch system proxy test` to check the route to the provider, then `openlatch doctor`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-PROVIDER — Agent config holds another provider entry

What happened

A provider table under the name OpenLatch writes already exists in the agent's config and does **not** point at our loopback listener — it is somebody else's. Install refuses rather than overwriting it, because uninstall could not then round-trip the file: the reversal would delete a table that was never ours and restore a pointer we never recorded.

What to do

Rename or remove the conflicting `[model_providers.<name>]` table, then run `openlatch init`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-SERVE — Model relay stopped on an error

What happened

The model relay axum server exited with an I/O error while serving.

What to do

Run `openlatch restart`, then `openlatch system model-relay status`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-SETTINGS — Agent settings hold your own proxy

What happened

An agent's settings file holds a proxy OpenLatch did not write in the key it would use, or cannot be parsed as JSONC. It is left untouched and that surface is not captured. Remedy: move that proxy into OpenLatch's [proxy] config, which the relay's egress already chains through.

What to do

Move that proxy into the client's configuration with `openlatch system proxy set`, then run `openlatch init`.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-STATEFILE — Agent state file cannot be edited safely

What happened

An agent state file OpenLatch edits to wire a provider slot could not be edited safely: it is larger than the bound it is read under, or it is not valid JSON.

What to do

Run `openlatch doctor` to see which file it is, and fix or shrink it.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-TRUSTPROOF — Relay trust not yet proven here

What happened

The user trust store verified, on a platform whose trust path has no real-runner proof yet (Windows, Linux). State :: Degraded with Resolution :: NothingToDo: a coverage limit (D-30).

What to do

Nothing to do.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

OL-RELAY-UNCOVERED — Provider not carried by the relay

What happened

A configured provider cannot be carried by the relay, for a named reason: its requests are signed over the host, this build does not know where it would otherwise send them, or its client exposes no setting to point.

What to do

Nothing to do. Run `openlatch doctor` to see which providers are not carried.

Note

This page is generated from the client source at a pinned commit. The message above is what the client prints.

Client 0.6.18

October 13, 2026 · Client

Choose your organization at sign-in, and move a machine to another organization.

- Signing in asks which organization to use when your account belongs to several; `--org` or `OPENLATCH_ORG` names it without asking.
- New command `openlatch system auth switch` moves this machine to another of your organizations.
- On Windows, `openlatch doctor` reports an agent's command-line tool whose machine-wide install bypasses the relay (OL-1425).
- Groundwork for a coding agent that OpenLatch does not yet support. Nothing changes for the agents already wired.

[@openlatch/client@0.6.18](#)

Client 0.6.17

October 12, 2026 · Client

Cursor returns to coverage once its windows reload.

- A held Cursor is covered again once its windows reload.
- The relay records each Cursor call once, and classifies the traffic of a coding agent that OpenLatch does not yet support.

[@openlatch/client@0.6.17](#)

Client 0.6.16

October 12, 2026 · Client

The client also connects the Codex desktop app, with no command-line tool and no new `openlatch init`.

- The client connects the Codex desktop app on a machine with no Codex command-line tool, without running `openlatch init` again.
- `openlatch doctor` no longer reports a false finding for a coding agent that OpenLatch does not yet support.

[@openlatch/client@0.6.16](#)

One policy in several Autonomy Zones

October 12, 2026 · Platform

An Agent Intent Policy can apply in several zones at once, each attachment with its own mode. Before you attach, detach or change a mode, the console previews the impact. An attachment that would loosen what a zone above already blocks is refused.

Client 0.6.15

October 11, 2026 · Client

Cursor and Cline command-line tools reach the relay on Windows with no setup.

- On Windows, the Cursor and Cline command-line tools route their model traffic through the relay with no setup.
- On Windows, the relay's certificate check no longer reads a file that is still open.

[@openlatch/client@0.6.15](#)

Client 0.6.14

October 11, 2026 · Client

Groundwork for a coding agent that OpenLatch does not yet support.

- Relay groundwork for a coding agent that OpenLatch does not yet support. Nothing changes for the agents already wired.

