Dweve
Weaving Intelligence

ENGLISH EDITION · JULY 2026

MARKETING BROCHURE · 16 PAGES

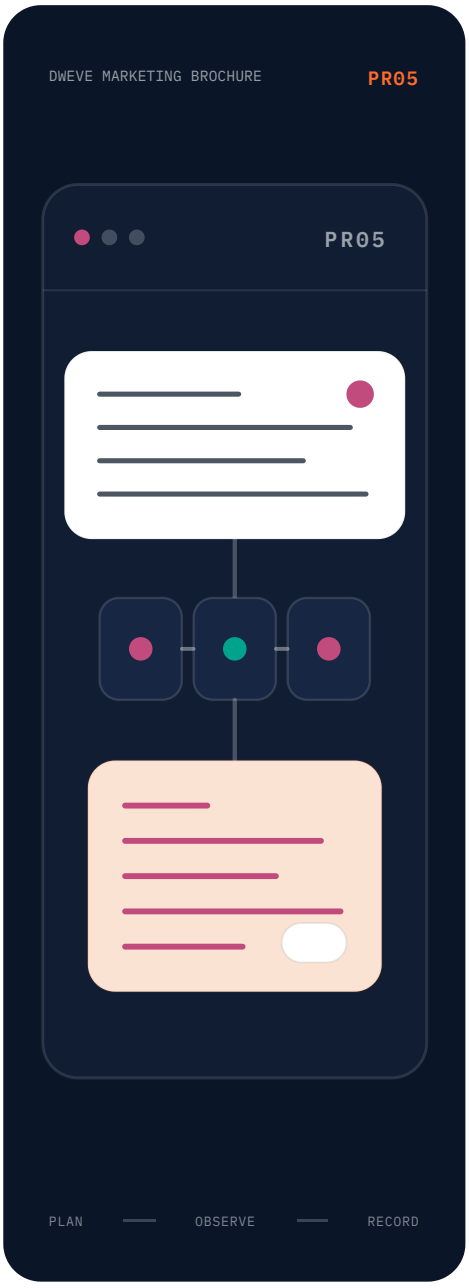
Aura: the product brochure

A standalone visual introduction to Aura, the job it owns, how it works, where it fits, and how to prove it.

VISUAL BRIEFING

PRODUCT LEADERS, BUYERS, ARCHITECTS, DELIVERY TEAMS, AND TECHNICAL EVALUATORS

ENGLISH



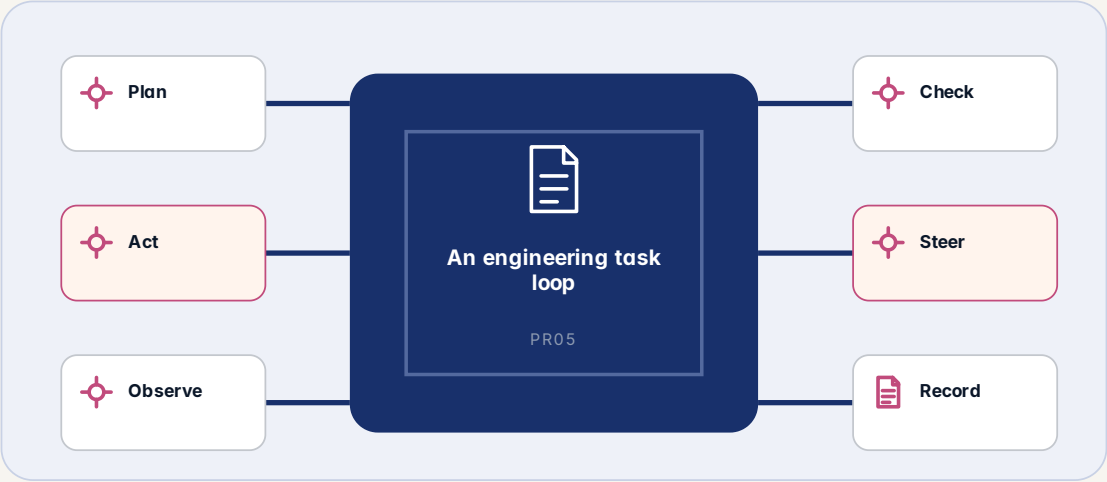
Decide whether Aura belongs in the selected workflow.

PR05

THE PROPOSITION

The engineering loop around the model

Aura is the closed-loop development runtime around the models your team chooses. It plans, executes bounded actions, observes the real repository, accepts human steering, runs quality checks, judges the result against explicit criteria, replans when required, and keeps the operational record. The model can change. The engineering loop remains Aura.



How to read this visual: follow an engineering task loop through the proposition.

DECISION

Read the promise against the operating reality

Aura is the closed-loop development runtime around the models your team chooses. It plans, executes bounded actions, observes the real repository, accepts human steering, runs quality checks, judges the result against explicit criteria, replans when required, and keeps the operational record. The model can change. The engineering loop remains Aura.

A task enters with an objective and success criteria. Aura retrieves the project state, creates a plan, executes one bounded action, observes what actually happened, accepts policy and human feedback, checks the result, and either finishes, stops honestly, or replans.

