



Continue

Written by a machine, read by a skeptic

How a hosting control panel was rewritten by prompt, from a forum bet to a first pass at security. Told from the decision logs, the transcripts and the code.

by **mikho**



About this book

This is the story of LESta, a Laravel and React rewrite of a hosting control panel, written by Claude Code from prompts, with as little hand-edited code as possible.

The code is public at github.com/mikho/LESta. The two LowEndSpirit threads that followed the project are listed at the end. Forum members are not named. The numbers were measured from the session transcripts and the repository, and where a number is uncertain the text says so.

Free to read and to share. The book stays open: more chapters will be added, and the first real install is still to come.

Version 1, September 2026.

Contents

BEFORE THE STORY

The panel	4
-----------	---

PART ONE. THE BET

1	A day or two	7
2	What the report found	10
3	The plan hat	14
4	The prompt I got wrong	18

PART TWO. RULES BEFORE CODE

5	A fresh pair of eyes	23
6	Four questions with no right answer	27

PART THREE. FIRST CODE

7	The session that refused	33
8	Go, and a real nginx	38
9	DNS, and an hour of AppArmor	43
10	Two web servers and one front door	50

PART FOUR. THE LONG CONTINUE

11	Thirty-three hours	55
12	Tokens	58
13	Databases, cron and a shared identity	62
14	Eight phases in twelve hours	68
15	Bundles, and going back	74
16	The test that had never run	79
17	The day of thirty-eight commits, part one	84
18	The day of thirty-eight commits, part two	91

PART FIVE. A PRODUCT, NOT A DEMO

19	What Vesta never had	98
20	Uninstall	105

PART SIX. THE AUDIT

21	The machine reviews itself	113
22	The manual nobody had	118

PART SEVEN. WHERE IT STANDS

23	Where it stands	122
----	-----------------	-----

APPENDICES

Appendix A. The numbers	127
Appendix B. Glossary	130
Appendix C. Where to read more	132

The panel

A hosting control panel is the thing between a person with a five dollar VPS and the pile of services that make a server useful. Web server, DNS, mail, databases, cron, backups, SSL. You click a button and a website exists. Vesta was one of the free ones.

For a long time it was the default answer on cheap hosting forums to the question of how to give someone a panel without paying per account. Small hosts ran it. Hobbyists ran it. I ran it myself, as a side project for a free shared hosting experiment with a name I still apologise for, and it got badly hacked.

That last part is not unusual, and this book starts with why.

What happened to Vesta

In May 2018 the project's installer was changed so that anyone installing Vesta on a fresh server also installed something else. According to the VulnCheck advisory, the malicious code sat in the repository from a commit on 2018-05-31 until it was removed on 2018-06-13. It sent the new server's admin password and domain to an outside address, and it dropped and ran a DDoS bot called Linux/ChachaDDoS. Kaspersky analysed the compromise. Vesta acknowledged that the exploitation was happening in October 2018. It is now tracked as CVE-2018-25117.

In March 2021 a security research group published two chained privilege escalation flaws, CVE-2021-30462 and CVE-2021-30463, that turned an ordinary panel user into root on Vesta 0.9.8-24 and earlier. Their advisory says the vendor had stopped answering three months before publication and that no patch existed. It pointed readers to two forks that had patched the problems.

One of those forks was HestiaCP, whose first release came in June 2019. A LowEndTalk thread from July 2019 shows how its supporters explained the split. One member who followed the project said they had started by forking to their own infrastructure and changing how the whole stack was deployed and installed, which they called one of the biggest problems in Vesta to begin with, and which they linked to the incidents of the year before, when Vesta's file servers had been compromised. Another member added that Vesta's developers never revealed the final point of entry. I cannot confirm that last claim from any source I have read. It is what a user said at the time.

On 2024-03-16 a private equity firm in Canberra, Outroll Ventures, announced that it had bought Vesta. It promised money for development while keeping the core open source. A page on the Vesta site later described a rebuild called Vesta 2.0, first started in Go and then moved to Rust, with a licence change from GPL to the Business Source License and a codebase that is private while it is being built. The page carries no date. By the time this story starts, in the summer of 2026, forum members were saying the project had been quiet since the announcement, with roughly 400 open issues and no pull requests.

None of this proves that Vesta's code is bad. A backdoor in an installer is a supply chain problem. Missing maintenance is a people problem. What it does show is a popular free tool that a lot of small servers depended on, with a history that any owner of one of those servers would want to know about.

What this book is

It is the story of what happened when I decided to write a replacement, and to do it in an unusual way. I would not write the code. I would write prompts, and Claude Code would write everything else. Manual edits to files would be kept to a minimum, and only made when I asked for them in a prompt.

The project is called LEStA. The code is public at github.com/mikho/LEStA. The full record of every phase lives in a decision log that has grown to about 393 KB. This book is not that log. It is the story the log makes possible to tell: why it started, what was tried, what broke, what got fixed, and where it stands now.

Two threads on LowEndSpirit followed the project while it happened. One was a discussion thread, where members asked questions and pushed back. The other was a closed thread of build-log posts. Where a member said something that matters to the story, I write "a member" and leave out the name. If you want to see where a quote came from, search LowEndSpirit for the phrase and you will find the thread.

The book is chronological. Sometimes a problem in one chapter gets fixed in a later one, and I will say so when that happens. Feelings are left out as far as I can manage. The numbers are measured from the Claude Code session records and the repository, and where I had to estimate, the text says so and estimates on the harsh side.

Length was not a goal. The story was allowed to be as long as it needed to be, and there will be more chapters added as the work moves on. The next step, after the last chapter that exists today, is the first real install on a real server. That has not happened yet.

SOURCES

- VulnCheck advisory for CVE-2018-25117: vulncheck.com/advisories/vestacp-debian-installer-malicious-backdoor-supply-chain-compromise
- Hackread coverage of the 2018 attack: hackread.com/vesta-control-panel-servers-ddos-malware-supply-chain-attack
- SSD Secure Disclosure advisory, VestaCP LPE vulnerabilities, 2021-03-20: ssd-disclosure.com/ssd-advisory-vestacp-lpe-vulnerabilities
- Outroll announcement of the acquisition, 2024-03-16: outroll.com/news/outroll-acquires-vestacp
- Vesta progress update on the 2.0 rebuild (undated): vestacp.com/docs/vesta-2-update
- Hestia 1.0 release, June 2019: cpanels.review/hestiacp-1-000-0-190618
- LowEndTalk thread "Hestia Control Panel (VestaCP Fork)", July 2019, read 2026-09-24: lowendtalk.com/discussion/159249/hestia-control-panel-vestacp-fork

PART ONE

The bet



A day or two

The thread was about control panels. I had gone back to Vesta's website and noticed it was under active development again, and the repository had been updated three weeks earlier. A member told me it had been bought by an Australian company in 2024, with big promises about a 2.0 and not much since. Another member checked the issue tracker for me: about 400 open issues, no pull requests. Their advice, given with the disclaimer that they were biased, was to stay away from it.

I had a look at the code myself, after practice, and from a glance most of what I remembered as its downfall was still there. I pasted one function into the thread as an example.

```
# Search objects
search_objects() {
    OLD_IFS="$IFS"
    IFS=$'\n'
    for line in $(grep $2='\$3\' $USER_DATA/$1.conf); do
        eval $line
        eval echo \$$4
    done
    IFS="$OLD_IFS"
}
```

This is how Vesta reads a record back. Every domain, mailbox and user is a line of shell variable assignments in a text file. To look one up, the function finds the line with `grep` and then hands it to `eval`, which means the shell runs it. A value that contains the wrong characters is no longer data. It is a command, and it runs as root, on every list screen, every time.

I wrote under the function that I remembered code like that from back in the day, and that I would vibe-code something better in a day or two.

The answer

A member who had been involved in the original Hestia fork answered. They were working on a new fork of their own, and offered it as a starting point: no self-built web server packages that need their own apt repository, no npm build. Then came the sentence that turned my comment into a bet. Even with agentic code, they wrote, it takes weeks or months to turn everything over and over again. A day or two would not do.

They were right, and the rest of this book is the evidence. But at the time, on 2026-08-25, I wrote back two words.

Challenge accepted.

The rule

The reply carried more than two words. I said I would write the first prompt and start a new thread where I would publish updates as I moved along. I said that the GitHub repository and a Laravel skeleton already existed. And I set one rule for the whole project: all I would do is prompt. Changing files by hand would be kept to a minimum.

That rule is the only unusual thing about the project. Writing a hosting panel is not unusual. Plenty of people have done it, most of them with more experience of hosting than I have. What was different was who would type the code, and that I had said in public that I would not.

The rule has a cost that I understood only partly on day one. If I do not write the code, every mistake in it is either the model's or mine, and I cannot tell which one until I look. When the model gets something wrong, I have to notice it. When it gets something right that I asked for badly, I have to notice that too. Chapter 4 has an example of the second kind.

What I expected

I expected simple. A rewrite of a shell-script panel in Laravel sounded like a day or two, as I had said. I would write one good prompt, review the result, run a few more prompts, and have something to show.

That is not what came back. The model did not plan a weekend. It planned a system. The plan had a control plane in Laravel and React, a separate agent written in Go, a versioned protocol between them, an installer contract, and a list of things the codebase would never be allowed to contain. I will come back to why in the next two chapters.

The expected timeline was a day or two. The repository's first commit is dated 2026-08-25. The last commit at the time of writing is dated 2026-09-18, and the work is not finished.

The repository, the same night

On 2026-08-25 the repository at github.com/mikho/LEsta had four commits: a fresh Laravel application, Pest for tests, Laravel Boost, and a configuration change for Boost. That is the whole of what I made by hand, and I used the framework's own installers for it. A second thread went up the same day, this one for the discussion, with a short post that promised a journey and warned members that some of my decisions would make them question me as a person. Members replied with encouragement. One asked what setup, model and costs I planned. I told them it would be Claude, with models and effort varying depending on how many tokens I had left on my company account. That last clause turned out to be the most accurate prediction I made.

The next chapter is about the first prompt, and what came back.

SOURCES

- The comment where the bet was made: lowendspirit.com/discussion/comment/240245, dated 2026-08-25.
- The discussion thread, posts of 2026-08-25: lowendspirit.com/discussion/11268/vibe-coding-a-vestacp-alternative.

What the report found

Before I wrote the first prompt, I had already asked for a review of Vesta's code. The result was a long report, published as a Claude artifact and titled "A panel that forks a shell to read a file." My first prompt for the rewrite links to it in its first paragraph. I read it, and the rest of this chapter is what was in it, because every later decision in the project answers something on this page.

The review was pinned to one commit of the Vesta repository, `5f4ee2ef`, so that every number in it refers to one exact snapshot and not to whatever the repository looks like on the day you read it.

The size of the thing

Vesta is 102,000 lines across three languages. There are 379 bash scripts, 34,600 lines, and they hold all the business logic. There are 136 PHP endpoints, 42,900 lines, whose entire job is to check the session and build a shell command. On top of that sits a React application with twelve list screens that share one design.

Four tiers, then, and each one talks to the next in a different language. The browser sends JSON to PHP. PHP builds a command string and calls `exec()`. The command is a bash script, which reads and writes flat text files under `/usr/local/vesta/data`. The report's own summary of the consequence is short: the type system evaporates at every boundary. A value that is a number in one tier is text in the next and nothing at all in the one after that.

The read path

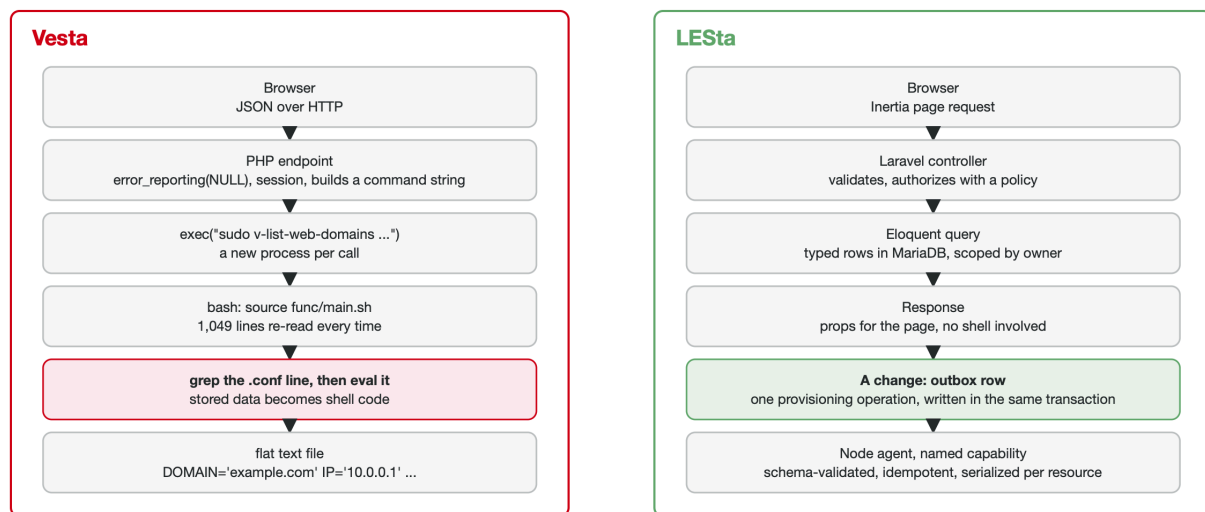
The function I pasted into the forum thread is the centre of it. The report found the same pattern at 105 sites in the scripts, and 191 uses of `eval` in the whole tree once every kind of use is counted.

Every record on disk is a line of shell assignments:

```
DOMAIN='example.com' IP='10.0.0.1' U_DISK='420' SUSPENDED='no' ...
```

To read it back, the code finds the line with `grep` and runs it with `eval`. That is the storage format. It is not one shortcut that somebody forgot to clean up. Every list screen in the panel reads its data this way.

Figure 2. Reading one record



Vesta: six exec() calls, each starting sudo and bash, to draw one list screen. LESta: a database query, and the node is only contacted when something changes.

The safety of that design rests on the validators that check a value before it is stored. The report found that these are blocklists of shell metacharacters instead of allowlists, and that they are applied differently from one resource type to the next. A validator for one field rejects a backtick and a dollar sign. The one for autoreply text rejects the same two but lets a semicolon and a pipe through. The boundary between text a tenant typed into a form and code that runs as root the next time anyone lists that resource is, in practice, whatever characters somebody remembered to block for that one field, years ago.

Eight findings

The report ranked its problems by blast radius and numbered them C-01 to C-08. In plain terms:

No locks (C-01). 103 scripts change shared config files. There is not one `flock` call in any of them. Every write reads a whole file, changes it, and writes it back with `sed -i`. Two admins on the same reseller account each add a domain within one second. Both scripts read the account's domain count as 5, both write 6, and one domain has vanished from the count. Nothing crashes and nothing is logged. The quota drifts until, months later, a disk usage figure stops adding up and nobody can say why.

JSON built with `echo` (C-02). 73 of the 80 `v-list-*` scripts assemble their JSON output by string concatenation with no escaping. A mail autoreply with a quote in it, a cron command, a user's last name with an apostrophe: any of them produces JSON that is invalid.

Config values `eval`-ed back into the shell (C-03). The read path above.

An arbitrary file write in the API (C-04). In `web/api/index.php`, and in a byte-identical copy at `web/api/v1/index.php`, there is a branch called `v-make-tmp-file`. It takes a path from one POST field and content from another and writes the second to the first. There is no constraint on the path. It sits

behind authentication, and that is the only thing between a logged-in tenant and a file written wherever the web server process can reach. The report also noted that the two copies are identical, so a fix applied to one silently misses the other.

Error reporting switched off (C-05). The second line of essentially every API endpoint file is `error_reporting(NULL)`. This is the reason the JSON bug in C-02 never shows up. PHP gets the broken JSON, fails to decode it, tries to loop over the result anyway, and dies, and the setting hides the death. The operator sees a blank page and an empty log. The report found a live example in the web list endpoint: a field for cached file extensions has been returning the wrong value to every user, with no error anywhere, because a variable is read that is never assigned.

The CSRF token (C-06). It is stored in `localStorage`, so any script injected into the page can read it. It is sent in the query string, so it ends up in access logs and Referer headers. And 39 of the 136 endpoints never check it at all.

Session pinning on headers the client controls (C-07). The fingerprint that ties a session to a browser includes five `X-Forwarded-*` headers. Someone who has the session cookie also controls those headers, so the check adds nothing for security. A real user behind a rotating CDN, on the other hand, is logged out whenever the forwarded chain changes.

Shell arguments without escaping (C-08). Twelve endpoints call `exec()` on a string with the username inside it, unescaped. The other 439 call sites use `escapeshellarg`. The report was careful here: the username comes from a validated field, so it was not a live injection. The safety depended on a validator three tiers away.

What it costs to draw one list

The performance chapter of the report puts it in one sentence: the unit of work is not a query, it is a process.

Drawing the web domains list costs six `exec()` calls. Each one starts `sudo`, then `bash`, then re-reads a 1,049-line shared script, and only then runs the code that was asked for, which itself calls `grep`, `wc`, `cut` and `awk` as further processes. After every change the panel repeats the whole session set-up to refresh its counters. Concurrency is limited by how fast a machine can fork processes, not by how much data there is.

The frontend had its own problems. 48 places use the array index as a React key, which on a sortable list makes a checkbox appear on the wrong row after a sort. One promise was written so that on a failed request it never settles, which gives a permanent spinner and no error. And twelve list screens carry the same seventeen functions, about 400 lines each, 4,666 lines in all. The report's remark about that number is the useful one: the copies have drifted, so a bug fixed in one is still live in the other eleven.

What the report recommended

It recommended fixing Vesta. It laid out four phases, each independently shippable: turn error reporting on and add locks and JSON escaping; collapse the duplication onto one abstraction per tier; cache and paginate; and finally retire `eval` as the read path and move the record store to SQLite. It also delivered code for the first phase and for a reference implementation of the second: a command gateway that escapes by construction, a response class with proper error handling, a JSON and locking library for bash, a shared list hook for React.

Two limits are stated in the report itself. The new PHP and JavaScript were never run, because the machine it was written on had neither PHP nor Node installed. Only the bash library was tested, against a string with a quote, a backslash and a tab. And a whole area was left out: "The installer scripts under `install/`, the RPM and DEB packaging in `src/`, and the update scripts in `upd/` were outside this pass."

That last line matters for the story of Vesta. The 2018 incident described in the preface happened in the installer. The report did not look there.

Fix it or replace it

The report's plan is a fair one if the goal is to keep Vesta alive. I read the same numbers and chose the other option, and the first prompt says so in one sentence: "The UI should be similar to the original project but the backend should be completely re-written because of all the vulnerabilities and bad decisions made in the original code."

I can give the reason in the terms the record supports. The report's fourth phase, retiring `eval` and moving the records into a real store, is close to a description of a rewrite of the backend. The only difference between the two plans was order: fix around it for three phases and then replace the middle, or replace the middle first. On 2026-09-03 I put it to the forum this way, when someone asked how it was going: "the project changed more than a bit with the decision to write something without all the current bugs and exploits." The same decision shows in the plan's list of decisions: fresh installs are the initial migration strategy, and importing Vesta data is left for later. LEStA does not try to inherit a single Vesta data file.

The report gave me numbers to argue from. It did not give me a design. That came from the first prompt, and it is the subject of the next chapter.

SOURCES

- The build-log thread, part 1, with the public version of the same findings: lowendspirit.com/discussion/11288/lestra-the-vibe-coded-version-of-vestacp.
- The discussion thread, post of 2026-09-03: lowendspirit.com/discussion/11268/vibe-coding-a-vestacp-alternative.
- Vesta at the pinned commit: github.com/outroll/vesta, commit `5f4ee2ef`.

The plan hat

The evening of 2026-08-25 I had very little time. I used it to write one prompt.

The prompt is 235 words. Here is most of it.

I'm planning on rewriting the web hosting panel which has the code in this github repository (Vesta). I ran a control of the code and the report can be found here (the report from chapter 2).

The codebase will be done with latest, stable version of Laravel and React.

The UI should be similar to the original project but the backend should be completely re-written because of all the vulnerabilities and bad decisions made in the original code.

Act like a senior technical lead managing a real engineering team. Before writing code: ask clarifying questions, challenge bad decisions, identify scaling risks, suggest better approaches, prioritize simplicity. Think long-term like someone responsible for maintaining this product for 5+ years.

Then provide: technical decisions, tradeoff analysis, recommended architecture, implementation plan, production-ready solution.

The prompt then hands the model a second role. "You are now 4 elite AI agents working together on the same project," it says, and names them: Architect, Engineer, Reviewer, Optimizer. The Architect designs the system, the Engineer builds it, the Reviewer critiques it, and the Optimizer makes it fit for production.