[@openlatch/client@0.6.14](#)

Client 0.6.13

October 10, 2026 · Client

Groundwork for a coding agent that OpenLatch does not yet support, and a Windows relay fix.

- Hook and inventory groundwork for a coding agent that OpenLatch does not yet support. Nothing changes for the agents already wired.
- On Windows, the relay's certificate is checked in the user's own context, and background processes start without opening a console window.

[@openlatch/client@0.6.13](#)

Client 0.6.12

October 10, 2026 · Client

More accurate model usage and PowerShell reading.

- `openlatch doctor --fix` reports the state it leaves behind.
- Usage that an interrupted model stream already reported is kept.
- A model is marked unknown only when its usage is estimated.
- An API key is told apart from a subscription token by its shape.
- A PowerShell parameter written as `-Param: value` is read the same way as `-Param value`.

[@openlatch/client@0.6.12](#)

Client 0.6.11

October 9, 2026 · Client

PowerShell commands are read more accurately.

- A URL in a PowerShell command is now read as a URL, not as a file path.

[@openlatch/client@0.6.11](#)

Client 0.6.10

October 8, 2026 · Client

Fixes for enrolment and for reading PowerShell commands.

- The daemon's idle health check now sends the API key.
- `openlatch init` now saves an exported `OPENLATCH_API_URL` to the client configuration.
- A file path is classified where it resolves, not as it is spelled.
- PowerShell commands are split into words by PowerShell's own rules.

[@openlatch/client@0.6.10](#)

Autonomy Zones on your organization tree

October 8, 2026 · Platform

Any unit of your organization can carry an Autonomy Zone; a unit without one inherits the nearest zone above it. Each agent card shows its zone and the layers it inherits from. You can assign agents to a zone by hand after previewing the change, and set a default zone for agents whose owner is not resolved yet.

Autonomy Zones and Agent Intent Policies

October 8, 2026 · Platform

Each agent acts within an Autonomy Zone built from Agent Intent Policies across five dimensions: Quality, Economics, Security, Safety and Compliance. You approve each policy as a generated sentence, with its generated test cases shown beside it. Verdicts are Allow, Ask, Block or Optimize.

Public REST API

October 8, 2026 · Platform

The console's operations are available through a versioned public REST API under `/api/public/v1`, authenticated with organization API keys. It supports idempotency keys and rate limits, and every error comes from a closed catalogue.

Client 0.6.9

October 7, 2026 · Client

Live configuration reports, and fixes for Cline, Cursor and macOS.

- Configuration changes on a host are reported as they happen.
- A Claude Code teammate session is linked to the session that leads it.
- The skills, rules and MCP servers that Codex CLI, Cursor and Cline load are inventoried as each agent actually loads them.
- Cursor reports name the MCP server on every call.
- Updating Cline.app no longer interrupts a conversation.
- A Claude Code session started from a plan records that plan as its first prompt.
- On macOS, the client no longer asks for access to the Desktop, Downloads and Documents folders.
- `openlatch init` no longer describes usage telemetry as anonymous.
- Native Windows paths are classified correctly on file reads and writes.

[@openlatch/client@0.6.9](#)

Database-enforced tenant isolation

October 6, 2026 · Platform

Row-level security is enabled and forced on every tenant table, with no runtime owner or bypass role, and it is checked in CI and at every start. It is an engineering control, not an attestation.

Client 0.6.8

October 6, 2026 · Client

Policy bundle and relay fixes.

- A cached policy bundle is revalidated correctly when the platform reports it unchanged.
- Usage from sessions that no hook registered is no longer recorded.

[@openlatch/client@0.6.8](#)

Client 0.6.7

October 5, 2026 · Client

Path-aware evaluation and Cline fixes.

- An action's effects are correlated with the paths it targets.
- The Cline plugin applies to every tool, not only the shell.
- `openlatch init` waits for the first Cline plugin delivery instead of failing.
- Each action is decided once, not again after it runs.
- Codex CLI side threads are filed under their parent session.
- Usage is filed under the right installation.