- Plan
- Act
- Observe
- Check

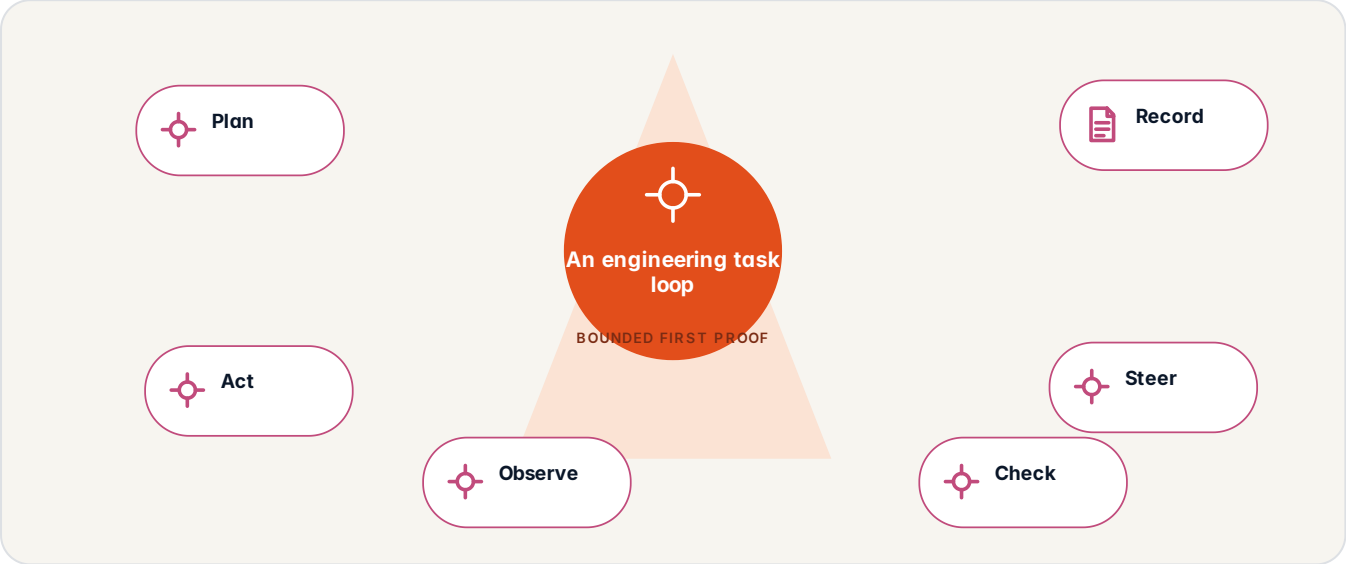
QUESTION TO CARRY

Which part of this proposition changes the outcome people receive today?

USE AND FIT

Where Aura can earn its place

Start with responsibilities that have a real owner, useful sources, and a consequence worth improving.



How to read this visual: follow an engineering task loop through use and fit.

MECHANISM

Separate useful scope from attractive distraction

Aura can run the project formatting, linting, compilation, tests, documentation checks, and audit stages. A failure goes back into the task. Aura can inspect it and correct the work instead of leaving you with a plausible patch and a broken build.

Aura supports sequential chains, parallel runs, coordinated teams, and background tasks. Shared task boards, messages, quality gates, worktrees, and file-level claims coordinate the work. Aggregation remains explicit rather than hiding several contributors behind one anonymous answer.

- Plan
- Act
- Observe
- Check

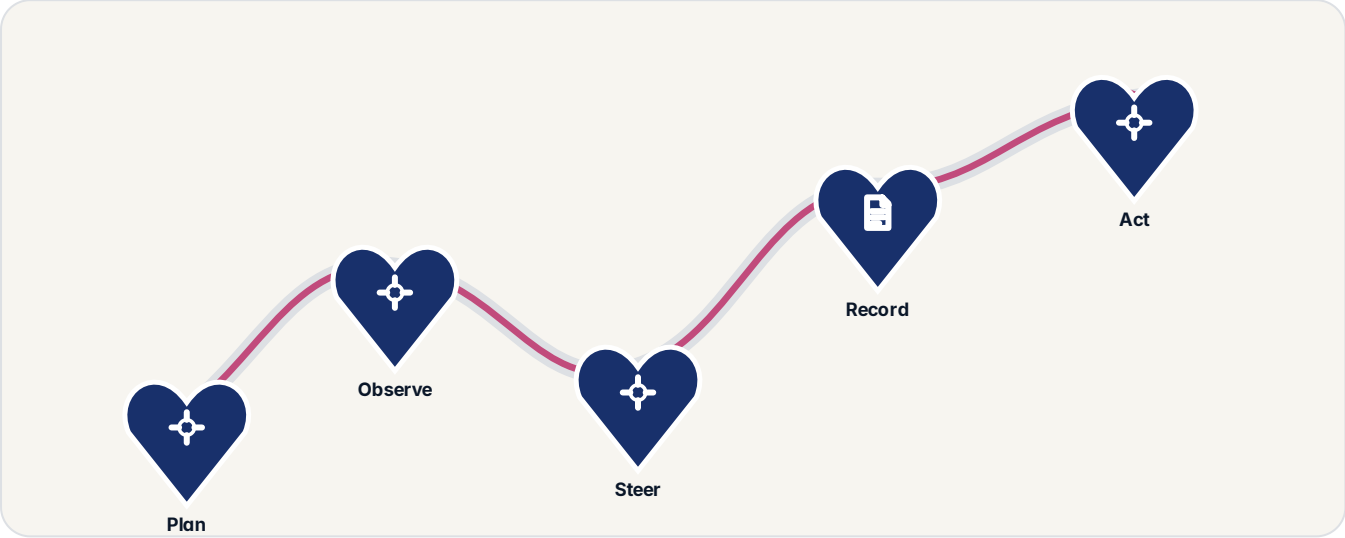
QUESTION TO CARRY

Where does this earn a place, and where should it deliberately not be used?