The typos in the first quotation are mine and stay. The prompt was written in Swedish and translated. For the whole planning phase I used a second Claude project with one short instruction: this is the folder with the documentation, I will write prompts in Swedish, translate them to English so I can use them in my coding project. The Swedish title on the file that holds this prompt is the one thing that survived the translation.

Which window

I meant to run the prompt in Claude Code. My forum post from that night says that I put "the Plan hat on Claude". In fact the first three prompts of the project ran in GitHub Copilot Chat. I had not paid attention to which window I was typing into. The build-log posts I wrote a few days later say so, and so does the first entry of the decision log.

I do not know which model answered in Copilot Chat. Nothing I kept records it. The book therefore says "the plan came back" and does not say from what.

The mix-up has one lasting effect. The fourth prompt, later on 2026-08-26, ran in a Claude Code session that had never seen the first three. It found the repository with a plan, an architecture document and a directory of installer metadata, and nothing else. The rest of the project has the same property. A session starts with no memory, so what the project knows has to be written down somewhere that the next session can read it. That is why the vault ended up holding the plan, the decision log and every question I was asked. Chapter 5 shows what that looked like the first time.

What came back

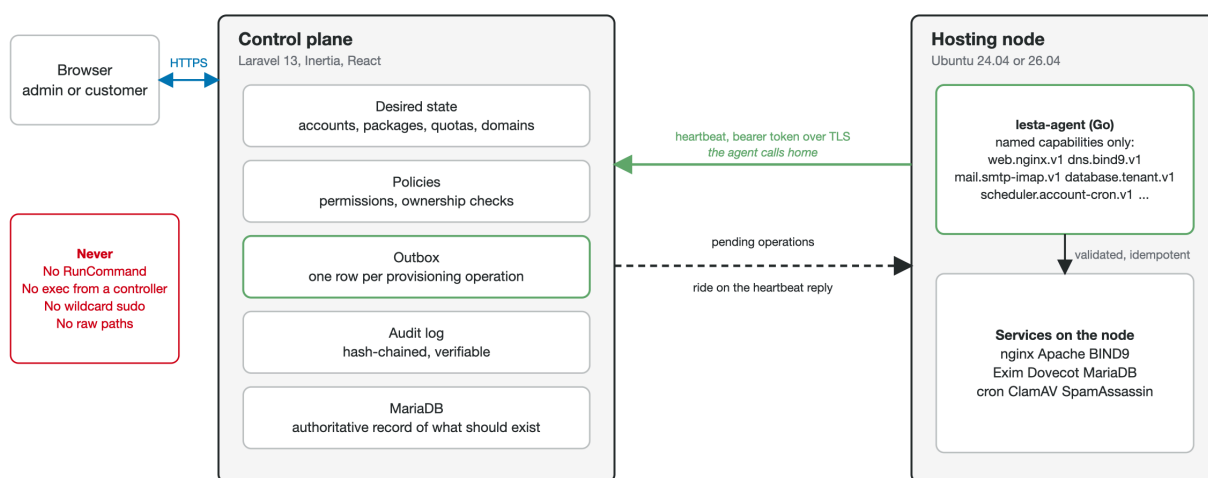
After what I called many long boring minutes, a plan came back. The forum post says almost 2,500 words. The plan as I have it today is 2,238. It is titled "Plan: LESta Hosting Control Plane Rewrite", it has seven phases numbered 0 to 6, and 28 numbered steps.

This is the plan as it is now. It has been edited since that first evening: the file names Ubuntu 24.04 and 26.04, which I only chose in the third prompt, and describes a blank-server choice of nginx, Apache or both, which I asked for in a prompt the next day. I do not have a copy of the first version, so I cannot say what the plan said before those edits, and the chapter after this one depends on that.

The architecture the plan settles on has three parts.

Figure 1. Who touches what

The control plane decides and the agent acts. Nothing in the control plane ever runs a shell on the node.



Every message names one capability, a resource ID, a desired-state version, an idempotency key and a deadline. The agent rejects any capability it never registered before it reads the payload.

A control plane in Laravel and React. It holds everything that is a decision and not an action: accounts, roles, hosting packages, quotas, the audit history, and a queue of provisioning jobs. Its database, MariaDB or MySQL, is the authoritative record of what should exist on each server.

A node agent in Go. Anything that changes a server goes through a small program running on that server. The control plane never touches the host. This is the biggest departure from Vesta, where PHP builds a shell command and runs it. The plan's own tradeoff analysis says what the choice costs: a

second deployable component and a second release pipeline. It says what the alternative would cost too. A PHP and bash adapter would be faster to build, but only as a prototype, because it would recreate the main risk.

A versioned protocol between them. Each message names one capability, carries a resource ID, a desired-state version, an idempotency key, a deadline, a digest of the request and a correlation ID, and gets a structured answer. There is a step for authentication: a short-lived bootstrap exchange, then per-node certificates.

The plan also writes down what the codebase must never contain. I quote the list because it is short and every item maps to something in chapter 2.

- No generic `RunCommand` .
- No arbitrary command-plus-arguments API.
- No `exec` from controllers.
- No wildcard sudo.
- No raw user filesystem paths.
- No secrets in props, logs or job payloads.
- No executable queue files.

Take the file-path item: it answers the arbitrary file write. The wildcard sudo item answers the fact that every Vesta script runs as root regardless of what it needs to touch. The last item is broader. It is the general form of what went wrong in Vesta, stored data treated as code, so the same failure cannot come back somewhere else under a different name, in a queue payload or a config template, say.

Seven phases

In order:

0. Decisions and the product contract: a capability matrix built from the legacy code, the supported environment, the threat model, and the database decision.
1. The control-plane foundation: accounts, memberships, roles, packages, nodes, audit events, idempotency receipts, provisioning operations.
2. The first vertical slice, web hosting, with a fake provisioner first and the real Go agent second.
3. The core resource slices: DNS, mail, databases, cron.
4. Higher-risk operations: filesystem access, backups and restores, provider operations, usage statistics.
5. Frontend parity and operations.
6. Review, optimization and release, with a Reviewer gate and an Optimizer gate.

I keep the numbering in this book only for the plan. The decision log I kept later has its own, finer numbering and it goes to 55. Those are called phases too, and the difference between "plan phase 2" and "log phase 2" matters, so I will write "the plan's phase" whenever I mean the first kind. Seven

phases became fifty-five. Most of the difference is that the plan gave a single line to things that turned out to need several phases each. DNS alone took four.

What the plan said it did not know

The last paragraph of the plan is a list of things it could not check. "The current environment does not contain the legacy Vesta checkout, and the linked Claude artifact is unavailable without access." Exact field mappings and vulnerability claims, it says, must be confirmed against a pinned commit before any migration work.

The planning session had not read Vesta's code. It was working from my prompt and, through it, from the model's general knowledge of the panel. The plan's first step therefore reads as an instruction to go and do the reading: pin a legacy commit, build a capability matrix from named directories, and treat the report as supplementary until it has been checked.

That step was carried out on 2026-08-26 and it changed some numbers. Chapter 6 has the details.

Decisions before any code

The plan closes with a list of decisions. The requested first release is core hosting plus self-hosted mail, no billing. One managed node to start, but every resource is assigned to a node and the protocol is multi-node ready. Provider administrators and customers as the only tenancy model, reseller nesting reserved for later. MariaDB or MySQL is a hard requirement. A Go service is the recommended agent. Fresh installs first, Vesta import later. Mail is in scope but gated behind its own threat model.

The last line of the decision list is the one I acted on: "No production implementation should begin until this plan is approved."

I had a question for the forum and I asked it that night. Should the original interface be kept, or a modern one built? A member said they liked the simplicity of the original and would take a subtle refresh. I agreed the next day: no icons, just text, properly formatted.

The plan needed reading. I had said I would read it all the next day and probably run a couple more prompts before any code was written. The next chapter is about the second of those prompts, and about a mistake in it that turned out to be mine.

SOURCES

- The discussion thread, posts of 2026-08-25 and 2026-08-26: lowendspirit.com/discussion/11268/vibe-coding-a-vestacp-alternative.
- The build-log thread, part 2, which states that the first three prompts ran in GitHub Copilot Chat: lowendspirit.com/discussion/11288/lestra-the-vibe-coded-version-of-vestacp.

The prompt I got wrong

The second prompt is 2,324 words, which makes it longer than the plan it builds on. I put it together in the spare time I had while waiting for a meeting to start, and sent it on 2026-08-26.

It has four tasks. The first is to audit the plan's story about installation: how a bare Ubuntu server gets nginx, BIND, Exim, MariaDB and the rest onto it, and who owns the configuration files afterwards. The plan had committed to a Go agent and to never running host commands from Laravel, but it had said nothing about how the services get there in the first place. The second task is to create a directory called `.install` at the root of the repository, with a written contract that every installer must satisfy. The third is to decide the order in which services are taken on, with a dependency graph. The fourth is to choose between three ways of building the whole thing: installers first, interface first against fakes, or one service at a time all the way through.

It repeats the four-role setup from the first prompt. It adds a rule I had not written before: "Where anything below conflicts with the plan, say so explicitly instead of silently resolving it." And it ends on two lines I added by hand after the translation was done.

Do NOT write any code at this point, we are still in the planning phase of the project. As you work thru the prompt, ask me questions if there are any doubts on what option to chose, unless it is clearly decided.

What the answer did

It asked questions, and I answered them. Which database? MariaDB 11.4 only, which was the model's recommendation. Which Ubuntu releases? The model recommended 24.04 alone, on the grounds that it halves the integration testing for a one-person project. I answered both, 24.04 and 26.04. Which ACME challenge mode for certificates? It recommended HTTP-01 only. I answered both, HTTP-01 and DNS-01. Should mail be on by default on a new node? It recommended that the operator has to opt in, and I agreed. It produced a design decision record, ADR 0002, 194 lines, and the `.install` directory: a base layer, a manifest format, an installer contract, a service directory for each planned service. It chose the third build strategy, one vertical slice per service.

Then I read the chat log of the session and found this sentence:

No Apache directory was added because the ADR explicitly defers Apache support.

I did not understand it. My prompt said, in the third task, that the ordering should account for "the web server, and whether the first release supports nginx, Apache, both, or nginx as a proxy in front of Apache as the original does, and what that choice costs in template surface." Vesta supports all of

those. I expected the new panel to support all of them too, and I had not expected to be asked, and certainly not to be answered with a deferral.

The third prompt

I sent a short prompt, on the same day.

In the installation phase on a blank server, the project will support the option to select nginx, apache or both. Include this and report what steps are to be taken after.

The result was a list of ten next steps, ending with "Write the real installer only after those capability tests pass" and a line about automated profile selection and rollback tests. I wrote in the forum that the deferral was a decision the model had made on its own, without consulting me.

Who was right

A member replied the next day. They pointed at the third task of my own second prompt:

Decide and justify the order in which the underlying services are taken on, from a bare Ubuntu LTS node to a complete panel.

The list about nginx and Apache came right after, they wrote, as input to that decision, with the instruction to account for all of them. So the model had done what I asked. It had decided the order, and its decision put Apache later. They added that the model's word "deferral" was pointing at the ADR it had written, which I could read.

They were right, and I said so in the thread: "that was my mistake, as in my head, services are more the function... like serving web pages, not the product, Apache, nginx, or any other web server. When I wrote the prompt, I had that in mind and that VestaCP offered both options. That's why it is important to write proper prompts and be super-clear about your own thought process and purpose."

I have a few observations about it that the thread did not have room for.

First, I had written a list of things the ordering should account for, and to me they were requirements. In the text I sent, they read as options with a question mark. "Whether the first release supports nginx, Apache, both" is a question that has answers. A model that is told to "ask me questions if there are any doubts on what option to choose, unless it is clearly decided" will treat an open question as open. Mine was open on the page and closed in my head.

Second, I cannot tell whether the plan had already closed it. The plan as I have it today contains a sentence in step 2 that reads "A blank server may select nginx, Apache, or both." Later the same night a fresh Claude Code session read the plan, concluded that it already specified exactly this, and marked as false the ADR's claim to be resolving a conflict with it. That is either because the plan said so from the first evening and my second prompt reopened a question it had answered, or because the plan had

been updated in between. I do not have the first version, so I do not know which. Either way it points the same way. The gap was between what I meant and what my text said, and the model filled it with a reasonable reading.

Third, it cost very little. It was caught the same day, by reading a log line. The second prompt, the ADR and the three-profile decision were all fixed before the first commit. But it is the first evidence in this project for a habit I had to build, which is to read what the model says it decided and ask whether I said that.

The ADR in the repository today supports all three profiles. It says that nginx, Apache or both can be selected on a blank server, and that in the combined profile nginx owns public ports 80 and 443 and proxies to Apache on a fixed loopback port. Tenants cannot choose or change the profile.

Four choices, one reply

The next step was the one that made the decisions irreversible. The list of next steps from the third prompt asked me to approve four things:

1. The database: MariaDB 11.4 LTS, recommended, against MySQL 8.4 LTS.
2. Ubuntu support: 24.04 and 26.04, recommended for long-term scope, against 24.04 only as the lower-cost option.
3. ACME challenge mode: HTTP-01 and DNS-01, recommended. HTTP-01 is simpler for web certificates. DNS-01 is required for wildcard certificates and works better when the panel manages DNS itself.
4. The web capability contract: one shared `WebProvisioner` contract with `web.nginx.v1`, `web.apache.v1` and a combined profile underneath it, with nginx on the public ports and Apache on `127.0.0.1:8080`.

I replied with two lines.

Ok, lets do according to your recommendation. and then go ahead with the implementation sequence

I approved all four, including the more expensive Ubuntu choice. It is the first example of what the next four weeks looked like for most of my part of the work. The model offered a small set of options, marked one as recommended and gave the reason, and I took the recommendation most of the time. The decision log keeps a file with every question I was asked, with the options and my answer. Its first 24 entries cover the first two or three days. Of those, 18 were answered with the recommended option and 6 differently: two of them the Ubuntu and ACME questions above, and a few later ones that I will point out when they come.

The first commit

The result of that reply was commit `9431d04`, "Document service installation architecture", on 2026-08-26 at 22:25. It changes 34 files and adds 618 lines. There is the ADR. There is a `.install` directory with a README, an installer contract, a manifest schema and a base layer definition. There are three profile files for nginx, Apache and both. And there are service directories for ACME, Apache, backups, BIND9, cron, the firewall, mail, MariaDB, nginx, node health and statistics, each with a manifest and a README, none with an installer. There is also a two-line change to `.gitattributes`, because a directory that starts with a dot could have been left out of a source archive, and the second prompt had asked whether it would be.

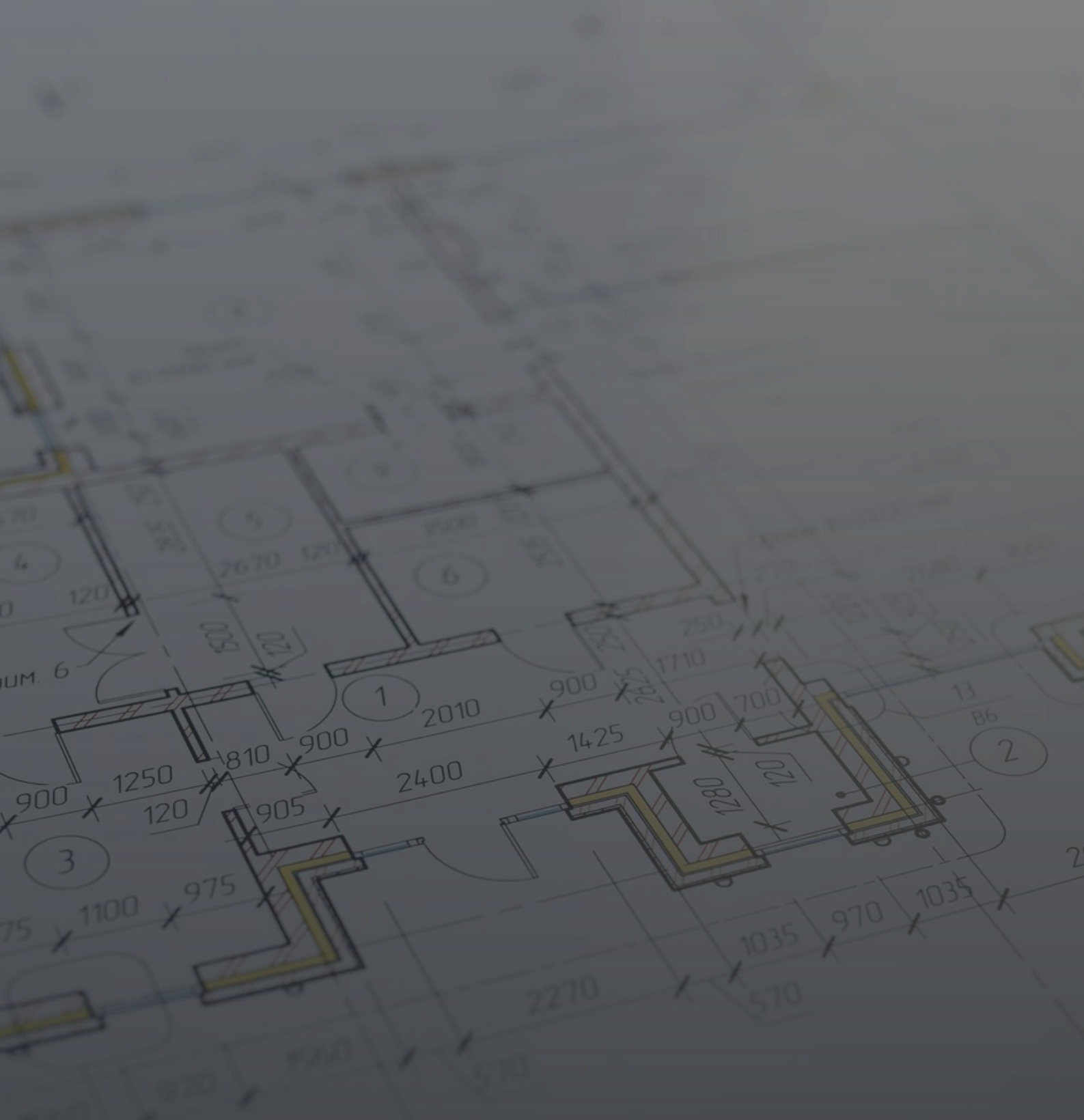
The record of the commit lists what it verified, and the list is longer than the code. Every installation JSON file parses. The web profile metadata matches its declared listener rules. The installer contract defines a dry run, a preflight check, pinned artifacts, checksums, signatures, deterministic exit codes, structured output, redacted logs, idempotent reruns and rollback. Every tracked `.install` file survives a source archive generated from the staged tree. And the negative check that mattered most: no executable installer, no shell gateway, no generic `RunCommand` and no service implementation had been added.

The commit contained no features. It was a set of decisions, a list of checks and a set of nearly empty directories, and it was the first commit in the repository that had anything of the plan in it. The four earlier ones were the Laravel starter kit, Pest, Laravel Boost and a Boost setting.

The next chapter starts with the first session that had never seen any of this, and with what it made of it.

SOURCES

- The discussion thread, posts of 2026-08-26 and 2026-08-27: lowendspirit.com/discussion/11268/vibe-coding-a-vestacp-alternative.
- The decision record written in that commit: github.com/mikho/LESt/blob/9431d04/docs/adr/0002-service-installation-and-build-sequencing.md.
- Commits, each at github.com/mikho/LESt/commit/ followed by the hash: `9431d04`.



PART TWO

Rules before code

A fresh pair of eyes

The first three prompts had run in one tool. The fourth ran in another, and it started with nothing.

On the evening of 2026-08-26, after commit `9431d04` was pushed at 22:25, I opened a Claude Code session in the repository. That session had no memory of the plan, the second prompt, the three prompts or the discussion. What it had was a Laravel starter kit, a `docs/adr` directory holding one decision record, and a `.install` directory of manifests and READMEs. The plan it was supposed to treat as the governing document lived in my Obsidian vault, not in the repository, so the session could not even find it at first.

It started by reading what was there, and checking it.

Three conflicts that were not conflicts

The second prompt had told the planner to say so explicitly whenever something conflicted with the plan. The decision record, ADR 0002, does report conflicts. The fresh session checked each one against the plan, and found that three of them did not exist.

The ADR said the plan had a PostgreSQL question near its fifth phase. The plan never mentions PostgreSQL near that phase.

The log lists a second claim about the Ubuntu releases the same way: the plan already names Ubuntu 24.04 and 26.04 in step 2 of phase 0, so there was nothing for the ADR to resolve.

The ADR said the plan did not define how a blank server selects nginx, Apache or both. The plan's same step 2 already specifies it, with nginx owning ports 80 and 443 in the combined case and Apache on a fixed loopback port.