[@openlatch/client@0.6.7](#)

Policies that read what an action does

October 5, 2026 · Platform

Each action is classified by its effect and its target, whichever tool of a wired agent carries it: a read of secret material, a call to a network host, a write to a path class. A policy written once applies to the same effect across agents.

Client 0.6.6

October 2, 2026 · Client

The Cursor IDE's model calls go through the relay.

- The Cursor IDE's model calls are routed through the relay.
- A fresh `openlatch init` no longer fails on components that are still starting.
- `openlatch doctor` shows one status line for every agent and recognizes the Cline.app hub.
- A shell process substitution is no longer read as a redirection target.

[@openlatch/client@0.6.6](#)

Client 0.6.5

October 1, 2026 · Client

One shared tool vocabulary, and Ask verdicts delivered per agent.

- Each agent's tools map onto one shared vocabulary; a tool with no mapping is still recorded under its own name.
- Policy evaluation reads one classification catalogue from the policy bundle.
- Ask verdicts reach each agent in the form it supports, with an authored verdict where an agent has no ask of its own.

[@openlatch/client@0.6.5](#)

Client 0.6.4

October 1, 2026 · Client

Cursor support.

- Policies decide on Cursor actions, and the verdict reaches Cursor in its own format.
- Cursor's model traffic goes through the relay and is recorded as usage, never as content.
- Cursor's wiring is checked, repaired and inventoried like every other agent's.
- `openlatch uninstall` names the agents a partial uninstall leaves wired.

[@openlatch/client@0.6.4](#)

Client 0.6.3

September 30, 2026 · Client

The Cline plugin reaches Cline.app and VS Code.

- The Cline plugin is delivered to Cline.app and to VS Code.
- The client checks that the plugin loaded on each host.

[@openlatch/client@0.6.3](#)

License and usage in the console

September 29, 2026 · Platform

The licensing page in Settings shows what your organization is licensed for against what it consumes, the contract dates, the daily history and the list of hosts, which you can export as a deployment report.

Agent ownership

September 29, 2026 · Platform

Each agent resolves to an owner: a named person, a declared service owner, or Unattributed with its reason. Overriding an owner is recorded in the audit trail, and an agent still running under a departed employee is flagged.

Agents you can recognize

September 29, 2026 · Platform

Each agent card shows a name a person recognizes and how many sessions across how many projects it brings together. Filter agents, inventory and events by repository, and see when one machine has become several agent rows.

Execution graph

September 29, 2026 · Platform

The Execution Graph tab draws an agent's model calls, tools, commands, files and network calls over a time window. Filter it to the costliest or riskiest paths, open any node to see the events behind it, and start a rule in Monitor mode from a command node, replayed against the window first.

Agent economics

September 29, 2026 · Platform

Each agent's Economics tab shows spend, a forecast and a session breakdown over the period you choose. Savings opportunities are ranked by their estimated impact; they are a diagnosis and change nothing on their own.

Client 0.6.2

September 29, 2026 · Client

Cline reporting, self-repair and a complete purge.

- Cline sessions, prompts and tool calls are reported to the platform.
- Relay wiring, hook trust and the Cline plugin files are checked for tampering and repaired.
- The client reports when a policy bundle is waiting to be delivered.
- `openlatch uninstall` sends an uninstall event, and `openlatch uninstall --purge` removes all client state.

[@openlatch/client@0.6.2](#)

Client 0.6.1

September 29, 2026 · Client

Cline through the relay, and relay certificate management.

- Cline's model traffic is captured through the relay, including the Cline desktop app.
- The relay proves its certificate is trusted before an agent is pointed at it.
- The relay's certificate can be reported, rotated, released and removed.
- Only detected agents are reported in host facts.
- Codex CLI hooks are trusted on Windows.

[@openlatch/client@0.6.1](#)

Policies that keep deciding offline

September 29, 2026 · Platform

Policy changes reach every client on its next poll, with no client release and no action on each host. Each client keeps the last bundle it received and keeps deciding with it while the network is unavailable. In Monitor mode, what a policy would have blocked is recorded with the event.