THE PATH

How Aura moves from question to decision

A focused engagement should make the next decision easier, not create a larger promise.



How to read this visual: follow an engineering task loop through the path.

WHY THIS SHAPE WORKS

The workflow, operating boundary, evidence, and decision criteria are considered together from the start.

DECISION

Make every stage earn the next one

The system does not assume the first plan remains correct after it touches the code. Tool output changes the state. Tests reveal new information. Aura treats each of those events as feedback into the next decision.

You do not have to wait for a long run to finish, throw away the result, and start a new conversation. The correction enters the next decision while the context is still active.

- Plan
- Observe
- Steer
- Record

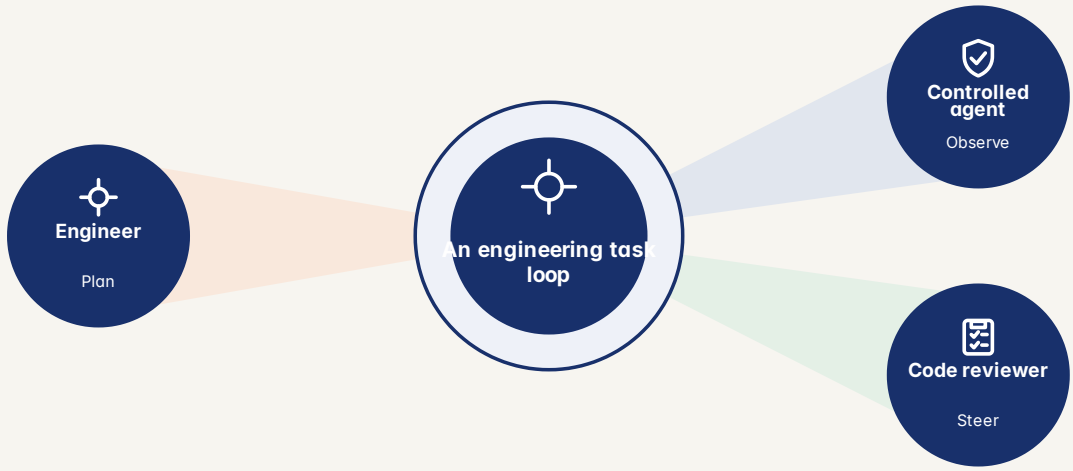
QUESTION TO CARRY

What evidence must exist before the scope is allowed to grow?

THE VALUE PICTURE

Aura seen from three responsibilities

Good AI work must make sense to the person using it, the team operating it, and the people reviewing it.



How to read this visual: follow an engineering task loop through the value picture.

PRIMARY READER

Product leaders, buyers, architects, delivery teams, and technical evaluators

The first conversation.

DECISION SUPPORTED

Decide whether Aura belongs in the selected workflow.

The next useful step.

MECHANISM

Read the same outcome from several responsibilities

Generation speed can move work into review rather than remove it. Developers carry context between the assistant, shell, test runner, Git, security checks, issue tracker, and reviewer. Aura makes that state explicit and reusable instead of leaving it in one person as attention.

Teams spend hours on work that does not need human creativity: finding bugs, writing boilerplate, updating dependencies, refactoring legacy code, onboarding people. Existing AI tools help but add their own problems, code shipped to the cloud, silent changes, no record. They rarely work across files or coordinate between people. Aura folds the assistant, the search, the memory, the security checks and the audit log into one program you already run.

- Plan
- Observe
- Steer
- Act

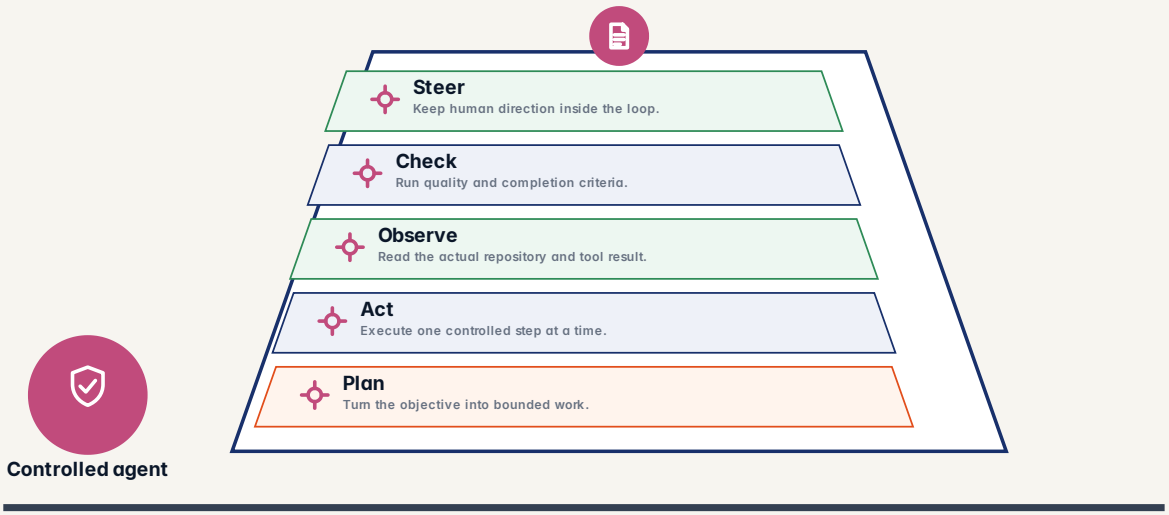
QUESTION TO CARRY

Can the same result satisfy its user, operator, and reviewer?