In each case the ADR had "resolved" a disagreement by adopting a decision that the plan had already made. That is a small fault, and it is not a harmless one. A reader of the ADR would come away believing that there had been three disputes and that the ADR had settled them, when the plan had settled them first. That kind of confusion is how a team, over time, forgets what was actually decided and what was merely repeated. The session corrected the text. It found the problem because it had no reason to trust the ADR.

Here I should return to what chapter 4 left open. The ADR, as committed, said that the plan did not define a blank-node selection, and this session says the plan already did. The plan as I have it today does say it. If the plan said so all along, chapter 4's mistake was my prompt reopening a settled question, and the ADR then made a fourth false conflict of it. If the plan was edited after the Apache prompt, then this session was reading an already updated plan. I do not know which. The decision log, which I read for this book, does not say when the plan file was edited.

The firewall that could be skipped

The other finding was a real defect, and it is the one I would point a sceptic at.

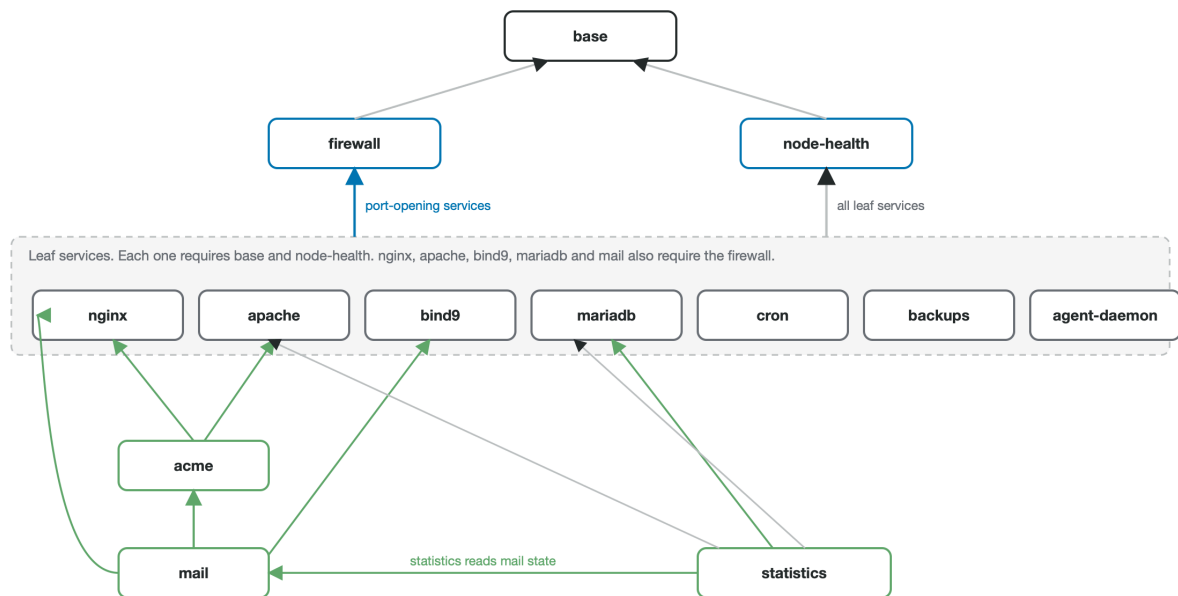
Each service in `.install` has a manifest, a small JSON file that says what the service provides, what it depends on, which ports it opens and which directories it owns. The installer order is not written into any script. It is computed by walking these manifests. If service A declares that it depends on a capability, an installer for A refuses to run until something that provides that capability has been installed and has reported healthy.

The firewall service provided a capability named `firewall.baseline.v1`. The base layer, separately, had declared a capability of its own with the same purpose, named `base.firewall.v1`. Nothing provided the first name to anyone who needed it. nginx, Apache and BIND had all been wired to depend on the second name, the base layer's. Mail and MariaDB, both of which open inbound ports, had no dependency on a firewall at all.

The result is easier to see with one sentence. The dedicated firewall step, which the ADR's own reasoning describes as a hard prerequisite for anything that listens on the network, could be silently bypassed, because the dependency graph pointed at a capability that the firewall step never produced. The manifests parsed as valid JSON, which was one of the checks in the first commit. The graph in the ADR did not match them.

Figure 3. Install order is data, not script order

Arrows point from a service to what it requires. Read from the service manifests in `.install` and checked by the contract validator in CI.



The firewall arrow is the one chapter 5 found missing: the graph pointed at a capability that nothing produced.

The session fixed the names, added the missing dependencies to mail and MariaDB, and redrew the ADR's dependency graph from what the manifests actually declared. Several manifests depended on capabilities, firewall and node health in particular, that the drawn graph never showed.

It also found and fixed three smaller things: duplicated paragraphs in the `.install` README, a stale README in the nginx directory that still said nginx was the only supported web server, contradicting the three-profile decision everywhere else, and the mermaid diagram, which was corrected to match the manifests.

Decisions made in the same session

The session also settled four decisions that the earlier prompts had left open.

Two database servers, not one. The control plane needs a database. Tenants need databases as a product. The question was whether they are the same MariaDB instance with logical separation, or two. I chose two separate server instances. The control plane's runs on port 3306, tenants' on 3307, each with its own data directory, configuration fragment, credentials and resource limits. They can share a physical machine, and on a single-node install they will. But a bug or a resource storm in the tenant side then cannot take the control plane's own store down with it.

The web profile does not change after install. Whether a node is nginx, Apache or both is chosen once on a blank server and stays. A later migration between them is written down as a feature request, not built.

Statistics read the tenant databases directly. The statistics service reads database state through a dedicated, least-privilege, read-only account, in addition to reading web logs, and not from web logs alone.

Backups declare what they can back up. The manifest schema gained an optional `backs_up` field, separate from the hard `depends_on`. It says that a backup service can include a given service's data if that service is present and healthy, without forcing that service to exist before backups can be installed.

The session committed this as `8fc22af` at 23:12, 47 minutes after the first commit. The message was "Fix installer manifest bugs and resolve remaining ADR decisions". It touched 18 files.

It also confirmed that the plan stays out of the repository and is not mirrored into it. That comes back in the next chapter.

What the pattern was

The build-log post I published about this session says the second pass found small things and one real bug, and that the interesting part is the method. A model that has no investment in earlier reasoning has no reason to wave it through. I would add one observation from having watched it happen many times since.

None of these findings came from more careful writing. The ADR had been written carefully. Each was found by a second, independent reader who checked the document against the thing it described. The document claimed a conflict, and the plan was checked. The document claimed a dependency, and the

manifests were checked. That happens again in the next chapters, many times, and more sharply. The difference between a plan that reads well and a plan that is true is a separate step, and it costs a session each time it is taken.

The next chapter goes back to Vesta.

SOURCES

- The build-log thread, part 2, section "A fresh session checks the homework": lowendspirit.com/discussion/11288/lestra-the-vibe-coded-version-of-vestacp.
- The installer metadata at that commit: github.com/mikho/LESta/tree/8fc22af/.install.
- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `9431d04` , `8fc22af` .

Four questions with no right answer

Up to this point no session in this project had read Vesta's source. The plan was written without it, and the report had been made separately, before the first prompt. The fifth prompt, later on 2026-08-26 and again in Claude Code, was "let's continue, what's next".

What came next, according to the plan, was phase 0: a capability matrix, a threat model and a draft of the node protocol. The first of those needs the legacy code open in front of you, and the session checked the machine and found no copy of it.

Getting the source

The session asked me how to get the reference material. It offered three options: clone `outroll/vesta` locally, read-only, pinned to a commit; use the report I had already made; or postpone the capability matrix and start on something else. I answered with a fourth option of my own, which the log records as a custom combination. Clone it locally, read-only, never push it to the LESTa repository, and use the report too, since I had saved it with my notes by then.

It cloned Vesta into a scratch directory outside the repository, pinned at commit `5f4ee2ef`, the same commit the report used. Nothing from it went into LESTa.

Before relying on the report, the session checked four of its headline claims against that commit.

- `error_reporting(NULL)` : the report said 136 endpoints. The session counted 133 files under `web/api/v1/`. It is still essentially the entire API surface with error visibility turned off, and it is a different number, which is why the numbers in this book come from the recount.
- Locking: no `flock` call anywhere under `bin/`.
- `eval` on stored data: confirmed at `func/main.sh` lines 312 and 313, and 191 uses of `eval` in the tree overall.
- The arbitrary file write: `web/api/index.php` and `web/api/v1/index.php` are byte-identical, and both contain the unconstrained `v-make-tmp-file` branch.

That was the whole check, and it matters for one reason. The plan had said that every vulnerability claim must be confirmed against a pinned commit before it informs a decision. This was the first time anything in the project was held to that rule, and the report survived with one number corrected.

The threat model

The session wrote a threat model into the repository's `docs` directory. It has the five trust boundaries that the plan named: the browser to Laravel, Laravel to the node agent, the agent to the operating system, tenant to tenant, and backups and secret storage. What sets it apart from a generic document is that every boundary is tied to a specific failure that had been verified in Vesta.

The tenant-to-tenant boundary is the clearest case. In Vesta, one tenant's data is kept apart from another's by a path convention, `data/users/<user>/*.conf`, and by the assumption that every script works out correctly whose files it is touching. There is no authorization model. Combined with the switched-off error reporting from chapter 2, a bug in that assumption would be hard to even notice in production, since it would show up as an empty page and not as an error. LESta's answer is that every query scopes by ownership at the query level, using indexed fields, and that nothing trusts a tenant or account identifier sent by the client as proof of anything.

The document also lists the non-negotiable prohibitions with the reason for each, and it ends with acceptance criteria tied to the plan's review gate. Its own rule is that a capability cannot claim to satisfy the threat model until the tests that prove it exist and pass. Writing a threat model is the easy part, and later chapters are about the tests.

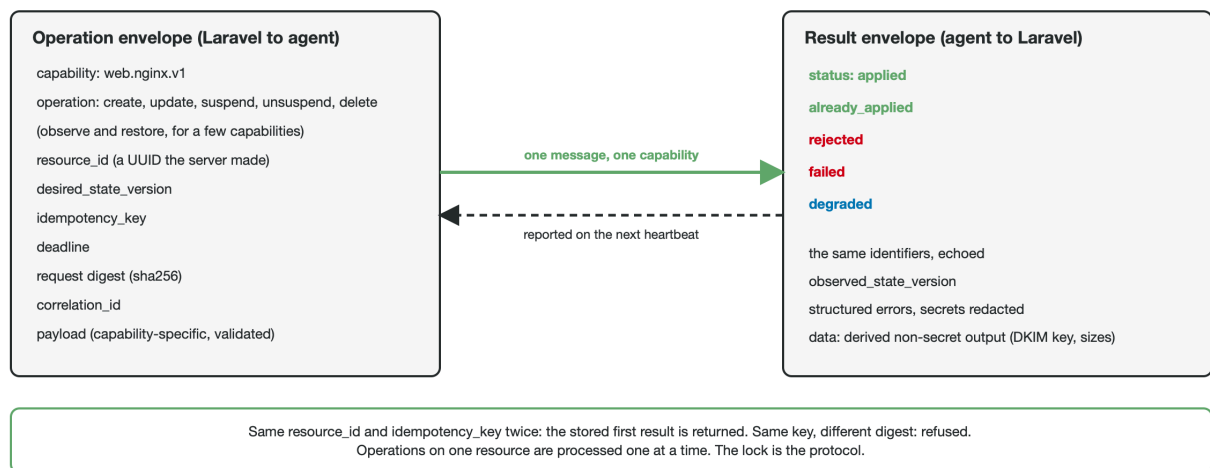
The protocol

The second document was the wire format between Laravel and the Go agent, with two JSON Schemas, one for the request envelope and one for the result.

There is no generic operation. A message names one capability, and capabilities are versioned strings such as `web.nginx.v1` or `dns.bind9.v1`. The agent rejects a message for a capability it has not registered before it looks at the payload. There is no field in the protocol shaped like "command plus arguments", so there is nowhere for a command to hide.

Every operation carries a resource ID, a desired-state version, an idempotency key, a deadline and a digest of the request. Every result echoes the same identifiers and carries one of five statuses: applied, already applied, rejected, failed or degraded.

Figure 5. One operation, one result



Two rules do most of the work. If the same pair of resource ID and idempotency key arrives twice, because a reply was lost and the control plane tried again, the agent does not do the work again. It returns the stored result of the first attempt. If the retried request's digest differs from the stored one, it refuses instead of guessing which is the real request. And all operations against one resource are strictly serialized, one at a time, whatever order the control plane sent them in.

That last rule is the direct answer to the missing locks in chapter 2. Two concurrent writers to one counter, both reading the old value and both writing the same new one, cannot happen when the protocol will not process two operations on the same resource at once. Nobody needs to remember to add a lock. The lock is the protocol.

What Vesta does that a working panel needs to preserve

The capability matrix was the big one, and the session did not write it alone. It handed the extraction to a background research agent, which read the pinned checkout resource by resource: web domain, DNS zone and record, mail domain and account, database, cron job, user, package, backup, firewall rule and IP address. For each it recorded the fields, the lifecycle verbs, the quotas, the suspension behaviour, who can see it and what happens to related resources when it changes.

The agent came back with more than a restatement of what the code does. It found gaps in the lifecycle that the report had not been scoped to look for, because the report was reading for security problems and not for product rules.

- Editing a package's quotas runs a script that pushes the new limits to every current subscriber, with a flag that skips the check that would reject the change when a subscriber is already over the new limit. Lower a package from 100 web domains to 10 and a subscriber with 40 is now 30 over, with no rejection, no warning and no suspension.
- The command that unsuspends a user reactivates every child resource unconditionally. Suspend one domain by hand for a terms-of-service problem, then suspend the whole account for a billing

problem, then resolve the billing problem, and the domain comes back online too.

- Suspending a mail account does not set a flag. It overwrites the real password hash in the Exim credentials file with the literal string `SUSPENDED`. State is stored in a credential field, so "restore the mailbox" and "restore the password" are two different operations that are easy to confuse.
- Assigning a package copies its values to the account at that moment. It does not point at the package, so editing the package changes nothing for existing subscribers unless someone propagates it, and the shipped interface does that automatically.
- Adding a web domain is not one operation. It is up to eight script calls chained together with no transaction and no rollback: the domain, an optional DNS zone, DNS records for each alias, an optional mail domain, Let's Encrypt scheduling, an uploaded SSL certificate, a statistics configuration and a statistics user. If the fourth step fails, the domain exists with no DNS zone, and nothing cleans it up. Turning on Let's Encrypt creates a system cron job for renewal that does not count against the account's cron quota and does not appear in the account's cron list.

The session adopted most of the corrections directly. A live reference to a package and not a copy. An explicit suspended flag. Transactions. A real relational database removes the technical reason Vesta did any of those things, so there was nothing to decide.

Four questions

Four things did have no correct answer, and the session put them to me as numbered questions with a recommendation each. They are product policy, and a wrong guess would have meant redoing work.

Q9. A package edit would leave existing subscribers over the new, lower quota. What should happen? Reject the edit outright, recommended. Or allow it and flag the affected accounts. Or allow it silently, which is what Vesta does. I chose to reject: an administrator cannot save a package edit that would put any current subscriber over the new limit. They have to raise the limit or fix the accounts first.

Q10. When an account is unsuspended, do resources that were individually suspended beforehand stay suspended? Preserve their prior state, recommended, against always reactivating everything. I chose to preserve it. Unsuspending an account only reactivates what was suspended because of that cascade.

Q11. Does deleting a suspended resource need an explicit unsuspend first? Automatic unsuspend and delete as one confirmed action, recommended, matching Vesta's real behaviour. I chose that.

Q12. May a provider administrator look at a tenant's resources without impersonating them? Add a read-only support view, separate from and logged separately from full impersonation, recommended, against impersonation only. I chose the support view. Vesta has no such concept. An administrator who needs to look at a tenant's data has to log in as that tenant first, and there is no lighter way to just look.

I took the recommended option on all four. It is worth being exact about what that meant. The session did not make these four decisions for me. It separated the questions that were decisions from the questions that were bugs, and treated them differently. "Packages should be a live reference" was a correction, since a real database makes the old approach pointless. what should happen to a subscriber who is already over a newly lowered quota was a policy, and it came to me. A lot of AI-assisted work skips that distinction and quietly picks whichever answer sounds most reasonable. The read-only support view exists in LESTa only because that question was asked.

The vault takes over

The sixth prompt asked for the documentation to move out of the git repository and into my Obsidian vault, and for the change log to be updated after every prompt from then on.

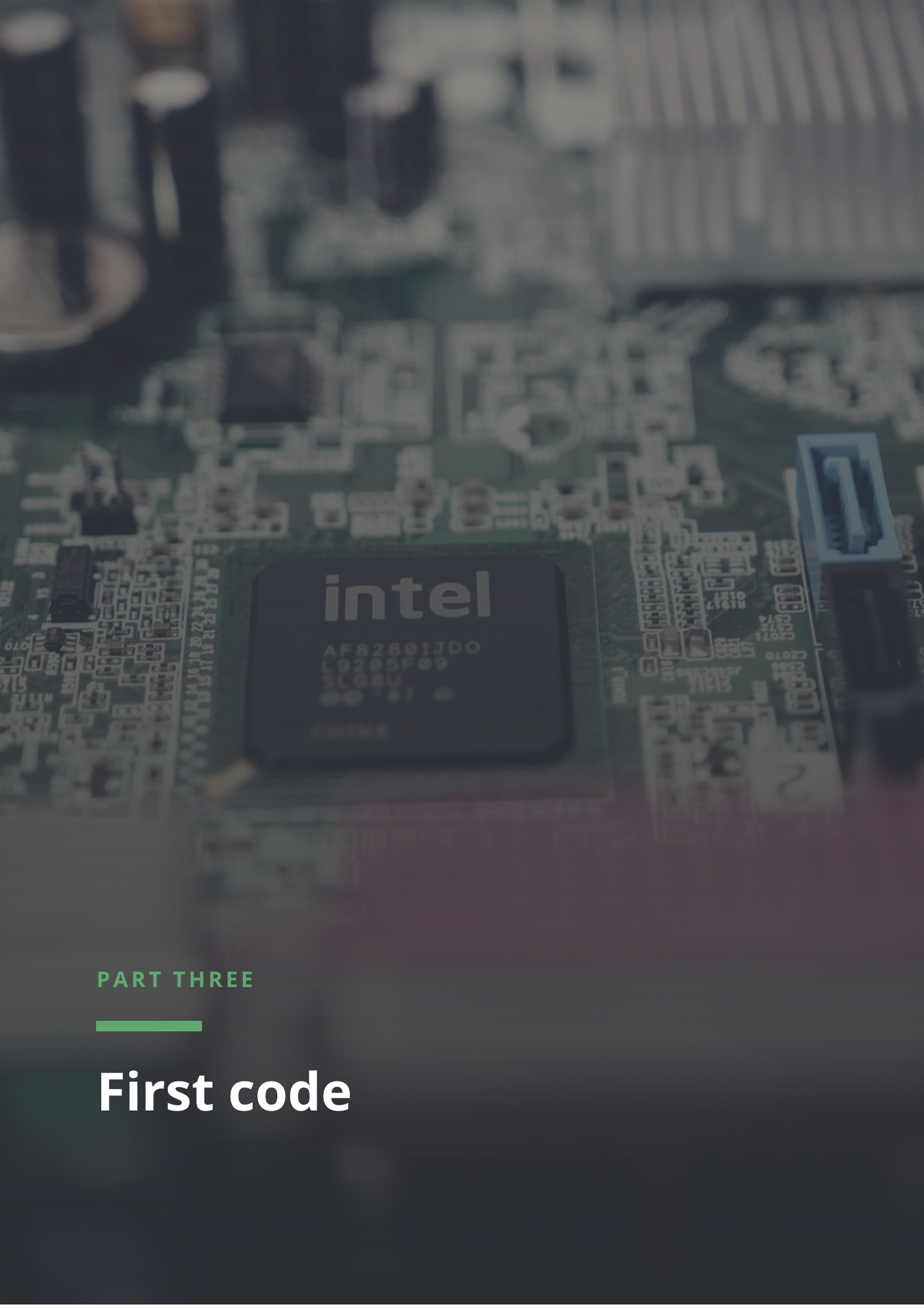
The session checked the scope first, since some of what is in `docs` is not prose. The ADR stays in the repository, because it was already pushed and ADRs live with the code they explain. The two JSON Schemas stay, because they are machine-checked contract files that future Go and PHP code will be tested against. The `.install` READMEs stay, because they are the installer contract and not documentation about it. I agreed to each of those in three short questions, Q5, Q6 and Q7, and took the recommended option every time.

The three prose documents, the threat model, the protocol narrative and the capability matrix, moved to my notes. They were deleted from the repository. They had never been committed, so nothing needed rewriting in git.