Client 0.6.0

September 24, 2026 · Client

A step-by-step `openlatch update`, with one breaking change to its JSON output.

- Breaking: `openlatch update --check --json` no longer returns `min_supported_client`.
`current`, `latest` and `severity` are unchanged.
- `openlatch update` shows its progress step by step, and the daemon comes back after it.
- Cline hooks, rules and workflows are reported by the configuration inventory.

[@openlatch/client@0.6.0](#)

Client 0.5.8

September 23, 2026 · Client

Cline prompt capture.

- Prompts are captured on the Cline CLI and the Cline desktop app.
- Agent types that are not wired are no longer reported.

[@openlatch/client@0.5.8](#)

Client 0.5.7

September 23, 2026 · Client

Terminal output fix on Windows.

- Colors and progress display correctly on Windows for every command, not only `openlatch init`.

[@openlatch/client@0.5.7](#)

Client 0.5.6

September 23, 2026 · Client

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.

[@openlatch/client@0.5.6](#)

Client 0.5.4

September 22, 2026 · Client

Daemon supervision and telemetry delivery fixes.

- On Windows, the scheduled task that keeps the daemon running keeps its version marker.
- The daemon's first telemetry batch is sent over HTTPS.

[@openlatch/client@0.5.4](#)

Client 0.5.3

September 22, 2026 · Client

Telemetry through the platform, and Linux binaries for older hosts.

- Telemetry and crash reports go through the platform's own ingestion endpoint.
- Linux binaries start on slim and older hosts.

[@openlatch/client@0.5.3](#)

Client 0.5.2

September 21, 2026 · Client

Startup and event delivery fixes.

- On Windows, `openlatch` runs as a console program, and its binaries start on a fresh host.
- Events still queued at shutdown are sent.
- The daemon notices a new login without a restart.
- Event delivery recovers after every reconnection to the platform.

[@openlatch/client@0.5.2](#)

Client 0.5.1

September 20, 2026 · Client

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.

[@openlatch/client@0.5.1](#)

Client 0.4.0

September 8, 2026 · Client

Subsystem commands move under `openlatch system`.

- Breaking: `openlatch auth`, `hooks`, `model-relay`, `proxy`, `supervision`, `telemetry` and `inventory` are now `openlatch system <name>`, with no aliases.
- Breaking: the `[boundary]` configuration section is now `[model_relay]`, `OPENLATCH_BOUNDARY_*` variables are now `OPENLATCH_MODEL_RELAY_*`, and `OL-BND-*` error codes are now `OL-RELAY-*`.
- `openlatch doctor` reports the live state of each hook.
- Proxy diagnostics report the proxy the client actually uses.

[@openlatch/client@0.4.0](#)

Client 0.3.3

September 6, 2026 · Client

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.

[@openlatch/client@0.3.3](#)

Client 0.2.2

August 28, 2026 · Client

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.

[@openlatch/client@0.2.2](#)

Client 0.1.18

August 19, 2026 · Client

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.

[@openlatch/client@0.1.18](#)

Client 0.1.14

April 23, 2026 · Client

Default API address fix.

- The default API address no longer repeats /api.

[@openlatch/client@0.1.14](#)

Client 0.1.13

April 23, 2026 · Client

The npm package resolves with `npx @openlatch/client init`.

- The npm package gains a `client` entry point, so `npx @openlatch/client init` resolves.
- Help output no longer prints a banner.

[@openlatch/client@0.1.13](#)

Client 0.1.12

April 23, 2026 · Client

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.

[@openlatch/client@0.1.12](#)

Client 0.1.8

April 23, 2026 · Client

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.

[@openlatch/client@0.1.8](#)

Client 0.1.7

April 23, 2026 · Client

The npm package ships the hook entry point.

- The npm package installs the `openlatch-hook` entry point.

[@openlatch/client@0.1.7](#)

Client 0.1.6

April 23, 2026 · Client

Release maintenance.

- Changes to how the client is built and released. The client behaves as in the previous version.

[@openlatch/client@0.1.6](#)