THE OPERATING MODEL

The control model around Aura

A credible proposition connects the visible result to the less visible responsibilities underneath it.



How to read this visual: follow an engineering task loop through the operating model.

DECISION

Name the controls behind the simple interface

Aura supports declarative criteria for substring, regular expression, file existence, and test pass. The deterministic path runs first. The optional model judge handles criteria that need interpretation. The result is explicit: satisfied, unsatisfied, or inconclusive under the selected policy.

Aura can truncate oversized output, strip control characters and terminal noise, detect prompt-injection patterns, fence web-origin content, and redact supported PII and secret formats before the result re-enters model context.

- Plan
- Act
- Observe
- Check

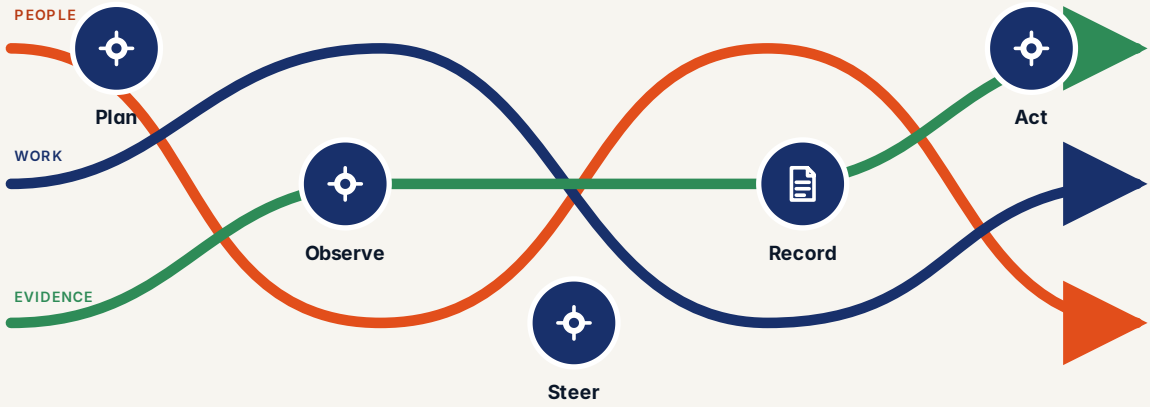
QUESTION TO CARRY

Who owns each control when the interface appears simple?

THE RESPONSIBILITY FLOW

People, Aura, and evidence move together

The work becomes easier to govern when each stage leaves a clear hand-off.



How to read this visual: follow an engineering task loop through the responsibility flow.

MECHANISM

Keep every hand-off attached to an owner

Aura is provider-agnostic. The backend layer in src/llm/backends ships native API clients for Anthropic Claude, Mistral and Codestral, the OpenAI-compatible API (Groq, Together, DeepSeek, Fireworks and more) and the Anthropic-compatible API (GLM, Kimi). CLI advisory backends additionally wrap Aider, Claude Code, the Gemini CLI and the Kimi CLI, so you can pull a second opinion from another agent without leaving the session. Gemini is reached through its CLI, not a native API backend.

Aura runs on infrastructure you control, strips personal data before anything leaves, gates every tool behind a permission, and records every action in a tamper-evident trail. None of this is a certification claim. These are real controls that give your GDPR, EU AI Act, NIS2 and DORA reviews concrete, exportable evidence instead of promises. Defaults are conservative; you turn the knobs up, never quietly down.

- Plan
- Observe
- Steer
- Record

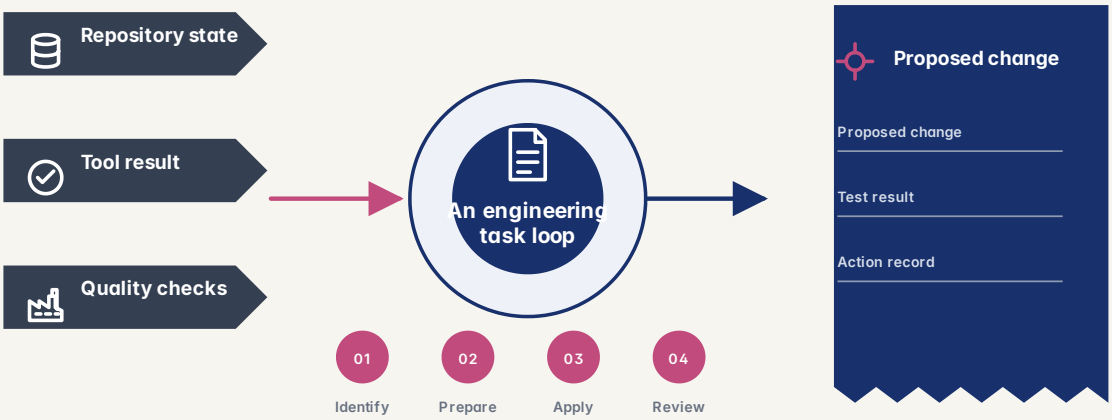
QUESTION TO CARRY

Is every hand-off visible, bounded, and assigned to an owner?

SOURCE TO RESULT

Aura: from authoritative source to result

The work stays explainable when source identity, transformation, decision, and hand-off remain visible between systems.



How to read this visual: follow an engineering task loop through source to result.

DECISION

Keep the basis visible as the work changes form

Tell Aura what you want in plain language. It reads the relevant project files, makes a plan, proposes or performs the actions you allow, runs the checks you choose, and corrects the work when the result does not meet the goal. You can steer it, stop it, and undo supported changes at any time.