That is why this book can be written. The vault now holds the plan, every prompt, every decision and every question I was asked, with exact options and my answer. A session that starts cold can read the log and continue. So can a person, three weeks later.

SOURCES

- The build-log thread, part 3: lowendspirit.com/discussion/11288/lestra-the-vibe-coded-version-of-vestacp.
- Vesta at the pinned commit: github.com/outroll/vesta, commit `5f4ee2ef`.



PART THREE

First code

The session that refused

The first real code in LESTa was written on the morning of 2026-08-27, and for a while it looked as if it would not be written at all.

The approval that did nothing

By then I had approved a plan for the first phase of the build: the relational foundation of the control plane. Accounts, memberships, roles, permissions, packages, nodes, the audit log, idempotency receipts, and the table that provisioning jobs move through. The plan had come out of a session in plan mode, which is a state in Claude Code where the tool itself will not let the model touch a file until someone exits the mode. It is enforced by the harness, not by the model's own good manners.

I answered the plan with four words: "Yes, and auto-accept". Auto-accept meant that the session should go ahead without stopping to ask me about every step, and the plan was to be saved to the vault and the log kept up after every prompt.

Implementation did not start. The plan mode was still active when my approval arrived, and it stayed active. The session handed the work to a background agent. The agent looked at the state it was in and correctly refused to write to the repository, approval or no approval.

What it did instead was more useful than I would have guessed. It produced a resolved, execution-ready version of the plan, checked against the real codebase, with the exact code for every file it would eventually write. And it listed ten places where the original plan had been silent or had waved a hand. Which models implement the marker interface for provider-admin access, which the plan only described in prose. How the "stop impersonating" action finds the session to go back to. Two configuration keys the plan named but never put a number on. A named exception for the quota-rejection decision from chapter 6. None of the ten changed what the plan intended. All of them were gaps that a plan, however carefully written, leaves for the first person who has to type out every line.

I read all ten and agreed with each of them. The session asked to leave plan mode a second time, and this time it took effect. A second agent then wrote the code for real: 53 new files and 4 changed ones. A schema of twelve migrations. The policies. The mechanism that sends provisioning work to a queue after a database commit, and never before.

The log notes that the second phase's plan went through the same sequence, and records it in one line: two rejections and then an approval.

Three small design decisions

Three of the design choices in that phase are worth writing down, because later chapters stand on them.

One table, not two, for memberships. A membership has an account or it does not. A null account means a platform-level grant, such as provider administrator. A non-null account means an account-level one, such as owner. Two tables would have been two things to keep in sync.

One table absorbs what the plan called the outbox. The plan named "outbox messages" and "provisioning operations" separately. In the built schema, a single provisioning operation is one row that moves through a life: pending, dispatched, applied, failed. The log flags this as a deliberate merge and says when to reopen it.

And no blanket administrator bypass. In Laravel, a gate can be told to let one kind of user through every check. LESTa does not do that. Platform infrastructure, meaning nodes, packages and roles, is open to provider administrators through a marker on the model. Tenant-owned things, meaning accounts and memberships, are not. A provider administrator cannot touch a tenant's account at all, except through the explicit, audited impersonation action. That is how the read-only support view from chapter 6 is enforced: it is the only way in.

Checking the work

The session did not rely on the agent's report of its own work. It ran the checks again itself.

`migrate:fresh` on an empty database: 12 migrations, clean. Pint, the code style tool: clean. PHPStan, a static analyzer, at level 7: no errors. The full test suite: 63 tests, 197 assertions, all passing.

Then it opened two of the files and read them: the authorization provider and the account policy. It wanted to confirm one thing above the others. A provider administrator with no membership in an account must be denied by the account's policy, with no bypass leaking onto it. The `Account` model does not carry the marker, and the check held.

Four bugs that only running found

Four defects surfaced when the code ran. Each one was caught by a machine and not by a person reading. None of them came out of the plan.

A required column left empty. `idempotency_receipts.correlation_id` must have a value, because every row should be traceable to what caused it. The plan's own code for creating a receipt never set it. The first real call would have thrown a database error.

A query that MariaDB accepts and SQLite refuses. The quota check from chapter 6, reject a package edit that leaves a subscriber over the limit, was written in a standard Eloquent form using `withCount` and `having`. SQLite refuses a `HAVING` clause on a query with no `GROUP BY`. MariaDB, the production database, allows it. Local development ran on SQLite. The bug appeared only because a test actually ran the path. It was rewritten with `whereHas` and a count, which gives identical results on both.

A type that the application never produces. A result object typed a timestamp as `Carbon`. This app tells Laravel to use `CarbonImmutable` for every date, so `now()` never returned a `Carbon`. A type hint written from general Laravel habit could not have accepted what the rest of the app handed it. It was retyped to the interface both share.

A test that failed on chance. Two factory calls in a cascade-suspension test each drew a capability from the same pool of three, and sometimes drew the same one against a unique database constraint. It failed once, on the first run, and then not again. It was pinned to fixed values.

None of the four is an `eval` on stored data, a missing lock or an unconstrained file write. The threat modelling of the previous chapters bought exactly what it was meant to buy: the first real code shipped with ordinary, boring defects instead of the category of defect the project began with.

The commit that was not what it said

The work went out as three commits, so that the Phase 1 message would describe only Phase 1. The pointer to the vault in the repository's `CLAUDE.md` went first, then the node protocol schemas, then the phase itself. They landed at 09:08, 09:09 and 09:09 on 2026-08-27.

A file was missing. `bootstrap/providers.php` had been correctly changed in the working tree, to register the two service providers that phase needed. It was never staged. The session's checks all passed, because they ran against the files on disk, and the files on disk had the fix.

It was caught two minutes later, at 09:11, by running `git status` again after the commit. One file was still modified. The follow-up commit, "Register the Phase 1 service providers", went out. Then the session did one more thing. It ran the whole verification again, on the tree that now matched what had actually been pushed, and not on a tree that happened to contain the fix all along. The lesson written in the log is exact: passing tests locally prove that the working tree is correct, and they do not by themselves prove that the commit is, and the two are the same claim only once the working tree and HEAD match.

Later I pasted a GitHub Actions log into the session. CI had failed with seven tests, two error patterns: an authorization exception on the node and package tests, and a container binding error on the provisioning dispatch tests. It was exactly the bug above. Without the providers registered, the scoped admin bypass never runs, so an administrator is denied like anyone else, and nothing is bound to the provisioner interface. The session checked with `gh run list` and `gh run view`. The run for the first commit had failed with those seven tests and those two errors. The next run, for the follow-up, was green. The pasted log was CI arriving late with the report of a bug that had already been fixed.

The second phase

The second phase followed the same day, and it shows what the working rhythm was going to be.

It was the web hosting vertical slice: a `WebDomain` resource with a full life, a table of IP allocations that phase 1 had deliberately left out for this moment, a provisioning contract, and the first interface. It went into plan mode again. Before designing anything it looked at the front end and found real gaps. The app used one form component everywhere. No reusable confirmation dialog existed, so every destructive action carried its own. And neither a `shadcn` table and pagination component nor the Pest browser plugin was installed.

Both of those were decisions about dependencies, and the project's rule was that dependencies do not change without my approval. So the session asked me, and I answered.

Add the browser testing plugin now? Yes, the recommended option. Add the `shadcn` table and pagination, or build the list without them? I chose to build without them, with no new frontend packages, which was not the recommended option. Then a question that came out of the design itself: when a package has no quota row at all for web domains, should creation be unlimited or blocked until a limit is set? I chose blocked. A row with a null value still means unlimited, but no row means none. That is the stricter reading of deny by default, and it costs something: every test that creates a web domain has to attach a limit first. The log records it as a deliberate constraint and not a slip.

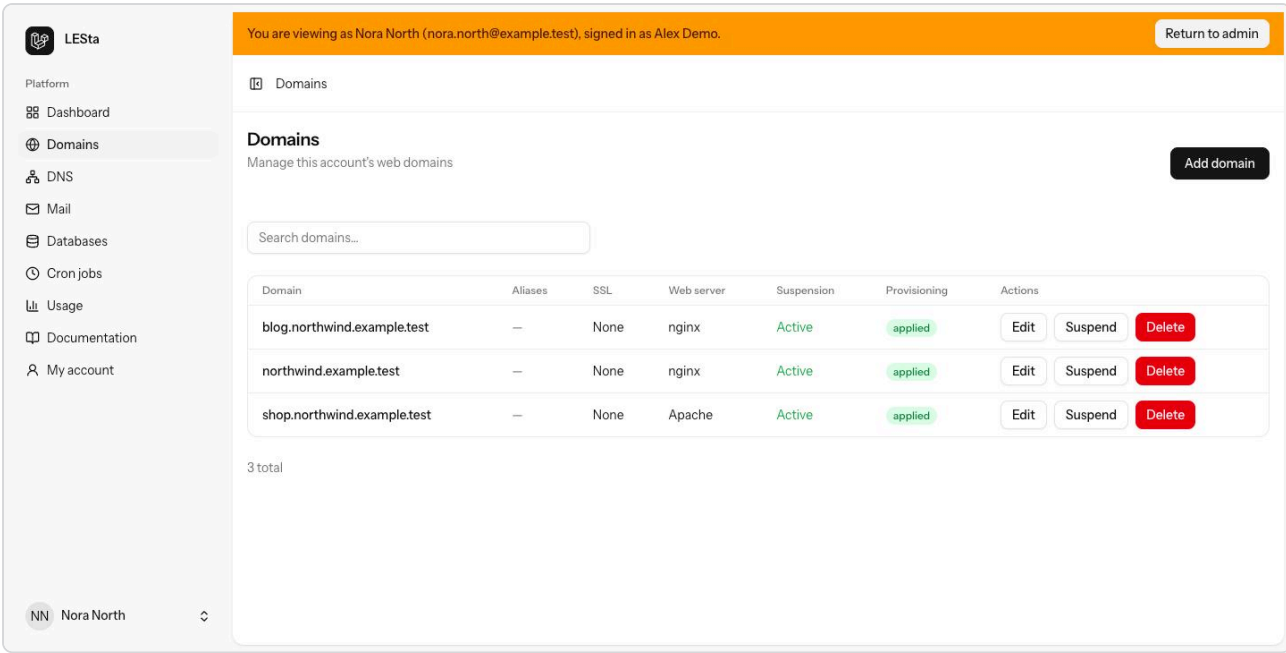
The design agent also settled a tension without asking. The foreign keys from web domains to accounts, nodes and IP allocations use `restrictOnDelete`, not cascade. A database cascade would let someone delete an account and silently wipe its domains, with no audit entry and no provisioning delete. That is the unguarded fan-out from chapter 6. With restrict, the account deletion action has to delete each domain through the real, audited action first, and if a future change forgets to, the database throws instead of leaving orphans.

The plan was approved at 11:16 and implemented by 16:32. Independent checks: 18 migrations clean, Pint clean, PHPStan clean, the first production frontend build the project had needed, and 114 tests with 334 assertions, up from 63. It came with five small deviations from the plan's text, each reviewed. One was a Laravel limit: `MorphMany` does not support `latestOfMany()`, so it used `morphOne`. One was a line in `CLAUDE.md` about the PHP version that changed from 8.5 to 8.4 on its own, because a Boost hook corrected it to the runtime that was actually installed.

The two browser tests existed but could not run, because the plugin needs Playwright. I approved installing it. When they ran, they found two bugs, both in the tests. One clicked "Suspend" twice on a page whose suspend button has no confirmation, so the second click undid the first. The other clicked the text "Delete domain", which was both a heading and a button, and hit the heading. A screenshot that the plugin saves automatically on failure made the second one visible at once: the dialog was plainly not open. The log's own comment is that neither bug was visible from reading the plan, the component or the test. They were visible from watching the interaction fail.

It went out as `876b6d9` at 16:43 on 2026-08-27: 60 files, 4,662 lines added. The session re-ran every check on the committed state, as after the missing-file episode, and then watched the GitHub Actions run to the end with `gh run watch` instead of assuming a push means a passing build. It was green in 50 seconds.

I asked for one more thing that afternoon: that every question I had been asked be written down in one place, with the options and my answer. That audit found the early questions recorded only as narrative, and produced the Q&A file that this book quotes. At that point it held sixteen questions. Twelve were answered with the recommended option. Four were answered with a stricter or more deliberately scoped one: both Ubuntu releases, both ACME modes, the hand-built list page, and blocking web domains until a quota exists. The log notes that this is a pattern worth having on record. In those four cases, at least, my disagreement with the recommendation was in the direction of more work now for less risk later.



The domains list of phase 2, shown here as it looks today for a demo account. The provisioning column is the honest state of the outbox row.

What this looks like from the outside

At the end of 2026-08-27 there were two phases in the repository, a green CI, and 114 tests. There was still no server involved. Everything so far was Laravel talking to a fake. The plan's next line was the real thing, the Go agent, and that is where the language changes.

SOURCES

- The build-log thread, part 4: lowendspirit.com/discussion/11288/lestra-the-vibe-coded-version-of-vestacp.
- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `5d82808` , `876b6d9` , `4966c6e` , `474d666` , `364c769` .

Go, and a real nginx

Everything so far ran on a laptop against a fake. On the evening of 2026-08-27 the project met a real web server for the first time, and the first thing it needed was a language it had not used.

A machine with no Go

The thirteenth prompt was "lets continue". The session checked the machine before assuming anything, and found no Go, no Multipass, no Vagrant, no Docker and no Lima. The plan's third phase was the node agent, and ADR 0002 had set its exit bar: the fake and the real adapter must pass the same contract suite, and it must be tested on disposable Ubuntu systems.

Two questions came first, since a wrong guess would have meant redoing infrastructure and not only code.

Where should the agent's code live? A subdirectory of the same repository, recommended, or a separate one. I chose the subdirectory. One git history, and the protocol schemas and the Go code versioned together. A separate repository would have added coordination between two repositories for a one-person project.

How should "disposable Ubuntu" work? Multipass locally and GitHub's Ubuntu runners in CI, recommended. Or Docker containers with systemd, which is faster but a worse fit for testing real service reloads. Or cloud VMs, which cost money on every run and need credentials. I chose the first.

The session scoped the phase narrowly and wrote down what it was leaving out. It would implement all six protocol operations for `web.nginx.v1` and nothing else. No network transport between Laravel and an agent, no Apache, no HTTPS. One item on the "out" list is worth quoting for what it says about how the project thinks: `ssl.mode` would be read and stored but never acted on, because acting on it needed certificates that did not exist yet, and would make the agent's own `nginx -t` check fail against itself.

It also had to reconcile two documents that disagreed. ADR 0002 said generation history lives under a shared `/var/lib/lesta/generations/` root. The nginx manifest listed the directories the service owns, and that path was not among them. The plan put the history under nginx's own directory instead, on the reasoning that the manifest is the concrete, binding boundary and the ADR's wording is the more general one. It also scoped generations per resource, one domain at a time, not per node, since a node-wide counter would tie unrelated domains' rollback histories together.

The plan was approved at 21:02 that evening. Go 1.27.0 and nginx 1.31.4 were installed through Homebrew on the laptop, neither having been there that morning, so the real tests could be run and checked locally and not only on a future VM.

What it built, and what it proved

Two hours later the session had a Go module with the protocol types and a test that checks them against the JSON Schemas, a store for idempotency and one for generations that any service can use, the shared contract test suite, a fake capability that mirrors the PHP fake formula for formula, and a real nginx capability implementing all six operations through one shared apply pipeline. The apply pipeline stages a new generation of the configuration, validates it, activates it atomically, reloads nginx, checks health, and rolls back if any step fails.

The session ran the tests itself and looked at what happened, not only at whether they passed. A real HTTP request against a real disposable nginx process on an ephemeral port returned the body of the vhost that had just been created. Suspending the domain swapped in the maintenance page, and the next request returned it. Unsuspending brought back the exact original body. Deleting removed the file and nginx kept serving a second vhost that was still alive.

The three failure cases were forced with a controlled reload-failure wrapper, not a mock. A reload that fails during `create` reports `failed`, never `degraded`. An `update` whose rollback also fails to reload reports `failed`. An `update` whose rollback succeeds reports `degraded`, and a follow-up HTTP request confirmed the original vhost was still being served.

It found two things that the implementing agent's own report had not mentioned. The new CI job pointed a Go cache at `agent/go.sum`, but the module has no external dependencies and so no such file, and the behaviour of a cache step on a missing file was unclear. It was changed to an explicit "nothing to cache". And the agent had created a settings file for its own permissions, `.claude/settings.json`, which had never existed in the repository. Deleting it was blocked by Claude Code's own permission checks, a guard against an agent editing its own settings. The session did not work around the block. It left the file on disk and made sure it never went into a commit.

The result went out as `8db3cad` at 23:12, 30 files and 3,447 lines. The session ran every check again on the committed state and watched the GitHub run. Both jobs were green, including the new `agent` job on its first ever run, which installs real nginx on a GitHub runner and executes the full contract suite there.

The token limit, for the first time in the log

Twenty-six minutes later, at 23:38, I sent a message that sits in the decision log as the fourteenth prompt: "my session tokens are almost over, can we automatically continue in 2 hours". It is the first time the token limit shows up in the record, and it is a plain statement of fact.

The session looked into whether a scheduled cloud routine could pick the work up when the limit reset, and found that it could not do it well. A cloud routine reads only what is in the repository it clones. The vault, which holds the plan, the log, every phase plan and every question and answer, lives on local synced storage and not in the repository, by the decision I had made a day earlier and chapter 6 described. A cloud routine would have seen the code and the ADR and nothing of why.

That was Q19. Just resume this local session later, recommended, or create the routine anyway with a condensed summary pasted in. I chose to resume. The log adds a sentence to it that is worth keeping: the vault gives continuity across separate sessions and even across different tools on the same machine, since both can read local files, but none at all to a cloud-sandboxed agent.

The installer

The session resumed at 08:49 the next morning with "continue". It checked that the tree was as it had left it, updated the backlog file that had gone stale, and asked what to do next. Three options were available: the real nginx installer, the Apache capability with the combined profile, or DNS as the next service slice. I chose the installer, the recommendation, since the ADR had said an executable installer would be written only after the capability tests passed, and that condition was now met.

A planning agent designed one script, `.install/services/nginx/install.sh`, and not a general runner. It handles the base layer, a firewall baseline and node health as named phases inside the one script, because none of them had any install logic yet. The plan put them before nginx. Node health was later moved to run after it, for a reason the security section below explains. The session read every manifest, the installer contract, the schema and the agent code again and confirmed that the claims in the design held.

Two questions came out of the design.

How does the include line get into `nginx.conf` ? The Go agent needs one particular line in the main nginx configuration to work at all. But the manifest marks that file as both read-only and refused, meaning the installer must never write to it. I chose the recommended option: a manual prerequisite. The installer reads the file, and if the line is missing it fails its preflight with the exact line to add by hand. The other option was a separate opt-in subcommand that adds it. That option came back later, as chapter 15 shows.

How does the nginx package satisfy the rule that every download carries a pinned version and checksum? The recommended option was to vendor a pinned `.deb` for each Ubuntu release and verify it offline. I chose the lighter one: a live `apt-get install` from Ubuntu's own archive, verified by APT's signature chain, with the installed version and checksum recorded afterwards in the run log. It kept the phase to the installer and out of release engineering. That, too, came back, in the same chapter.

A security hole found in review

The background agent implemented the plan, and the session reviewed the result. It found a serious problem that the agent's own report did not mention.

In its first draft, the shipped production agent read `LESTA_NGINX_*` environment variables to override its built-in paths. One of them was `LESTA_NGINX_BINARY`, the path that the agent passes straight to `exec.Command()`. Anything that could set that process's environment could have made a privileged component run any binary. The reason given was harmless: it let the installer's self-test point the shipped binary at a disposable nginx instead of the real host.

The session reverted `main.go` to exactly the state it had after phase 3, with fixed paths and no sensitivity to the environment at all. It deleted the test that had been written for the removed feature. It changed the self-test to run after nginx is installed. It now creates a throwaway resource named `selftest.lesa.invalid` against the real system nginx using the unmodified production binary, checks for `applied`, and deletes it.

It also rebuilt the vendored agent binary with `-trimpath` and stripped symbols before it became the first binary file ever committed to the repository, and it added the CI job that the plan had listed and the agent had not created.

The macOS trap

One test problem is worth telling because it looks worse than it was. Running the script under macOS's plain `sh`, with BSD `sed`, produced JSON with every key and value empty. The session installed ShellCheck and Multipass to test properly. Multipass's own installer wanted an interactive `sudo` password prompt, which a background shell cannot answer, so it never finished, and that stayed a known gap because CI covers real Ubuntu. Instead the session installed GNU `sed` and used macOS's `/bin/dash` with a shim, to reproduce the combination Ubuntu actually has. The output was correct. The bug was BSD `sed` mishandling one escaping pass, and not a defect in the script. No code changed, only the toolchain used to test it.