Aura runs as a native program on developer machines, controlled servers, containers, CI runners, and isolated infrastructure. Supported local models can keep inference inside the environment. Cloud backends remain optional and visible. The team chooses which content may leave and which tools may reach the network.

- Identify
- Prepare
- Apply
- Review

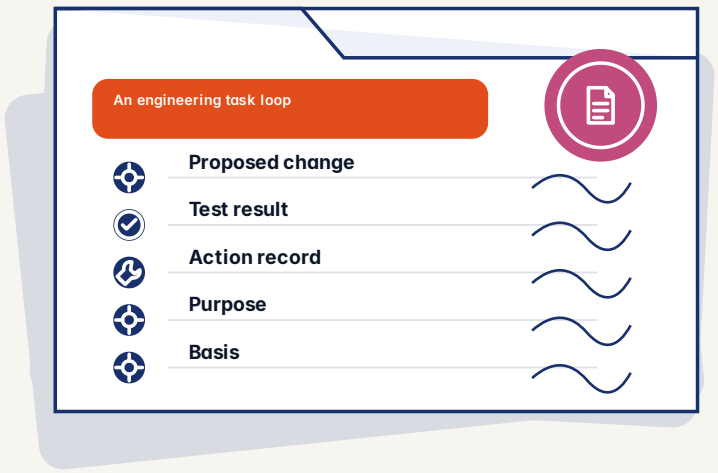
QUESTION TO CARRY

Can the result still be connected to the exact basis that shaped it?

EVIDENCE PACKET

The evidence Aura leaves with the work

Capture enough to explain the result, inspect authority, investigate failure, and make the next decision without rebuilding the story from memory.



How to read this visual: follow an engineering task loop through evidence packet.

MECHANISM

Preserve the record needed by the next question

Aura listens between actions. When a result reveals the wrong assumption, redirect the next step without starting a new session and re-explaining the project.

Aura sits in your terminal and helps you write better code faster. Ask it to fix a bug, explain a function, or write a test. It shows you exactly what it plans to do, asks before making changes, and keeps a record of everything. Your code stays on your machine. No cloud required.

- Purpose
- Basis
- Authority
- Action

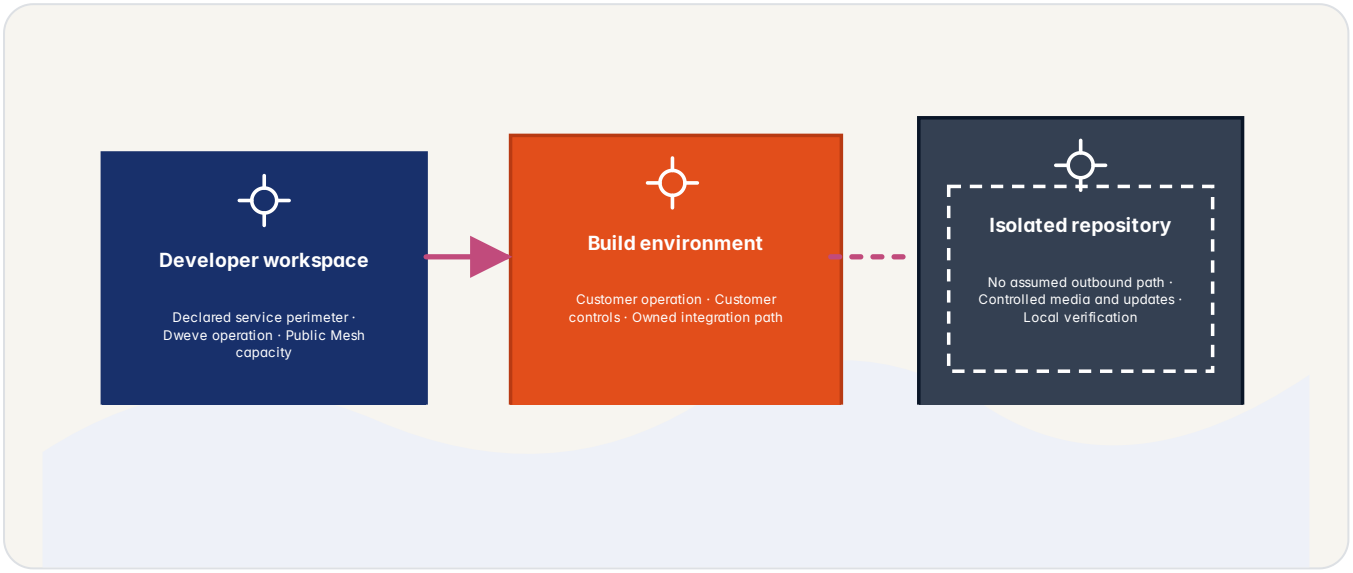
QUESTION TO CARRY

Does the retained record answer the next difficult question without reconstruction?

DEPLOYMENT CHOICE

Choose the operating posture for Aura

Location is only one dimension. Decide who operates, who holds keys, which dependencies exist, how updates arrive, and what continuity requires.



How to read this visual: follow an engineering task loop through deployment choice.

DECISION

Compare operating responsibility, not location alone

Aura works with several set-up helpers and compatible services. A helper on your computer can keep the thinking with you. A helper in the cloud only receives what you choose to send. You can let Aura pick among the options you set up, with a fallback for when one is unavailable.

Aura's security architecture operates at three independent layers, the tool layer (permission system, bash AST analysis), the content layer (PII detection, prompt-injection detection, content sanitisation), and the infrastructure layer (filesystem scope, session allowlist, network controls). Each is configurable per project, per session, per env.