Five pushes to green

The installer went out as `a4d77d0` at 12:45 on 2026-08-28, and CI ran it on real Ubuntu for the first time. It took five pushes to go green, and four of the five fixed real problems.

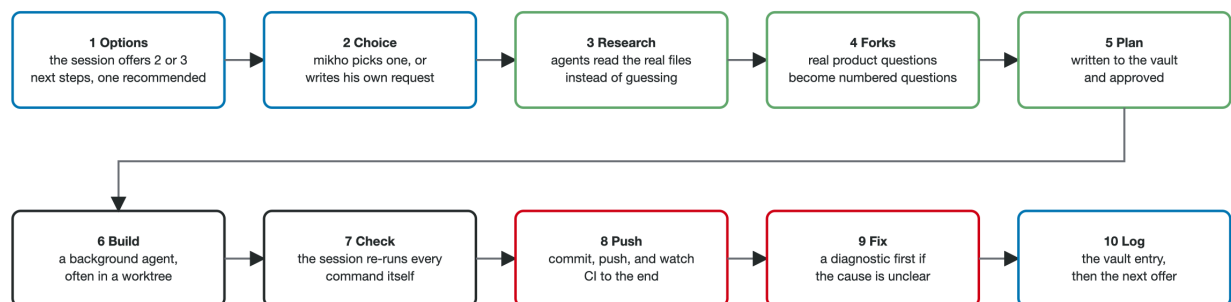
- `87ce1cc` at 12:48. ShellCheck failed, because its most verbose severity flag is a level and not a filter, and it flagged "not following" notices for sourced files. Fixed by excluding that one code.
- `0678bf3` at 12:50. The dry run failed real preflight on two correct rejections. GitHub's runner image ships with Apache installed, which is a conflicting package, and its nginx has no include line. The CI job now does by hand the two things the README says an operator must do.
- `cbe72f4`, then `0422712` at 12:52 and 13:43. The idempotent rerun failed with `port_occupied` on port 80, because after the first successful apply nginx itself holds the port, and the check did not tell its own nginx from a stranger. The first fix compared a single result and broke on the runner's dual-stack listener, which produces two lines. The second walked every line and required each owner to be nginx.
- `0e0d2e7` at 13:45. The rerun then failed in the self-test with `resource_already_exists`. Deleting a resource does not erase its generation history. It records a terminal "deleted" generation, since a deleted domain's identifier must never be reused, and the self-test used one fixed UUID on every run. It now generates a fresh one per run.

After the last push, CI ran green on the first attempt: ShellCheck, runner preparation, a real dry run, a real apply that changed an actual Ubuntu host, a real rerun, and the agent's real-capability suite against the state the installer had produced.

The log's comment on it is one I would keep: four of the five commits fixed real bugs that only appeared when the installer ran a full apply and a rerun on a genuine host. No amount of local ShellCheck could have found them. That is the reason the session insists on watching the real CI run instead of trusting a green implementation report. From this point the project has a working, tested path from an empty Ubuntu server to a running web capability.

Figure 8. Anatomy of a phase

What one round of the project looked like, from the session's offer to the next one.



After step 10 the loop starts again at step 1.

Steps 3 and 7 are where most bugs were found. Chapters 5, 7, 8 and 16 show them at work.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `8db3cad` , `0e0d2e7` , `a4d77d0` , `87ce1cc` , `0678bf3` , `cbe72f4` , `0422712` .

DNS, and an hour of AppArmor

DNS was the next service, and it was built the way web hosting had been. That was the point of having a pattern. A control-plane model and a fake provisioner first, then the interface, then the real Go capability, then the installer, four phases in that order, each one proven before the next started.

It took about 22 hours of wall clock, from 14:48 on 2026-08-28 to 12:32 the next day, and most of the interesting things in it happened in the last hour.

Phase 5: the model

The sixteenth prompt was "continue". The session offered two choices for what came next, DNS with BIND as the recommended one, or Apache and the combined profile. I took DNS. ADR 0002's build sequence named DNS as the next full slice after the installer, and the session had been asked to follow it.

Two research agents ran side by side. One mapped the web hosting slice in detail so that DNS could copy its shape. The other read everything the ADR said about DNS: the capability string was already settled as `dns.bind9.v1`, the ports were 53 over TCP and UDP, and `/etc/bind/named.conf` was marked read-only and refused, exactly as `nginx.conf` had been. A planning agent designed the phase. It noticed that DNS records need a different suspension cascade from web domains. A domain is provisioned on its own, so suspending an account calls the suspend action for each domain. A DNS record is never provisioned on its own, only as part of its zone, so it copies the pattern used for a node's capabilities: flip the flag directly on the children and provision once, on the parent.

There was one small question. Should DNS records get a public identifier now, on an empty table, or wait until the interface needed one? I took the recommended option, add it now. It was cheap on an empty table and would avoid a backfill later.

The agent implemented it and the session reviewed it. It found one real problem the agent had not: a unique index on `dns_records` was written across a `text` column, with a comment saying that this project's only database driver was SQLite. That was true of the local `.env` and untrue of the target. ADR 0002 had settled MariaDB 11.4 on the first day. A full unique index on a text column runs into InnoDB's key length limit the day it is created on real MariaDB, and works on SQLite only because SQLite does not enforce that limit. The column became a bounded string of 512 characters, which fits realistic record values including long DKIM and SPF text records. Same bug class as chapter 7's `HAVING` query: something that works on the development database and would have failed on the production one.

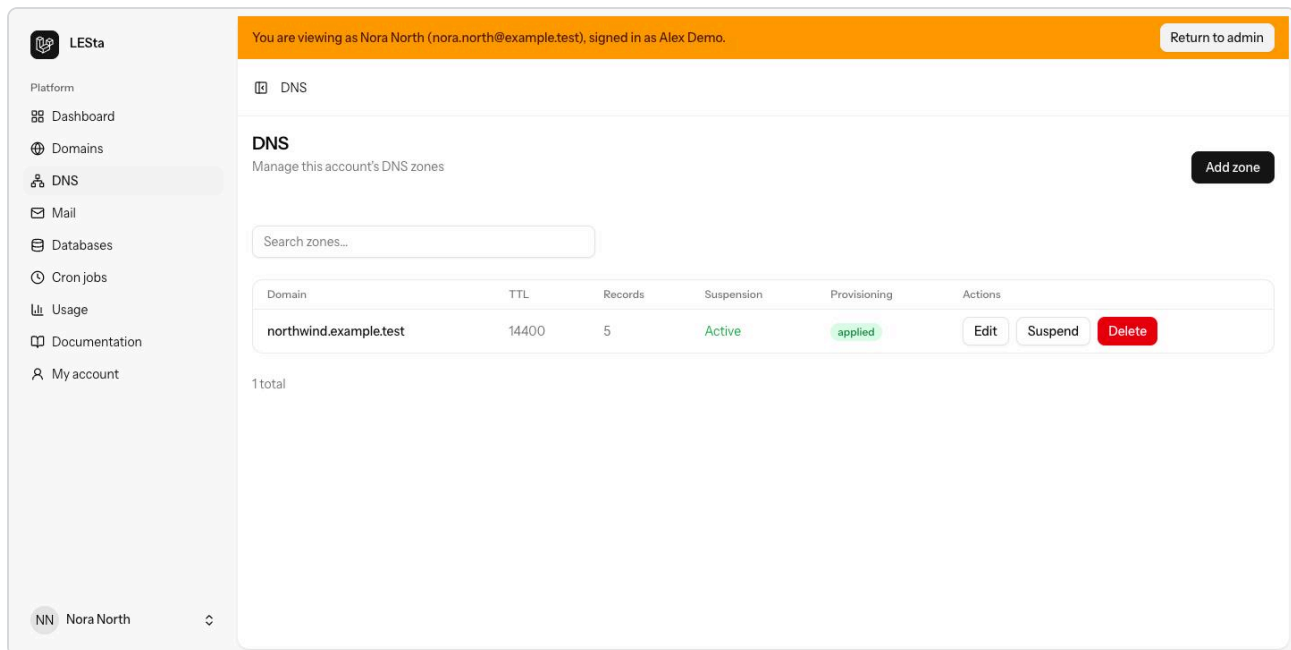
Phase 6: the interface, and a dialog that would not close

Phase 6 needed no fresh question about what came next. The DNS sequence was already fixed by precedent and written into the backlog. The design made four decisions without asking me, because each followed from something already approved: two controllers, one for zones with full CRUD and one for records nested under a zone; record editing built now, since the edit action and its policy ability had shipped in phase 5; two new validation rules, one for zone-relative names that allow underscores, `@` and `*`, and one for the hostnames that CNAME, NS, PTR and MX records point at; and TXT, SRV and CAA values validated only as bounded opaque strings.

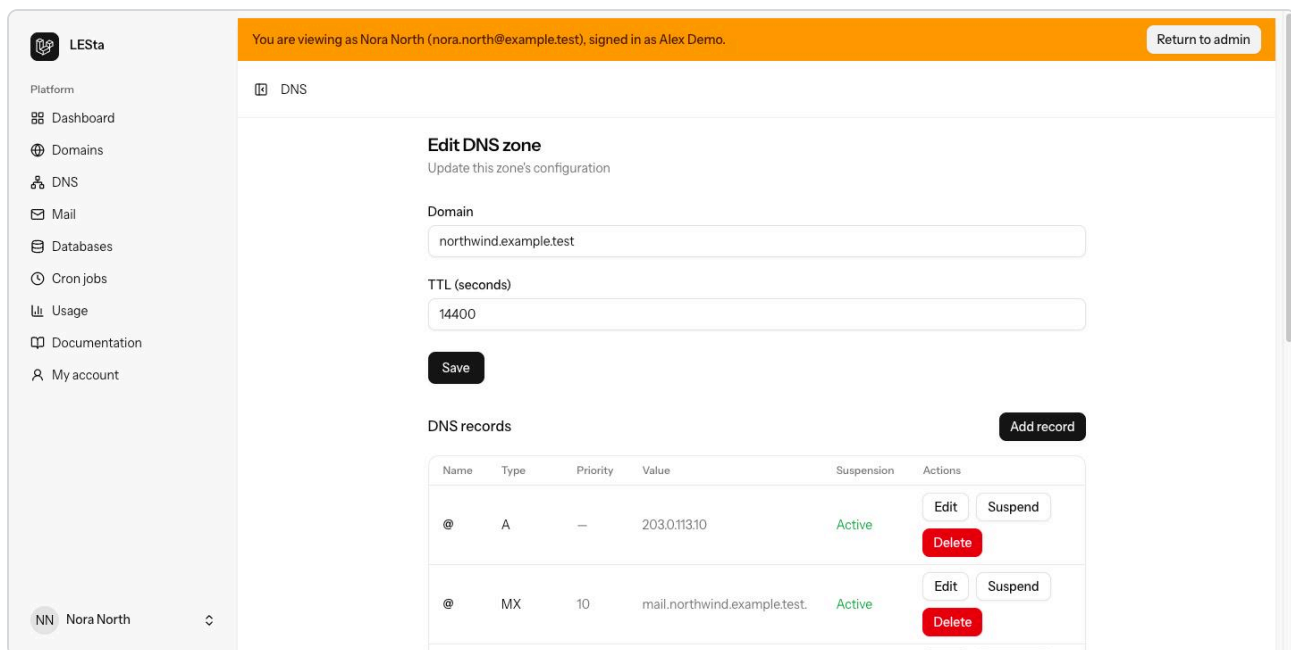
The session re-checked the two most load-bearing claims in the plan before adopting them. One was that a shared TypeScript types file would break if the new DNS types redeclared two types that existed already. It read the file and confirmed it. The other was a plan-agent diagnosis that nested routes would fail scoped binding because of a pluralization mismatch. The session traced Laravel's own binding code by hand and found the real reason was simpler. Laravel only scopes nested bindings when a route calls `scopeBindings()`, and these routes did not. The fix was the same either way, an explicit check that the record belongs to the zone in the URL, but the session recorded the correct reason and not the plausible one.

The implementing agent found a bug that nobody had anticipated. Inertia's form component keeps the page state by default after a submit. On the DNS edit page, several actions redirect back to the same page, so the page never remounts and a dialog with uncontrolled state stays open forever. Each of the three affected dialogs got explicit open state, closed from the form's success callback. The delete dialogs did not need it, because their redirect either goes to a different page or removes the row that contains the dialog, and the log explains that reasoning too.

The session read the two most safety-critical tests to check that they were not vacuous. The test for mismatched zone and record returns 404 uses two zones under the same account, which matters, since a plain tenant-isolation test would not catch the bug. It needs the ownership guard specifically. The commit went out as `a6c77a4` at 16:03 with 20 files and 2,191 lines. All three CI jobs were green on the first attempt.



The DNS zone list from phase 6.



Zone edit with inline record management, the interface phase 6 built.

Phase 7: BIND9 in Go

Before designing anything for the real capability, the session installed BIND 9.20.27 locally, matching what had been done for Go and nginx, and tested three behaviours on the real thing. `include "path/*.conf";` accepts globs, like nginx. A zone's NS record needs a glue record only when the nameserver is inside the zone itself, confirmed with two hand-written zone files, one refused and one accepted. And `named-checkconf -z` checks the whole thing in one call, the config structure and the content of every zone file it references, which was confirmed by feeding it a deliberately corrupted zone file.

The planning agent then found a bug in its own reasoning before it presented the plan. It had proposed mirroring nginx's rollback exactly: on failure, stage the previous generation's stored content again as a new generation. That works for nginx, whose vhost fragments are self-contained. It would not work for BIND. A zone stanza's `file` directive points at a specific generation's data path, so a rollback that copied the old stanza would embed a stale generation number in the live config. The session did not accept this on faith. It traced the scenario by hand, generation by generation, through repeated rollbacks, and found that enough of them would widen the gap between the referenced old generation and the ever-rising counter until it passed the generation store's retention window and the file vanished. The fix was that each new generation renders its own stanza and its own zone file, and refers to nothing older.

Two more design findings. TXT and CAA values are free-form strings up to 2,048 characters, a decision from phase 6, and can contain a quote or a backslash that would break the zone file if written raw, so they need an explicit escaping step that nginx's narrower fields never needed. And a node that no longer serves a zone answers REFUSED, not NXDOMAIN, since NXDOMAIN is a claim made by the parent zone's servers and not something a node can produce itself.

One question came out of it: should the existing agent binary handle both nginx and BIND by dispatching on the capability string, or should DNS get its own binary? One binary, recommended, and I agreed. It matches the ADR's picture of "the agent" as one long-lived process per node, and the security rule from chapter 8, fixed paths and no environment sensitivity, applies to both.

The implementation found something new. BIND's `include "dir/*.conf";` fails outright when the glob matches zero files, unlike nginx's forgiving equivalent. On a fresh node with no zones the include would break `named`. The session re-verified it on the local BIND install before accepting the fix, a placeholder fragment that is never removed and keeps the glob non-empty. And it wrote a note for phase 8: the installer must seed that placeholder before `named` starts for the first time. The commit was `d850da8`, at 21:00 on 2026-08-28.

Phase 8: the installer, and the firewall bug that had been there since phase 4

The real bind9 installer started on the morning of 2026-08-29 with a hit against a usage limit. The first attempt to start the design agent ran into the session's rate limit in the middle of the call and stopped with an error, without producing anything. It was retried after the limit reset, and nothing was lost.

Then the session read the shipped nginx installer again, and found something that had been sitting there since phase 4. `bootstrap_firewall_baseline` renders the entire nftables table from scratch on every run, from whichever installer happens to be running. `nft -f` on a full table block replaces what is there. It does not merge. A bind9 installer that copied the pattern with only its own two ports would silently close ports 80 and 443 on any node already serving web pages. On a node that serves both, which is the ordinary case for one server, DNS installation would have taken the sites offline.

A planning agent designed the fix, and in doing so caught a flaw in the approach the session itself had first suggested. The session had proposed deriving the firewall's port list as a union over every service manifest in the repository. The planning agent pointed out why that is wrong. It would open MariaDB's and mail's ports on every node whether or not those services were ever installed there, which contradicts deny-by-default, and it would open new ports on an existing node after an unrelated `git pull`. The corrected design was a small persistent registry on each node, one fragment file per service actually installed there, joined when the table is rendered. It depends on nothing but what is installed, in any order.

The session verified the refactor of the nginx installer function by function against the original, ran `--version` on both trees under the toolchain shim from chapter 8 and got byte-identical output, and tested the firewall union with `nft` and `systemctl` stubbed. It committed the installer as `5129e97` at 11:36. What followed was seventeen more commits in 56 minutes.

The hour

CI runs on real Ubuntu, and every push showed something. The record of that hour is the commit list, and it reads like a conversation with an operating system.

11:39, 88f901e. ShellCheck on the runner is version 0.9.0. The local one is 0.11.0. A guard-clause shape passed locally and failed in CI, because the newer version quietly stops warning when the last command is `continue`. Rewritten as an explicit `if`.

11:43, e6b195d. The port check flagged port 53 as in use. It was, by systemd-resolved's stub resolver, which listens on `127.0.0.53:53` on every Ubuntu 24.04 and 26.04 node. A specific-address bind and a wildcard bind on the same port from a different process coexist on Linux, so this was never a conflict. The check now skips loopback-only listeners, which also fixes nginx.

11:45, 3cf873e. `systemctl enable --now bind9` refused: "Refusing to operate on alias name". Ubuntu's real unit is `named.service`, and `bind9.service` is an alias that `enable` will not touch. Fixed to name `named`.

11:50, 1cf3a12. The health probe led with `curl` on a `telnet://` address, copying nginx's curl-first probe. But curl's telnet scheme negotiates options that `named` does not speak. It now leads with `nc`.

11:53, c6e830d. The committed agent binary was from phase 4 and had never been rebuilt after phase 7 added BIND dispatch. Phase 7's own tests ran against source, so nothing had ever run the shipped binary against a non-nginx capability. This was the first thing that did. Rebuilt, and the pinned checksum updated.

11:58 to 12:15, the SERVFAIL. A zone create kept failing its health check with a DNS SERVFAIL. It took three fixes and three diagnostic commits.

`named`'s journal said permission denied on the zone file. `/var/lib/lesta/bind` is owned by root and the group `lesta`, mode 0750, and `named` runs as the package's own `bind` user, which had no reason to be in that group. `786036d` added `bind` to `lesta`. Same error.

The bind9 package starts `named` during `apt-get install`, before the group change, and a running process does not re-read its groups. `systemctl enable --now` does nothing to a unit that is already active, and `rndc reload` re-reads config inside the same process. `cb8532d` forced a restart after the grant. Same error, even though `namei -l` and `id bind` showed every permission and the group membership were now right.

The last clue was in the shipped AppArmor profile. `/etc/apparmor.d/usr.sbin.named` allow-lists `/etc/bind/**`, `/var/lib/bind/**` and `/var/cache/bind/**`, and nothing under `/var/lib/lesta`. AppArmor confines the binary, not the user, and that also explained why the session's earlier check, `sudo -u bind test -x ...`, had been a false comfort. That command never runs under the profile at all. `2135cde` at 12:15 appended an allow rule to `/etc/apparmor.d/local/usr.sbin.named`, the extension point that the shipped profile's own last line includes for exactly this purpose, so the vendor file is never edited and an upgrade cannot overwrite the change. The group grant and the restart stayed in, as defence in depth that had turned out to be independently justified, and not been reverted.

12:19, 530d7e0. The regression test added to prove the firewall fix showed each accept rule three times over. `nft -f` does not replace a chain's content when the table and chain already exist. It adds on top. That bug had been in nginx's own firewall code since phase 4, and had gone unnoticed because nobody had looked at the rendered rules across repeated applies until this phase's own check existed. The fix is the standard nftables idiom: declare the empty table and chain, flush the chain, repopulate it, in one transaction.

12:22, 2691361. A check for a listener on port 443 failed with no visible message. A `grep -q ... && echo` tail failing under `bash -e` ends the whole step in silence. It was also a wrong test. A bare `apt-get install nginx` never binds 443. It now checks the firewall's allow-list.

12:26 to 12:32, four commits. With AppArmor now enforcing, phase 7's own tests broke. Its test harness runs a real `named` against a config under `/tmp`, a path the same profile denies. The fix, scoped to the throwaway CI runner only, puts the profile in complain mode right before the test step. It took two steps to get right, from the error text. `aa-complain` is in a separate package, and once installed it failed because its Python scans all of `/etc/apparmor.d/`, where this runner image has two unrelated overlapping profiles for Microsoft Edge that break the scan. `apparmor_parser -r -C` takes one file and skips it.

The last push, `1a259df` at 12:32, went green in all three jobs: `ci`, `agent` and `installer`.

What the count says

Seventeen commits in 56 minutes, four of them diagnostics only. The decision log says eleven distinct bugs, and I have not reconstructed which of the seventeen commits map to which of the eleven. The log's list is the more careful account, and this chapter follows it where they differ.

Two of the bugs were not new in this phase. The duplicated firewall rules had been in the project since phase 4, and the stale agent binary since phase 7. They stayed hidden until something exercised them from a new direction, a second service on the same node. It is a pattern with a plain cause: the first service of any kind cannot prove that shared code is shared correctly. The second one can.

Phase 8 closed the DNS slice: model, interface, capability, installer. Four phases, two days, and the installer had a real BIND running on a real Ubuntu host in CI. The next service was the second web server.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `3c43fa8` , `1a259df` , `a6c77a4` , `d850da8` , `5129e97` , `88f901e` , `e6b195d` , `3cf873e` , `1cf3a12` , `c6e830d` , `786036d` , `cb8532d` , `2135cde` , `530d7e0` , `2691361` .

Two web servers and one front door

This is the chapter where the Apache decision from chapter 4 gets built. It was also the chapter where I overruled a scope recommendation, and where the log first says out loud that the session will not guess twice in a row.

What to build next

The twentieth prompt, on the afternoon of 2026-08-29, was "whats next?". There was no established sequence any more, since DNS was finished. The session asked, and offered three choices: the Apache capability in Go, ACME hardening, or automated enforcement of the installer contract. I chose Apache.

Three research agents ran in parallel: one on everything the ADR said about Apache and the combined profile, one on the exact architecture of the nginx capability, one on the state of the Laravel side. The most important finding was that the control plane needed nothing new. `WebDomain` already produced a payload that had no nginx in it. A node capability was a bare string, so a node could already hold `web.nginx.v1` and `web.apache.v1` together, and the factory already faked exactly that. And the code that picks a node for a domain already preferred nginx whenever both were present, with a comment that said so. That settled the scope: Go agent work only, in the same shape as phase 3.

The planning agent found two things specific to Apache. One was hands-on, verified against the Apache that ships with macOS. Apache's `Include` directive fails on a glob that matches nothing, as BIND's does and nginx's does not. But Apache has `IncludeOptional`, which does not, so unlike BIND the Apache capability needs no placeholder file. The other was a problem. nginx can answer a request inline with `return 200 '...';`. Apache has no equivalent. The realistic option was `mod_asis`, which serves a companion file as the complete response, so the plan followed BIND's two-file design: a thin config stanza that points at a numbered content file.

The session also settled one fork itself, and said why. The shared contract test suite could have been made generic for Apache, since Apache's payload is byte-for-byte the same as nginx's. It chose to duplicate the suite anyway, because the shared suite asserts that a suspended-then-unsuspended resource has an identical digest, and that would be silently wrong for Apache, whose vhost fragment also embeds a generation-specific path, as BIND's does. The log adds: this is the third time in the project that a self-contained real-capability test file was chosen over growing a shared helpers file.

Eleven minutes, four commits

The agent implemented it, and the session read every new file against the plan and ran `gofmt`, `go vet`, `go build` and `go test` fresh, with the cache turned off. Thirteen real-capability tests passed, with runtimes of seven to nine seconds that are consistent with real processes starting and not with skips. It

committed as `3530363` at 14:27, and CI failed.

The failing step was the one meant to prove that nothing needs any capability installed yet. Every Apache test failed with the same message: `module unixd_module is built-in and can't be loaded`. GitHub's Ubuntu runner ships with `apache2` already installed. Ubuntu's `apache2` compiles `mod_unixd` into the binary. The Apache on the laptop does not, and the test harness had a fixed `LoadModule unixd_module` line that worked on every local run and was a hard error on the real target.

The fix, `211d56b` at 14:31, was to ask the binary what it has built in before loading anything. The rerun failed differently, and worse: a bare `exit status 1`, with no output at all. The session then did what the log calls diagnostics first. It reasoned about how Apache starts. A failure while parsing the config is printed to stderr before the `ErrorLog` directive is reached. A failure found later is written to the `ErrorLog` file, and that would explain a silent exit. Instead of guessing a second time, it made the harness read and print the `ErrorLog` content whenever startup fails, and the list of modules it thought were built in, and the whole rendered config. That went out as its own commit, `ec82ee4` at 14:35, with no fix in it.

CI showed the real cause at once. Ubuntu's Apache also compiles `mod_log_config` in, among others, and separately needed a `TypesConfig` file. `edc5af3`, the last commit, went in at 14:38. Four commits in eleven minutes, all four CI jobs green. The log states the rule it had followed: never guess twice in a row once real evidence is available. One of the four commits was diagnostic only.

The Laravel side was untouched. The scoping claim from the research, that the control plane needed no work, had held in practice as well as on paper.

The widest option

That evening the session offered a choice of next steps again: the real Apache installer, ACME, or contract enforcement. I chose the Apache installer.

Before planning it, the session looked at what the backlog item meant, and found that it was three things wrapped into one. The installer itself. A fix for `nginx/install.sh`, which rejected `--web-server apache` and `--web-server both` outright. And the reverse proxy from nginx to Apache for the combined profile. A second question then surfaced a gap that no document had covered. On a node that runs both web servers, nothing decided which domains nginx should serve directly and which it should hand to Apache. `WebDomain` had no field for it, and the resolver's own comment said nginx always wins when both are present.

The session presented three scopes and recommended the narrowest: the Apache installer alone, deferring everything about the combined profile, in line with the project's habit of putting off cross-cutting design until a real need exists. The log continues: "The user chose the widest option anyway: installer, both-profile orchestration, and the real per-domain routing design, in one phase."

"The user" is me. The combined profile was what I had asked for in the third prompt on 2026-08-26. It was also the largest phase so far, and it was the one that the recommendation had warned against. Four bugs followed.

The design

The plan had three parts.

A standalone Apache installer. It mirrors the BIND installer's independence: its own `main()`, its own trap, and a narrower `--web-profile apache|both` flag that only changes its topology awareness. The nginx script, which the installer contract still calls "the web installer", starts the Apache installer as a child process for `--web-server apache|both`. It never sources it, which keeps each script's `set -eu` and its trap from leaking into the other.

A choice per domain. `WebDomain` gets a `web_server` column, an enum. The resolver that picks a capability returns an ordered list instead of a single string. The provisioning payload takes the capability as a parameter, and for the nginx half of a proxied domain it overrides the template with a fixed sentinel, `apache-proxy`.

A proxy template. nginx gets a new `apache-proxy` template that forwards to `127.0.0.1:8080`, with a deliberately generic health check. The two operations, one to nginx and one to Apache, have no ordering guarantee against each other, so nginx's check cannot depend on a marker that only Apache's content provides.

A background agent implemented all of it. The session read every diff: shell, Go, migration, enum, five action classes, two form requests, the controller, the factory, three React pages, the types, and every touched test. It ran the checks fresh, including a new test in which a real disposable nginx proxies a real HTTP request to a real disposable Apache and returns Apache's own content. Then it pushed, and CI found four problems.

Four bugs, none of them guessed at blind

The first CI run, of a new job that tests the combined profile, ended in an error message: `AH00111: Config variable ${APACHE_RUN_DIR} is not defined`. The installer's own `apache2 -t` call skips `/etc/apache2/envvars`, which the package's `postinst` and `systemd` both source. The Go agent already handled this in phase 9. This was a different code path, and it would have broken the plain Apache-only apply too, if anything had ever run one. The fix, `6e3efa0` at 21:59, was `apache2ctl configtest`, which is the tool made for it.

The next failure was `unsupported capability "web.apache.v1"`. The vendored agent binary had last been rebuilt for BIND, in the commit from the previous chapter, and never after Apache was added. Phase 9's plan had said explicitly not to touch the installer job, so nothing had refreshed it. The session rebuilt it, confirmed with `strings` that the capability string was in there, and updated the pinned checksum. That was `91c81e7`, at 22:04.

The same push failed an unrelated test, in BIND's harness: `TempDir RemoveAll cleanup: directory not empty`. The harness stopped `named` by sending `rndc stop` and returning at once, without waiting for it to exit. Go cleans up in reverse order, so removing the temporary directory sometimes raced a `named` that was still shutting down. The bug was old. It appeared now because the extra load in CI shifted the timing enough. The fix waits for the process to exit, in `9166506`, also at 22:04.

The last one came when the apply finally reached its own self-test and got a real HTTP 403. Apache's workers run as `www-data`, which must read the content file that `mod_asis` serves, under `/var/lib/lesta/apache`, mode 0750 and owned by root and the group `lesta`. nginx never needed this, because its `return` line is parsed into the config and never read from disk at request time. The gap is the same class as BIND's in the previous chapter, and so is the answer: a group grant is not enough, because Ubuntu's `apache2` package also ships a default-deny AppArmor profile, and the package starts the service before the grant runs, so `enable --now` never makes it pick up the new group. The session applied both lessons at once. `6499158`, 22:10.

All four jobs went green: `ci`, `installer`, `agent` and the new `installer-both-profile`. That closed the web capability set that ADR 0002 had scoped, nginx, Apache or both, real from the installer through the Go capability to per-domain routing in Laravel.

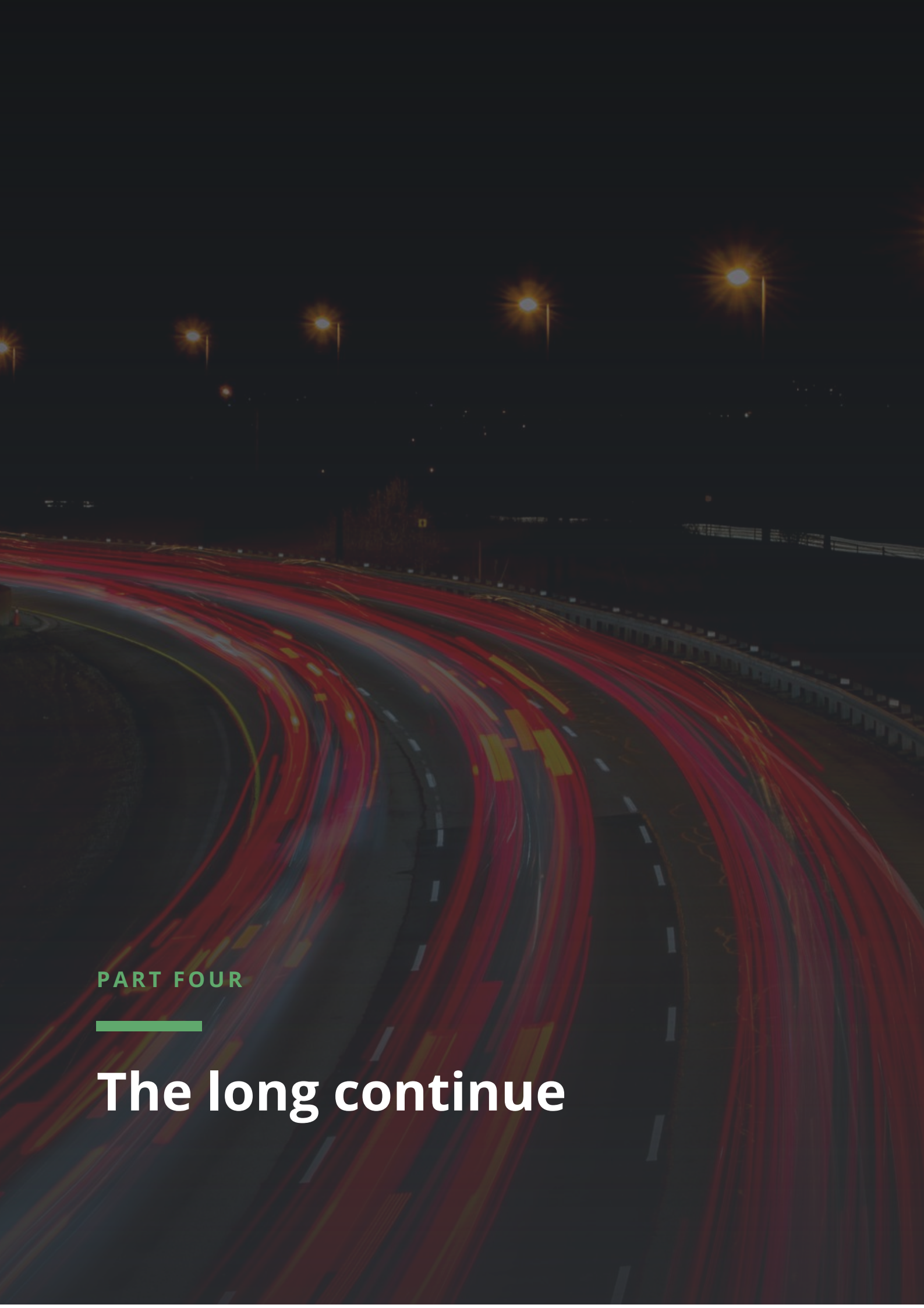
The one honest gap

The log ends this phase with a sentence I take for the chapter's last. No real network connection yet existed between Laravel and an agent running on a node. That had been a documented scope cut since phase 3. So the Laravel side of the dual dispatch was tested against the fake provisioner and against real web servers in Go, but never end to end across a network, since there was no network between them. It was the oldest gap in the project, and chapter 14 closes it.

Between 14:03 and 22:10 on 2026-08-29, in one day, the project got a second web server, a combined profile and a routing design. The commit history is then empty until 22:59 on 2026-09-01, although chapter 11 shows that work went on through 2026-08-31.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `3530363`, `6499158`, `211d56b`, `ec82ee4`, `edc5af3`, `6e3efa0`, `91c81e7`, `9166506`, `c006fec`.



PART FOUR

The long continue

Thirty-three hours

HTTPS was next. On 2026-09-01 the session offered the usual three choices: ACME hardening, automated enforcement of the installer contract, or a pipeline for offline release bundles. I chose ACME, the protocol that Let's Encrypt and other certificate authorities use to hand out certificates to whoever can prove they control a domain.

This chapter is short on the surprises that fill the other chapters. It is the phase with the least drama in the code and the most in the calendar. The log calls it, by a wide margin, the longest and most interrupted implementation pass of the whole project.

Two facts that changed the design

The session did its research first, and two things it found decided the shape of the phase before any design could start.

The first was that `WebDomain` has no link at all to `DnsZone`. The second was more important. The node protocol from chapter 6 is strictly one-shot: one operation envelope in, one result envelope out, and then the agent process exits. The real ACME flow is nothing like that. You create a challenge, wait for it to be visible, tell the certificate authority to validate, wait for that to finish, and download the certificate. It is a multi-step process that takes minutes.

That ruled out putting the ACME client in the Go agent at all. The client had to live in Laravel, which has a persistent queue worker and direct access to the database, and it would orchestrate a much simpler Go capability that only ever writes files: the challenge file, the certificate, the key.

Because this reframing was large, the session put it back to me and did not assume. I confirmed it. A second question settled the library. Writing the JSON Web Signature handling and the ACME state machine by hand was the alternative to `acmephp/core`, an established PHP ACME v2 client. A new dependency needs my approval under the project's own rule. I approved it. A third question narrowed the scope. HTTP-01 and DNS-01 would both be supported, honouring the earlier decision from the first evening, but the HTTPS virtual host template would be nginx only for now.

The plan

The plan saved to the vault has these parts.

- A system-wide `AcmeAccount` model. Its private key is encrypted and never appears in any provisioning payload, which is what ADR 0002 says about ACME account keys. The key never has to leave Laravel, since Laravel is the one talking to the certificate authority.

- An `IssueAcmeCertificate` job with a deliberately narrow DNS-01 rule: it is used only when explicitly requested, and only when a zone that matches the domain exactly exists. There is no automatic detection.
- A DNS-01 design where the temporary TXT record never touches the real `dns_records` table.
- A small Go capability, `tls.acme.v1`, with two kinds of resource.
- A shared `.well-known/acme-challenge/` location added to every nginx template, and a new `default_ssl.conf.tpl`.
- A real test certificate authority in both languages. It is called Pebble and is Let's Encrypt's own test server.

The interruptions

A background agent implemented the plan. Then the log records something that has nothing to do with the code. The agent was cut off by the local machine going to sleep in the middle of a response, five separate times. It was also cut off once by hitting the session's rate limit. It took roughly 33 hours of wall clock, from 2026-08-31 into 2026-09-01, and the log's own words are that this was purely due to the interruptions and not to the complexity of the task.

Each time, the session did not restart the agent from scratch. It checked the actual state of the working tree with `git status`, ran `gofmt` and `go vet` to find exactly which line had been left half-edited, and resumed the same agent with the evidence of where it had stopped. That kept all the context it had built, and avoided doing correct work twice. Between two of the resumes the session found an unrelated one-line change to `CLAUDE.md`, in a line that Laravel Boost manages. It reverted it and told the agent to leave that file alone, and it stayed untouched afterwards.

The commit history has nothing between 22:10 on 2026-08-29 and 22:59 on 2026-09-01. That silence is where most of the 33 hours sit.

Checking the work, and a mistake in the delivery

The session checked every file itself. One question needed real thought. The challenge and the certificate install both use the same `resource_id`, the web domain's own UUID, under one capability. Could the cleanup of the challenge, a `delete`, accidentally delete the certificate that was just installed? It read the code and found that it cannot. Both `applyDelete` and `applyWrite` work out the file path from the payload's own `token` and `domain` fields and never from the resource ID. The resource ID is used only for the idempotency bookkeeping.

It also confirmed that the account key isolation held end to end, that the trigger wiring did not fire twice in the combined profile's dual dispatch, and that a change forced by the new library, a downgrade of Guzzle from 8 to 7.15.5, was within what Laravel supports and was used in one file only.

Then the session installed Pebble locally to run the real test and found a bug in what had been delivered. With the newest Pebble on the path, the full test suite showed two skipped tests, but filtering to the Pebble tests showed that they had failed, with `Undefined array key "token"`. That contradicted

the skip message in the test itself, which recommended installing the newest version.

It found the cause by installing `pebble@v2.5.0` specifically. Both tests then passed. Newer Pebble versions offer an extra challenge type, `dns-persist-01`, whose JSON response leaves out the `token` field on purpose, and `acmephp/core 2.1.0` reads that field on every challenge type it is offered. The agent's own report had found this and pinned the CI job to the right version. But the skip message in the PHP test file still named the wrong version. The delivered text was wrong, and not only incompletely disclosed. The session fixed the message and added an explanation in the comment at the top of the test file, since Pebble's own `-version` output is just the string `dev` and cannot be used to detect the mismatch at run time.

CI

The commit `b61f451` went out at 22:59 on 2026-09-01. CI failed in its static analysis step. `openssl_x509_parse()` returns an array or `false`, and `certificateExpiry()` read `validTo_time_t` without checking. It had never failed locally because every local run had a certificate that parsed. A guard for `false` and a check with `isset` fixed it, and `composer ci:check` passed: PHPStan with zero errors, 226 of 226 tests with Pebble on the path. That was `9c0b51d`, three minutes later. All four CI jobs were green, including the agent job, which installs Pebble and runs a real Pebble-backed Go test in CI itself.

What was left out on purpose

The log lists four things that it deferred and disclosed, so that they would not be dropped silently.

- Apache's own HTTPS virtual host template, following the same nginx-first pattern as the web capability. It arrives as phase 14.
- An installer for the ACME service.
- Wiring Pebble into the PHP CI job. The test ran locally and skipped itself in CI. It stayed that way for nine days, and phase 26 explains what it hid.
- A forced redirect from HTTP to HTTPS, which the log calls a product decision and not an engineering constraint.

The bet, seen from here

By the end of this phase, on the night of 2026-09-01, it was seven days since the challenge. Two days later a member of the forum would post the arithmetic on it, and I would answer. That exchange, and what was going on around it, is the subject of the next chapter.

SOURCES

- Commits, each at github.com/mikho/LESt/commit/ followed by the hash: `b61f451`, `9c0b51d`.

Tokens

This chapter is about what the project ran on. It does not follow the code, and it goes back and forth in time.

What I said it would run on

On the first evening a member asked what setup, model and expected cost I planned. I answered that it would be Claude, with models and effort varying depending on how many tokens I had left on my company account.

On 2026-09-03 another member wrote that a week had gone by on a bet of a day or two. My answer, in full: "In my defense, I use Claude as a work tool, and I have prioritized things I get paid for above this project. Using 'left-over' tokens. And the project changed more than a bit with the decision to write something without all the current bugs and exploits."

The same member added that they had spent about three months on the Pro plan and one on Max, on a refactor of their own, and that it was not just leftovers. Their view was that a first deployable version of mine would take some months. I told them I was about to finish two larger projects that month and would have more tokens for this afterwards. I have the Pro subscription.