- Begin through managed Fabric
- Run on customer infrastructure
- Close the environment
- Plan

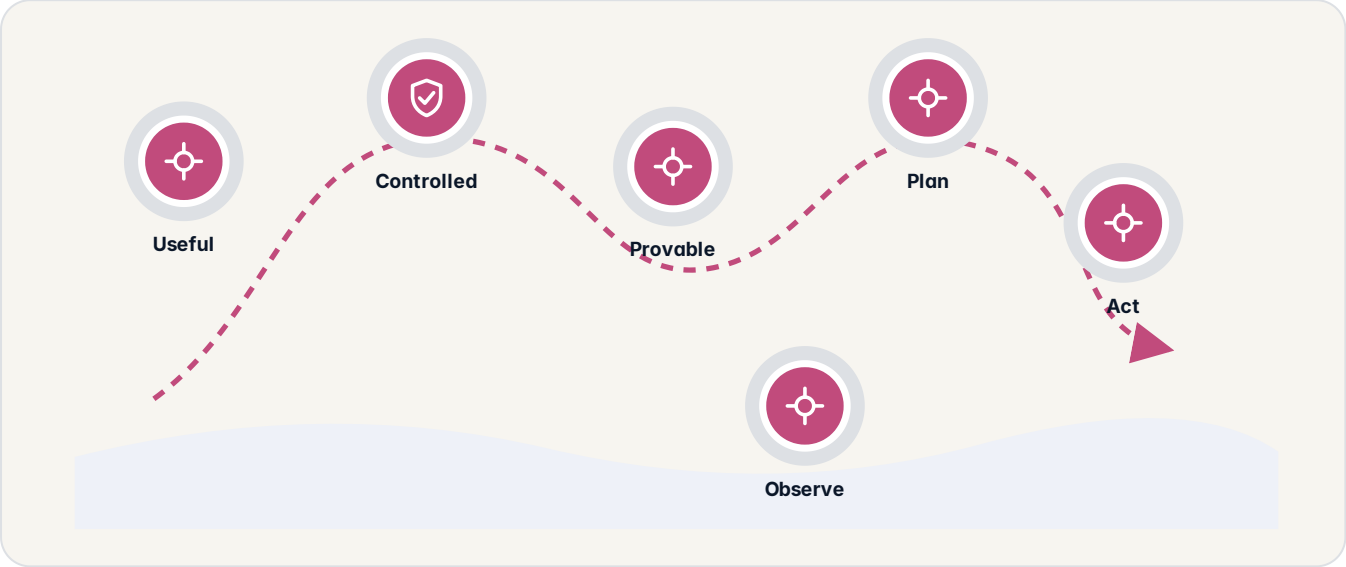
QUESTION TO CARRY

Who holds infrastructure, keys, updates, logs, recovery, and the exit path?

WHAT TO LOOK FOR

Signals that Aura is earning trust

Progress is a useful outcome under understood control, supported by reviewable evidence.



How to read this visual: follow an engineering task loop through what to look for.

MECHANISM

Use signals that can change the next decision

Aura can keep the task open until a supported condition is satisfied. You can require a test result, a file, a matching output, or another declared condition. It stops when the goal is met, when you stop it, or when it cannot continue honestly.

Aura is an AI development agent that works in your terminal, understands your codebase, and writes code with full transparency. Every change is explained, every tool call is logged, and every session leaves an audit trail. 25+ built-in tools, multi-agent orchestration, and security guardrails that keep you in control. Built in Rust, runs anywhere.

- Useful
- Controlled
- Provable
- Plan

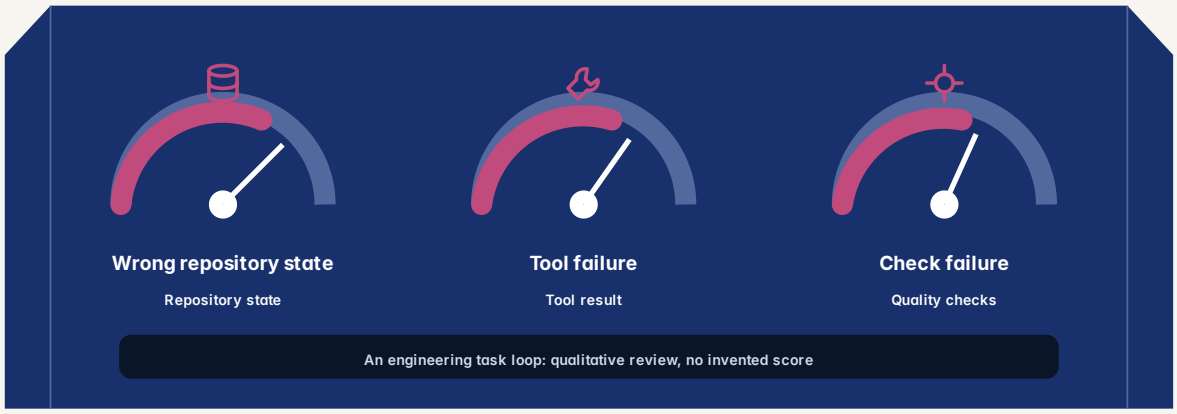
QUESTION TO CARRY

Which signals would change the next decision rather than merely impress the room?

HONEST LIMITS

Test how Aura fails before the first success demo

Visible failure is more useful than a polished result whose limits remain unknown.



How to read this visual: follow an engineering task loop through honest limits.