Another member asked how anyone works on Pro. My answer was that I cannot sit with it all day, since I have other customers. It is enough for one to three five-hour shifts a day. My weekly limit resets on Thursdays at 21:00 Swedish time. On the Monday I was at 80%, and then I remembered I would be at a client's site for the next couple of days, so I managed. On 2026-09-02 I had installed a second coding plugin and logged in, and my weekly limit for Claude was reset. It reset again on the Thursday at 21:00. A member found that the first reset was one that the vendor had given everyone in the middle of the week, and that the timing with the plugin was a coincidence.

None of that is in the decision log. It comes from what I posted at the time. The log records the limit only when it interrupted work.

Where the limit stopped the work

The decision log mentions the limits in five places.

- **2026-08-27, 23:38.** The fourteenth prompt, in full: "my session tokens are almost over, can we automatically continue in 2 hours". It led to the finding in chapter 8 that a cloud-scheduled routine could not read the vault, and the answer was to resume the local session.
- **2026-08-29 morning, phase 8.** The first attempt to launch the design agent for the BIND installer hit the session's rate limit in the middle of the call and stopped with an error before producing

anything. It was retried after the reset, "with no loss of context or rework".

- **2026-08-31 to 09-01, phase 11.** Five interruptions from the machine going to sleep and one from a rate limit, over roughly 33 hours, as chapter 11 tells.
- **2026-09-05, phase 20.** The first attempt to launch the implementing agent failed at once on a session rate limit, before a worktree existed. Relaunched identically, and it worked.
- **2026-09-06, phase 25.** The session hit a rate limit while an agent was in the middle of an implementation. The core installer changes were done and the CI scenarios were not. Instead of relaunching from scratch, the session read the partial worktree diff and continued from it.

Each of these is a run that could be resumed. Why they could is a property of how the project had been run since chapter 6. The state of the work was on disk, in the vault and in the repository, and not in any session's memory.

What the transcripts say

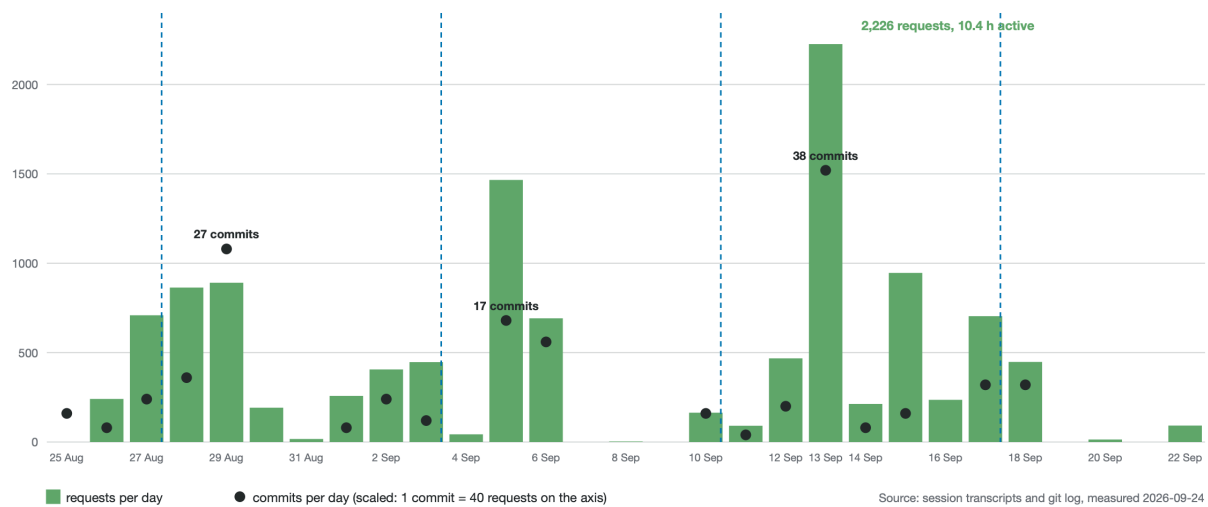
The Claude Code session files for the repository are kept on my machine, with a record for every request. I summed them for this book. The method and the caveats are in Appendix A. The short version: about 11,800 requests, of which 4,100 came from subagents, about 7.5 million output tokens, and about 62 million fresh input tokens. Active machine time is about 58 hours, counting only the gaps under ten minutes, over 24 calendar days.

Split by the weekly windows that my limit resets on, Thursday 21:00 Swedish time, the record looks like this.

Weekly window starting	Requests	Output tokens	Fresh input tokens	Active hours
Thu 2026-08-20 (partial, the session records start 08-26)	763	0.59 M	5.8 M	5.0
Thu 2026-08-27	3,122	2.11 M	23.7 M	14.9
Thu 2026-09-03	2,344	1.25 M	12.2 M	10.9
Thu 2026-09-10	4,899	3.12 M	16.5 M	22.4
Thu 2026-09-17 (partial, to 09-22)	703	0.41 M	4.0 M	1.8

Figure 4. Requests and commits per day

Bars: assistant requests in the Claude Code transcripts. Dots: commits. Dashed lines: weekly limit reset, Thursday 21:00 Swedish time.



The two scripts I used to build this table and the headline numbers differ by 41 requests, because one of them leaves out synthetic messages. Neither number is more right than the other at that level.

The week starting 2026-08-27 is the one my forum post described. Inside that window, the cumulative share of the week's output tokens on this repository was 6% by the end of Thursday, 38% by Friday, 72% by Saturday, 79% by Sunday and 80% by Monday. Then almost nothing on Monday, 17 requests in the whole day, which fits a person who is at a client's site. The rest came on Tuesday to Thursday.

I want to be careful with that 80%. My own figure in the forum was a share of my plan's weekly limit, which includes everything else I use Claude for, and the transcripts cover only this repository. The two numbers are the same by coincidence. What the transcripts do show is the shape of the week, a heavy run in the first three days after a reset and a tail. Nothing in the record contradicts what I said.

The busiest week is the one starting 2026-09-10: 4,899 requests and 3.1 million output tokens. The busiest day is 2026-09-13, 2,226 requests, 1.37 million output tokens and 10.4 hours of machine activity. It is also the day of 38 commits, and chapters 17 and 18 are about it. The quiet days are as informative. There are no requests at all on 2026-09-07, 09-09, 09-19 or 09-21, and 3 on 09-08.

What "continue" means in practice

Of the 58 prompts I typed in these sessions, 18 are 25 characters or shorter. The most common are "next phase" four times, "commit and push" four times, "resume" three times, and "next" twice. The rest of the 58 are longer, and a few of them are the long prompts that this book quotes. What I did in the long middle of the project was read the proposal the session had made, choose among the options it offered, and tell it to go on. That is how I would describe it, and the numbers agree.

It is worth saying what that number does not mean. 58 prompts did not produce 160 commits. Each prompt started a run in which the session designed, delegated to background agents, checked their work, ran the tests, pushed, and watched CI. The 4,100 subagent requests are the trace of that

delegation. The work is real, and I am not the one who typed it.

What the numbers cannot say

The record has clear limits, and they should be stated as plainly as the numbers.

It cannot say what fraction of my plan any of this used, since the plan's limit covers all my work and the transcripts cover one repository. It cannot tell how many hours I spent, because a long agent run continues while I am away, and the active time is machine time. It has nothing on the first three prompts, which ran in another tool. And I have not converted any of it into money. A subscription is not billed per token, and I did not look up the API prices for the model in question, so any dollar figure here would be invented.

What I can say is one fact about the budget as I posted it: this project ran on the leftover part of a Pro plan, next to paid work. That fact explains the calendar of the next chapters better than any of the numbers. Some days show a burst and others show nothing, and the record does not say why the empty ones are empty.

A comment on the whole subject

On 2026-09-17 a member wrote that it is insane that a big tech company decides whether you can keep thinking, "a mental taxing system", and that they do not let AI run their projects completely. Another member answered that it only takes speed away unless you pay more, that development without AI would need more developers to get the same speed, and that it will not stop you from thinking.

I did not answer either of them. What I can add is what the limit did to the method. Because a run can be cut off at any moment, the state of the work had to be written down after every prompt, and it is the reason this book can be written from the record. A limit is a nuisance, and it also forced the discipline that made the work resumable.

SOURCES

- The discussion thread, posts of 2026-09-03 and 2026-09-17: lowendspirit.com/discussion/11268/vibe-coding-a-vestacp-alternative.

Databases, cron and a shared identity

Two more services in three days, and a decision about cron that took two days to undo.

Tenant databases

On 2026-09-02, with HTTPS done, the session offered tenant databases, Apache's own HTTPS template, or contract enforcement. I chose databases, because ADR 0002's build sequence said so. Two rounds of questions followed, and I asked for the widest scope: relational foundation, quota, actions, policy, interface, the real Go capability and the real installer, all in one phase. That matched how wide phases 9 to 11 had gone.

The second question was about how a Go program should talk to MariaDB. Every capability so far rendered a config file and reloaded a service through an external binary. None held a database connection. The choice was between adding a Go MySQL driver, which would be the agent's first external dependency, and running the real `mariadb` command-line client. I chose the client. The agent module had no external Go dependencies at four capabilities, and that stayed true.

A third question came out of a tension in the ADR's own wording. It says database credentials are never included in normal desired-state payloads or queue payloads. That is the rule that kept the ACME account key out of every payload in chapter 11. But an ACME key never has to leave Laravel. A tenant's database password has to reach the node, or no user can be created. The design resolved it with an explicit invariant on both sides. Laravel's `toProvisioningPayload()` takes the password as a named parameter and never carries it otherwise. On the Go side a pair of functions, `requirePassword()` and `forbidPassword()`, insists that only the operations that need a password have one. There is a negative test that does a literal `array_key_exists`, and not only a check that the value is hidden.

The planning agent verified two facts it could have guessed at. `Account` has no slug or username, so database names use the always-unique numeric account ID as the prefix. And Ubuntu's `mariadb-server` package ships a real `mariadb@.service` systemd template unit, which answers how two MariaDB instances, the control plane's and the tenants', can coexist on one host, with evidence and not from memory.

One thing it could not settle, and said so. MariaDB Foundation's own apt repository publishes metadata for Ubuntu 24.04 and 22.04 only, not 26.04. The installer was built to fail closed on 26.04 and not to substitute Ubuntu's own version, which nobody had approved. The log flags it for re-verification when a future MariaDB release or an ADR amendment changes the picture.

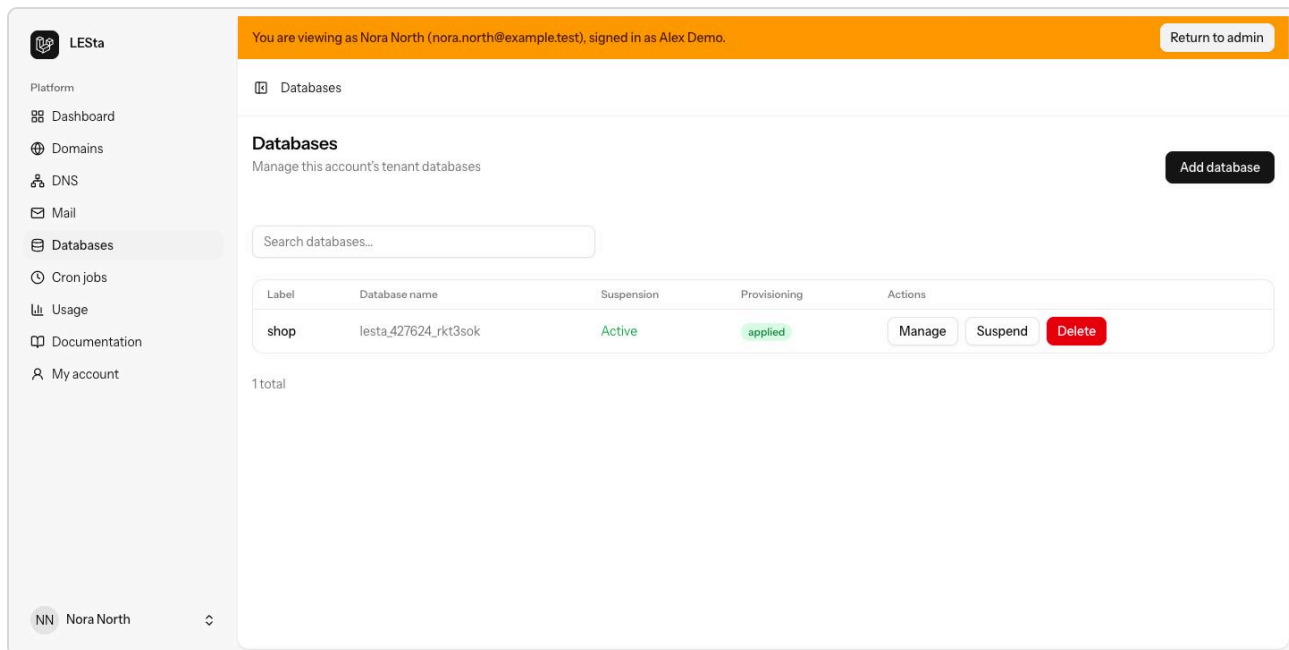
The session reviewed the agent's code with one detail in mind. Rotating a tenant's password must use `ALTER USER ... IDENTIFIED BY . CREATE OR REPLACE USER` is documented as drop-and-recreate, and would silently wipe every grant the user had. The code did it right.

It also found that the agent had never touched the CI workflow. The new installer changed a live systemd multi-instance setup, an AppArmor profile and an external apt repository, and it had no proof on real Ubuntu, which the nginx, BIND and Apache installers had all had. The session added the dry run, apply and idempotent re-apply sequence, a runner step that purges whatever MySQL or MariaDB GitHub's image ships, and a port check, and committed it as `0b6b614` at 13:38 to get evidence.

The evidence was four bugs in fourteen minutes:

- **13:42, 11ec372** . The real package reports its version as `1:11.4.13+maria~ubu2404` . A pattern that looks for `11.4` never matches a string that starts with `1:` , so the installer failed with "wrong version, repo shadowed" although the right package had installed. It now strips the epoch before comparing, and also checks provenance with `apt-cache policy` : that `deb.mariadb.org` , and not just a matching number, provided the package.
- **13:46, 2a68128** . `mysql` could not walk into its own data directory's parent. `/var/lib/lesta` is 0750, root and group `lesta` . `mysql` is neither. `mariadb-install-db` failed with "Permission denied" even though the tenant directory itself had the right owner. It is the same gap that `named` and `www-data` had hit, and the same fix.
- **13:49, a2f7724** . The vendored agent binary was stale again, for the third time in this book. It had not been rebuilt since Apache's fix, so `database.tenant.v1` was missing from it. The session had learned the sequence from phase 10: check with `strings` first, then rebuild.
- **13:52, 94ccec3** . The admin account's password went out of sync with its credentials file on the first re-apply. `CREATE USER IF NOT EXISTS` does nothing to an existing user, but a fresh password was generated and written on every run. The idempotent rerun step, which has run for every installer since phase 4, caught it at once, as "Access denied for user 'lesta_agent'@'localhost'". The fix was `CREATE OR REPLACE USER` . It is right here and would have been wrong for tenant passwords, because the next statement re-grants this account's fixed set of privileges on every run anyway.

The final run was green on all four jobs. The tenant database slice was real end to end, on the first attempt at the feature.



Tenant databases as a customer sees them. The name is generated from the account ID and a label.

Cron, and the decision I deferred

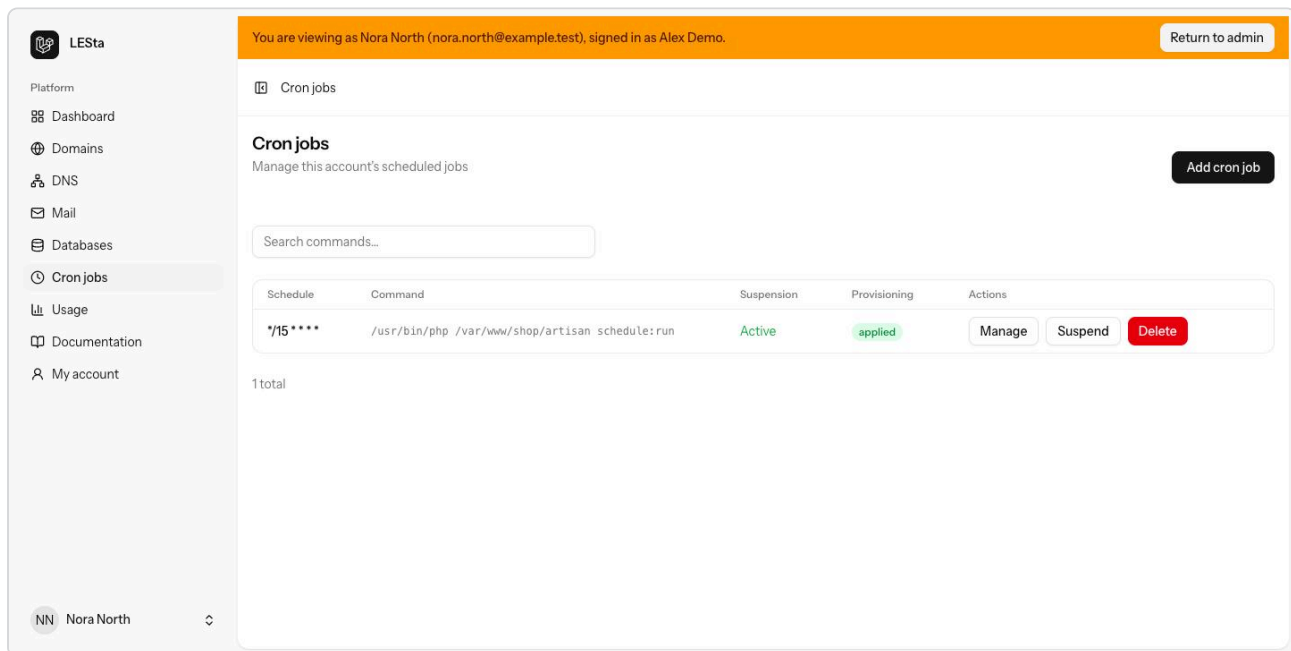
The next day, 2026-09-03, the session offered cron, Apache's HTTPS template or contract enforcement. I chose cron. The session's research found that the plan left a real fork open on purpose. The ADR's gate text says: "Prefer a catalog of supported tasks or an isolated account-scoped runner." Nothing in the project had ever created a per-account Linux user. It asked me, and gave three options: a shared unprivileged runner with a disclosed limitation, per-account Linux users built now, or a catalog of approved task types only. I chose the shared runner, the narrowest slice. It resolved the fork toward the least work, and toward a limitation that would have to be stated.

Designing it, the session found the phase's security idea before writing any code. A tenant's command text never has to appear in a crontab line at all. Each fragment's scheduled line calls a fixed wrapper, `lesta-agent cron-run <resource_id>`, where the resource ID is the server-generated UUID that every operation already carries, never tenant input. The wrapper reads the real command from a JSON file next to it when it runs. That removes crontab-line injection through the command field by construction, and does not depend on escaping text. Only the five schedule fields end up in the line itself, and they are range-checked numerically.

A consequence followed from an old gap. No network connection yet existed between Laravel and a node's agent. So a job's execution history could be recorded on the node for real, but not shown in the Laravel interface, in the same way every installer's self-test reported "structurally installed, not yet control-plane-registered". The create-job page says so on the page itself.

The agent built it from an unusually detailed prompt, and the session read every file. It traced the installer's file permissions by hand: setgid directories, mode 2750 and 2770, so that the shared `lesta-cron` identity can read sidecar files and write execution logs it did not create. One bug came from CI, and it was in a step added that day. A check that MariaDB was still running called `mariadb -u root`

without `sudo`. MariaDB's `root@localhost` authenticates by the operating system user, which the installer's own comment already said. The fix, `8b67d7d`, added `sudo`. The self-test in this phase went one step further than any before it, by also running the real `cron-run` wrapper against a sidecar the test had just written.



The cron list. Only the five schedule fields end up in a crontab line. The command lives in a separate sidecar file.

Splitting the log

That night I asked for something unrelated to the code. The decision log had grown to 493 lines, and every prompt that attached it dragged in the whole history. I asked for it to be split. The session gave me three ways to do it: one file per phase, a recent log plus an archive, or by calendar range. I chose one file per phase.

It moved the whole log into one file for the early foundations and one file per phase, each with a pointer to the index and to the phase before. It rewrote the main log as a light index, one line per phase. Nothing was summarised away. This book reads the log in that form. The log's stated reason is that a prompt no longer needs the whole project history to have context. A shorter attachment also costs fewer tokens, which is my own addition, and fits chapter 12.

The Apache HTTPS template

Phase 14 followed the same evening, and it is the smallest here because the session found that every fork in it was already decided by shipped code. The answer to the one open question was in the research. Ubuntu's stock `ports.conf` wraps `Listen 443` in `<IfModule ssl_module>`. That is evaluated before this project's own module fragment, included later with `IncludeOptional`, loads `ssl_module`. So

the stock `Listen 443` never binds. The fragment has to write its own `Listen` directive, and it has to suppress it in the combined profile, where nginx owns the public ports. The commit was `97f9aa1` at 22:26.