DECISION

Design the failure before presenting the success

Aura can run format, lint, compile, test, documentation-test, and security-audit stages under project policy. A failure becomes observed state for the next plan. Warnings can remain advisory or become blocking gates. The repository, not the model as confidence, provides the feedback.

When something fails, the failure goes back into the loop. Aura can fix the cause instead of merely reporting that the first patch looked reasonable.

- Wrong repository state
- Tool failure
- Check failure

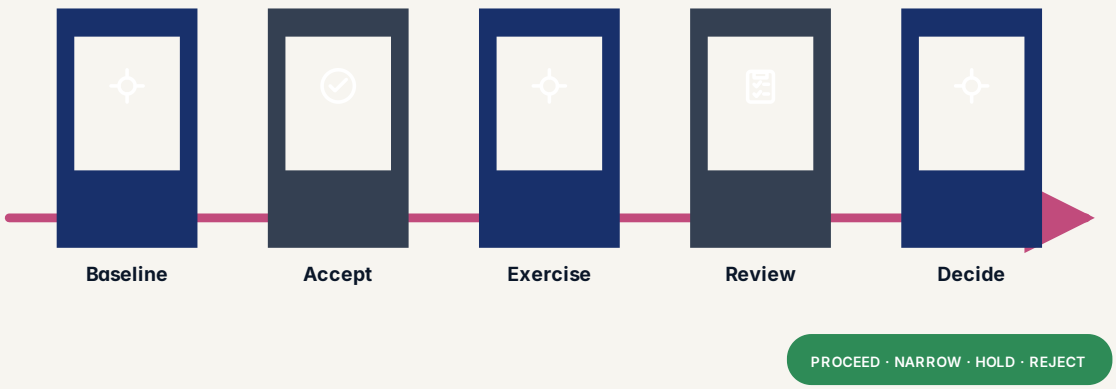
QUESTION TO CARRY

What happens when a source, permission, target, model, or supplier is unavailable?

ACCEPTANCE PATH

Close the Aura proof with a decision

Agree the decision path before the demonstration so a strong or weak result can change what happens next.



How to read this visual: follow an engineering task loop through acceptance path.

MECHANISM

Close the proof with an owned decision

The inner loop parses the model tool call, validates it against the schema and policy, executes it with timeout and progress handling, filters the result, and feeds the observed output into the next decision. Mid-turn steering can modify that decision before the next action.

Chains are defined as markdown files with YAML frontmatter, a name, description, ordered steps. Each step names an agent, specifies a goal, and may reference the output of previous steps through Jinja-like variables. Success criteria declare what counts as completion (substring match, regex, file existence, test pass). Built-in chains cover debug, feature implementation, optimisation, refactor and review. Drop a markdown file in the chains directory to define a custom one, no rebuild.

- Baseline
- Accept
- Exercise
- Review

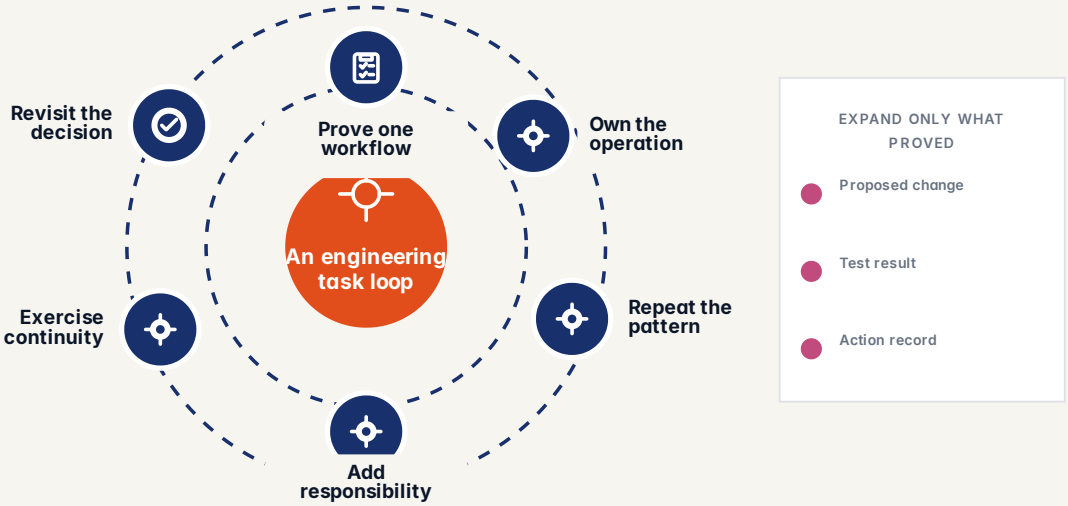
QUESTION TO CARRY

Are success, counter-metrics, failure cases, and stop conditions agreed first?

ADOPTION

Adopt Aura through owned choices

Move from one bounded result to a repeatable operating capability without making the scope vague.



How to read this visual: follow an engineering task loop through adoption.

DECISION

Expand only what proved reusable

MCP servers are declared in Aura's config: command, arguments, environment. On session start, Aura spawns them as managed subprocesses, runs the typed capability handshake from `dweve-protocol::mcp::capability`, and registers the discovered surface in the same tool registry the builtins live in. Resources, tools, prompts, sampling and logging capabilities all flow through the same typed message bus from `dweve-protocol::mcp::message`. External tools aren't trusted by default, they inherit per-tool permission levels, audit logging, PII filtering and circuit-breaker protection.

Native API backends, compatible API shapes, custom model entries, streaming, rate limits, capability metadata, complexity routing, and fallback sit behind one Aura interface. The runtime records which model and route were used without pretending every provider supports the same tools, context, or thinking modes.

- Prove one workflow
- Own the operation
- Repeat the pattern
- Add responsibility

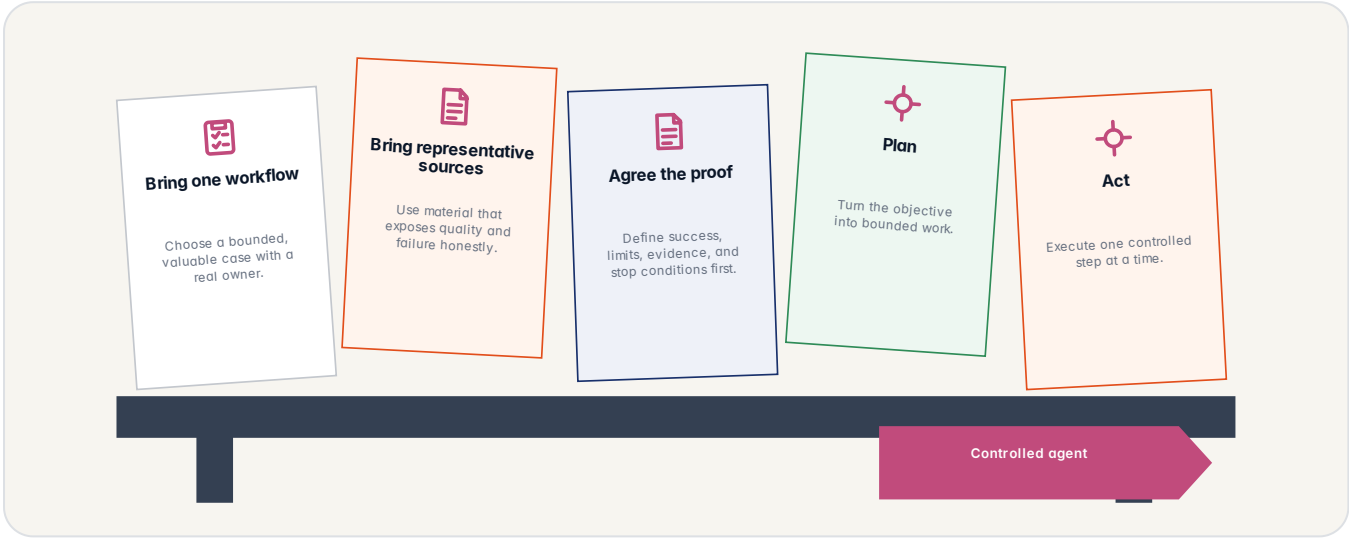
QUESTION TO CARRY

What can be reused safely, and what requires a new decision and proof?

WHAT TO PREPARE

What to bring to the Aura working session

A small amount of real context produces a better conversation than a broad catalogue of hypothetical features.



How to read this visual: follow an engineering task loop through what to prepare.

KEEP IT BOUNDED

Sensitive production data is not required for the first conversation. Bring representative material and the rules that define a useful result.

MECHANISM

Bring enough reality to make the session useful

Aura is around 330 KLoC of Rust across 433 source files in src/, compiled to one static binary with no runtime dependencies. The atlas below is the real module tree by file count, ui and agents and tools carry the weight, with security, audit and memory small and sharp.

Type your question or request in plain English. Aura understands context from your current file, project structure, and conversation history.

- Bring one workflow
- Bring representative sources
- Agree the proof
- Plan

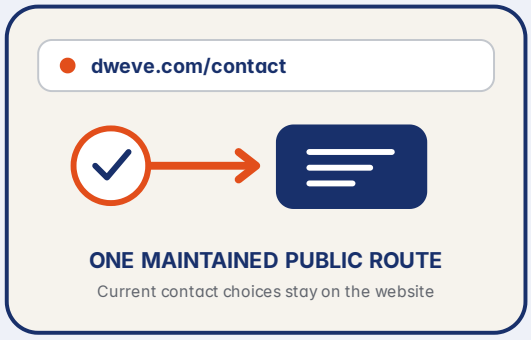
QUESTION TO CARRY

Which real sources, rules, owners, and examples will make the first session honest?

THE INVITATION

Continue the Aura conversation

Scope a briefing, demonstration, or proof around the workflow and decision that matter to your organisation.



CONTINUE ONLINE

dweve.com/contact

Current route and contact choices

START A CONVERSATION

Continue through the current contact page

Scope a workflow or ask a technical question, then use the contact page to begin.

START	**dweve.com/contact**	Use the maintained public contact route
BRING	**One bounded question**	Workflow, audience, decision, and desired evidence
CHOOSE	**Briefing · demonstration · PoV**	Select the smallest useful next step
VERIFY	**Current scope and terms**	Confirm availability, pricing, and responsibilities before commitment

DECISION

Turn the brochure into one bounded next step

Ask why something keeps failing, where part of the project lives, or how a piece works. Ask Aura to add checks, build something new, or tidy up. It uses your files, project layout, instructions, and conversation, so you can start with the outcome instead of a list of commands.

Aura is not a chatbot that suggests snippets. It is an autonomous agent that understands your entire codebase, plans multi-step changes, executes them with your approval, and documents every decision in an immutable audit trail. It scales from individual developers to enterprise teams with multi-agent orchestration.

- Start: dweve.com/contact
- Bring: One bounded question
- Choose: Briefing · demonstration · PoV
- Verify: Current scope and terms

QUESTION TO CARRY

What is the smallest useful next conversation Dweve should prepare?