The limitation comes due

Two days later, on 2026-09-05, phase 21 undid the decision from cron. The session had listed the shared identity as a disclosed limitation, and I chose to close it: per-account operating system isolation, one Linux user per tenant account on each node.

The research found that it was deeper than a small extension. Every Linux user in the project was created once, at bootstrap, by a shell installer. Nothing created users afterwards on behalf of a tenant. And since Laravel never shells into a node, a per-account user could not be an installer change. The agent daemon itself had to gain a capability: create a system user for a tenant account, dispatched like any other operation.

Three forks came to me. Create an account's identity on a node lazily, the first time the account gets a cron job there. I chose the recommended lazy option. Accept that the identity and the cron job may be created at slightly different times, since both were already asynchronous. I accepted that. And whether to delete an orphaned identity when an account's last cron job on a node goes. I chose to leave it and disclose the gap. While the plan was being written I added a fourth thing. Rather than leave the gap with no tooling, a provider admin should be able to see the orphaned identities and delete them by hand. It went into the plan as its own part before it was approved.

The result was a new `system.account-identity.v1` capability with idempotent `useradd`, `userdel` and `id`. The agent binary and the directories above it became world-traversable, since per-account users are deliberately never in the shared `lesta` group. Cron's state moved from one group-readable tree to `StateRoot/accounts/<run_as>`, owned by root and that account's own group, mode 2750. A new model, `AccountNodeIdentity`, plugged into the existing outbox and result pipeline with no new plumbing. It went in as `cbbb9f2` at 20:38 on 2026-09-05, and a fix, `c3afaad`, followed eight minutes later for an orphaned-identity interface piece and another stale agent binary. That is the fourth.

What a decision costs

The shared runner was the narrowest option for cron, and I chose it. Two days later I chose the per-account users. A reasonable reading is that the first choice was wrong. A more accurate one is that it was scoped to what could be proven in a day, and that the limitation was written on the page where users would see it. It was undone when its turn came, at a cost of one phase, with nothing left over that anyone would have to unlearn. That is the case for putting a limitation in writing when you choose it.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `5b56edd` , `94ccec3` , `7a9e75e` , `8b67d7d` , `97f9aa1` , `cbbb9f2` , `0b6b614` , `11ec372` , `2a68128` , `a2f7724` , `c3afaad` .

Eight phases in twelve hours

Saturday 2026-09-05 has 17 commits in the repository, and the first one is stamped 09:41. By 21:17 the project had moved through eight phases, from 15 to 22. This chapter follows that day up to the network. The offline release bundle at the end of the day has its own chapter.

Here is the shape of a session by now. It was a single run of the "twentieth prompt" that started on 2026-08-29 and kept going. After each phase, the session asked me what to build next and offered three options, and I picked one. Then it researched, put the real forks to me, wrote a plan, had an agent implement it, checked everything, pushed, and watched CI. The logs of that day read like a shift report.

09:41, phase 15: making the contract enforceable

The oldest item on the backlog was in ADR 0002, section 9. The installer contract had been written on the first day, and it listed the checks that should enforce it: a manifest schema validator, a static contract checker, shell linting and disposable Ubuntu tests. The ADR said plainly that CI enforced none of them yet. Shell linting was the only one that existed.

This phase took three rounds of questions, more than any phase since the first web installer. In the first I chose the full scope of the three offered: the static checks, plus a test scenario matrix for fresh installs, reruns, interrupted runs, conflicting packages, tampered artifacts, failed validation, failed health checks, upgrade and rollback. That was wider than the recommended default.

The second round came from a finding the session made while checking that "full scope" could really be built. It could not, wholly. `checkpoint.sh` says in its own comment that it never causes a phase to be skipped, and only labels. And upgrade and rollback had no mechanism anywhere in the installer layer. Every installer installed one pinned version unconditionally. What the ADR called side-by-side generations was pure aspiration. The session did not narrow my choice quietly and did not quietly begin designing a new version comparison subsystem. It put the question to me. I confirmed: build real tests for the other six scenarios now, since real mechanisms stood behind them, and defer upgrade and rollback as tracked backlog items. Phase 20 was the first of those.

The third round settled two smaller things. The forbidden-token check would look for the one concrete example the ADR gives, an unpinned remote fetch piped into a shell, and not attempt a broad and ungrounded analysis of writes into refused directories. And the validation tooling would be a Node script with no new dependencies, since it runs only in CI and has no reason to inherit the installers' rule of avoiding `jq` on the node.

The agent built it, and the session tested the tester. It did not accept a clean pass from the new validator at face value. It injected a real `curl | sudo bash` line into the nginx installer and confirmed that the checker caught it with an accurate file and line. Then it added a genuine dependency cycle

between the mail and BIND manifests. Its first attempt used the wrong capability name, which showed that the unresolved-dependency check also worked. It restored both files after each try.

The commit, `1e673ab`, failed in CI at once. The new scenario job reported `checksum_mismatch`, and the reason was in the job itself. The agent had added a step that rebuilt the vendored agent binary from source on the runner, and a fresh `go build` produces a different hash from the one pinned in the manifest. No other job does that. The rebuild was the defect. The fix, `e25b1f3` at 09:44, removed it and the Go setup step that went with it. `bcd38b2` followed to tighten the MariaDB conflicting-package scenario to what it really returns.

11:12, phase 16: the day Laravel met MariaDB

The next choices were the control-plane database, cron's execution history, or the upgrade item that phase 15 had just deferred. I chose the control-plane database, the one remaining piece of the MariaDB slice.

Two questions shaped it. The first was about the service unit. The control plane's instance could use a templated `mariadb@control.service`, for symmetry with the tenant instance, or Ubuntu's plain `mariadb.service`, since it is the only instance on the standard port 3306. I chose the plain one. The second widened the phase. Should Laravel's own local and CI database move from SQLite to a real MariaDB against the new instance, or should the phase stop at the installer and the documentation? I chose the wider one, which proved the control plane's database usable end to end and not only installable.

The design found an important fact before any code was written. Bootstrapping the control-plane instance is a relocation, not a creation. The log's reasoning is that Ubuntu's package almost certainly starts `mariadb.service` against the stock data directory during install, before the installer's control-plane phase runs. So the installer has to stop the service, move an already-initialised data directory, and start again. That is a different operation from the tenant instance's, which creates an empty directory for a template that has never run.

The same pass calculated a defect in advance. The composite unique index on `dns_records`, over name, type and the 512-character value, came to about 4,092 bytes in utf8mb4. MariaDB's InnoDB limit is 3,072. It was the same class of gap as chapter 9's, now recurring in the other columns of the same index, since SQLite enforces none of it. The fix was to shorten the name and type columns, and the log checks the new sum by hand: $(191 + 16 + 512) * 4 = 2,876$ bytes.

The agent then did something the prompt had not asked. It ran the test suite against a real MariaDB and found another sixty-four-character problem. A `morphs()`-generated index name on `provisioning_operations` was 69 characters, over the 64-character identifier limit. SQLite does not enforce that either. It had been invisible until the whole suite ran against a real database for the first time, eleven days after the first commit.

CI failed where the design had said the risk was highest. The relocation could never be tested locally. The diagnostic step added in phase 12 looked only at the tenant instance, so it was blind to a control-plane failure that never reached the tenant phase. The session did not guess. It extended the diagnostic

to capture the control-plane service's status, journal and error log, and pushed it as its own commit, `5f4126f`, with no fix in it.

The evidence was unambiguous: "Can't create test file ... (Errcode: 13 Permission denied)" on the data directory, although the directory showed the correct owner. The session recognised the fourth occurrence of a class of bug. `mysql` needs to be in the `lesta` group to walk into `/var/lib/lesta`. The grant lived only inside the tenant phase, which had once been the only one and always ran first. Now the control-plane phase ran first. The fix, `030f579`, moved the grant into the package-installation step that both phases share. All five CI jobs went green. Laravel was, for the first time, running its tests against real MariaDB.

14:30, phase 17: the oldest gap

There was one item that every phase since the third had listed under "disclosed": there was still no network connection between Laravel and a running agent. I chose to close it, using cron's execution history as the first thing that needed it.

The session's design pass found the problem green-field, and confirmed a rule from the installer contract. Laravel never executes host commands and never calls `.install`, directly or through any channel. That rules out any path into a node started by Laravel. The only one left is the agent calling home. The one hint in the code was a shell comment repeated in five installers, which promised "mTLS enrollment, heartbeat reporting".

The session put a real cost to me. Literal mutual TLS would need Laravel to run its own certificate authority and to configure its own web server for client certificates, infrastructure the project had never touched, since `.install` only ever sets up leaf nodes. I chose a bearer token over TLS instead. A long-lived credential per node, minted once at enrollment, stored as a one-way hash, the pattern Sanctum uses, with no new dependency. And I chose the wider build: a real persistent daemon, real enrollment, and one real use wired end to end, over a cron-only workaround that would have skipped both.

The plan drew one line before any code. This phase would not touch the existing one-shot provisioning dispatch for the six capabilities. It was to be deliberately deferred. The next phase would take it.

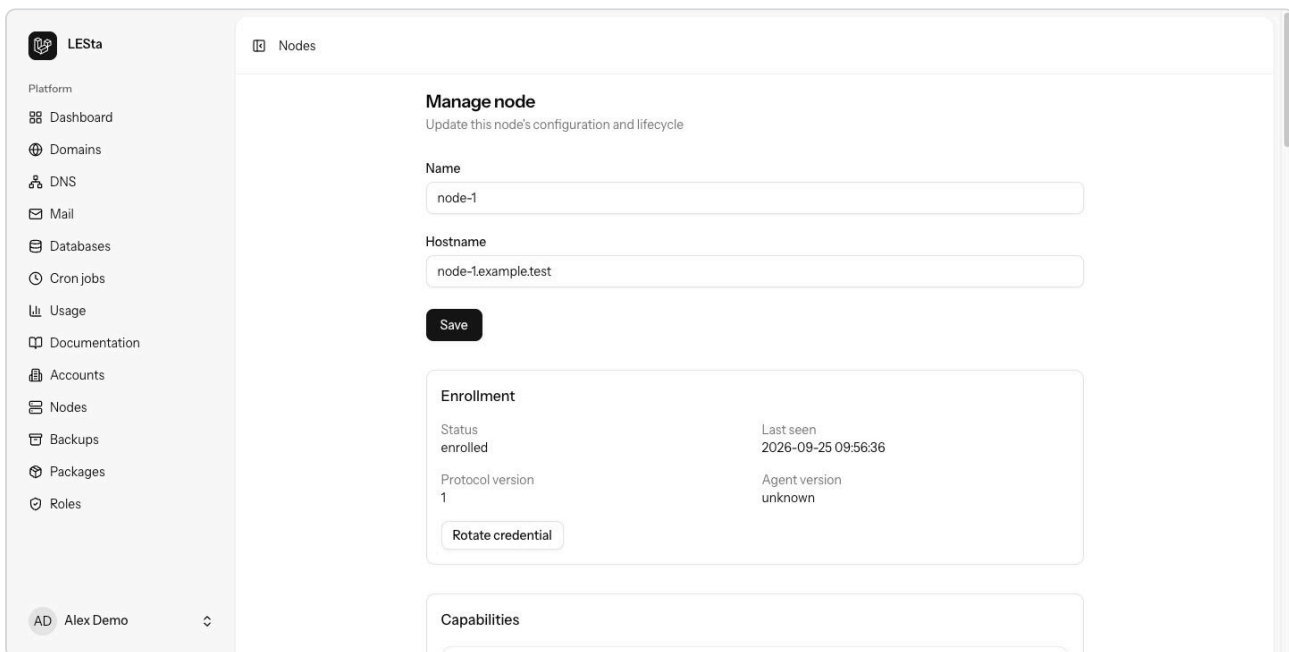
The agent built enrollment columns and an enum, an Artisan command to issue tokens, a Go `daemon` package with a ticker loop, exponential backoff with jitter and byte-offset tracking of cron logs, routes and middleware for the agent's calls, a new "Recent runs" section in the cron interface, and an installer for the daemon.

Reading the systemd unit against the installer's ownership lines, the session found a bug before any CI run. `/var/lib/lesta/agent` is created 0750, owned by root, group `lesta`, and the daemon's user `lesta-agent` is a member of the group, not the owner. It can read and traverse. It cannot write. The daemon's watermark file was directly inside that directory. The first time there was a new cron execution to report, writing the watermark would have failed.

The log counts this as the fifth occurrence of one bug: a worker identity that needs write access to a subpath of a parent it can only traverse. It had been `named` for BIND, `www-data` for Apache, `mysql` for the tenant instance, `mysql` for the control-plane instance, and now `lesta-agent`. This time it was found by reading and not by a failing job, because the installer's self-test only checks that the service is active and never exercises the write. A new `daemon-state` directory, owned by the daemon's own user, fixed it.

Before committing, the session ran the checks it always ran: format, static analysis, tests, ShellCheck, the contract validator. It also searched every touched file for em dashes. That is a rule from my global instructions, not something the project needed, and it is one of the smaller things that a session carries between prompts because the rule is written down.

CI ran green on all five jobs on the first attempt. Commit `b8f0546`, 14:30. The oldest disclosed gap was closed, for one use only.



A node after enrollment. Last seen is the agent's heartbeat. Rotate credential is the revocation path from chapter 21.

16:02, phase 18: live dispatch

The one use became all of them. The session offered to extend the new transport to live provisioning, upgrade and rollback, or per-account isolation for cron. I chose live dispatch. Every create, update, suspend, unsuspend and delete still went through a fake provisioner inside the process that never touched a node.

The research found that the outbox was real and tested, with idempotency keys, digests, duplicate guards and even a `deadline` column, computed on every dispatch and never enforced. Only two legs were missing: delivery to a node, and getting results back. Three forks came to me. Deliver pending operations on the heartbeat response, and do not build a separate poll endpoint. Wire all six capabilities at once, since the mechanism does not care which capability it carries. And add deadline reconciliation now, since the column existed for that purpose. I took the recommendation on all three.

The agent worked in an isolated git worktree, so the main tree stayed clean for review, and the session read each file against the plan before merging. It found one real bug. The request class validated `observed_state_version` with `min:1`. The wire schema allows a minimum of 0, and the daemon reports 0 when a dispatch itself fails, since no state was ever observed. Laravel's array-wildcard validation rejects the whole batch if one entry is invalid. So a single dispatch failure would have thrown away the results of every successful operation reported in the same heartbeat. It was fixed to `min:0`, with a regression test, since none of the agent's own tests had exercised it. Commit `db69f4d`, 16:02.

17:36, phase 19: a screen for the nodes

While offering the next choices the session noticed something that no phase had planned for. There was still no interface for managing nodes. Everything to do with a node was an Artisan command. I chose to build one.

The research found it only partly new: the node policies existed as stubs where every method returned false, and the suspend and unsuspend actions were done. It also found that nothing outside test factories ever created a capability row on a node, so a capabilities panel would stay empty. Both forks went to the fuller option: a minimal "add capability" action so the panel would not be decoration, and real node deletion gated on the node having no dependent rows, and not suspension alone.

The agent found a latent bug outside its own prompt. The global authorization bypass for provider administrators from chapter 7 only ever matched a model instance. Laravel passes a class name, not an instance, when no model exists yet, as it does for `viewAny` and `create`. So a provider admin's checks silently failed for every one of the six models that carried the marker. It had been like that since the provider was written, because no real interface had used it. The fix was one line: match class names too.

The commit `829d5cc` was at 17:36. Two small formatting commits followed within six minutes, `aea41a6` and `379734d`. The first replaced em-dash placeholders in the node edit page with plain text, the rule from my instructions again.

19:34, phase 20: installers that can go back

The last of the deferred items from phase 15 came next: upgrade and rollback. The research confirmed there was nothing. No installer compared the version on disk with the version to install. The exit code reserved for a failed rollback, 22, was used by no code path. This was where the session's first attempt to start the implementing agent hit a rate limit before a worktree existed, and it was relaunched.

The scope was cut in the direction of the narrower option, twice. Real generations and rollback for the one artifact that all six installers share and that already has a checksum: the vendored agent binary. Building it for nginx's, BIND's, Apache's and MariaDB's own packages would be a phase of its own, and apt has no clean side-by-side model. And automatic rollback on a failed health check right after install, which is what the contract's own wording, "failure leaves the old generation active", asks for. The commit is `5d1c383` at 19:34. Phase 21, the per-account users from chapter 13, followed at 20:38, and phase 22 at 21:17.

What a day like that is

There are two ways to read the day. In one, eight phases in twelve hours shows how fast the machine builds. The commit log supports that. In the other, the day is a list of eight moments where it stopped and asked me something, and I chose. Looking at the choices, the pattern is not hard to state. In phases 15, 16, 17 and 19 at least one of my choices widened the scope beyond the minimum. In phases 20 and 22 I picked the narrower option. There is no rule behind it that I could point to. The wide picks were where a gap was visible and would stay visible if I did not close it. The narrow ones were where the recommendation's reasoning, prove it on one case first, seemed right to me.

The log also names something else. Twice that day, a bug turned up from the same family: a process that runs as one user needs to write inside a directory that belongs to another. By the fifth occurrence, counting from BIND, the log had a name for it, an ownership-math bug class.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `1e673ab` , `46fb1c8` , `e25b1f3` , `bcd38b2` , `5f4126f` , `030f579` , `b8f0546` , `db69f4d` , `829d5cc` , `aea41a6` , `379734d` , `5d1c383` , `23e0550` .

Bundles, and going back

In chapter 8 I answered two questions about the nginx installer, Q21 and Q22, and each time I chose the option that kept the phase small. This chapter is where both came back. It covers the evening of 2026-09-05 and the whole of 2026-09-06, and the work is a single idea seen from four sides: a node should be installable without the internet, with the exact files you meant, and it should be possible to go back.

Phase 22: the answer I deferred in Q22

Q22 had asked how the nginx package should satisfy the contract's rule that every download is pinned, with a checksum. I had chosen a plain `apt-get install` against Ubuntu's own archive, verified by apt's signature chain. It kept that phase to the installer. The log had filed the other option, vendoring the packages, as a fast-follow.

On the evening of 2026-09-05, after per-account users, the session offered three things. I chose the release-bundle pipeline, the item deferred since phase 4.

The research found the gap total. Five of the six installers did a live, unpinned `apt-get install`. Only MariaDB, whose repository is third-party, treated supply-chain integrity as a first-class concern, and it too was fetched live. The contract, the README and the ADR all talked about signed bundles and offline installation. Nothing executable implemented any of it. The only artifact with a signature field in the whole repository, the vendored agent binary, was checked by SHA-256 alone, and the field was a label.

Two questions came to me, and I picked the narrower option on both, in the same style as phases 3, 9 and 20: prove it on one case first. Build and prove the pipeline on nginx alone, leaving the other four on live apt as a disclosed fast-follow. And use checksum-only integrity, extending the discipline already proven for the agent binary, and not build real GPG signing, which would give the project a private key to manage for the first time.

The agent worked in a worktree. It wrote `build-release.sh`, which uses `apt-get install --reinstall --download-only` to fetch nginx and its dependencies without installing anything and writes a manifest of every file's name, version, SHA-256 and an honest provenance label. It added an opt-in `--offline-bundle` flag to the nginx installer. Absent, nothing changes. Present, every `.deb` is verified before any change is made, and a mismatch fails closed with `EXIT_VERIFICATION_FAILURE`, an exit code reserved since the earliest phase and never produced by any code until now.

It added two scenarios to CI, and the way they prove themselves is what I would point at. The offline install runs with the runner's apt sources physically moved out of the way, and first runs a direct `apt-get install --download-only` and checks that it fails. That failure is the proof that the isolation is real

and not simulated. The second scenario corrupts the bytes of one vendored `.deb` and checks that the install is refused.

The agent also said something that is easy to skip past. Its own sandbox, macOS with no root and no `apt`, could not run a local dry run, and it said so, and relied on CI as the actual proof, and did not invent a local result.

It merged as `46fb1c8` at 21:17, and CI failed with a real bug in `build-release.sh`. `dpkg -i` said nginx depends on `nginx-common`, which was not installed, because the bundle did not contain it. The cause was in how `apt` behaves. `--reinstall --download-only` only fetches what it would need to change, and an earlier scenario in the same job had already installed nginx and `nginx-common`. `Apt` saw nothing to do and silently downloaded nothing. The session pointed out that this is not a CI artifact. Any real build host that already has some of nginx's dependencies would produce the same incomplete bundle without a warning. `38a3d51`, at 21:22, computed the dependency closure explicitly and fetched each package with `apt-get download`, which has no notion of install state. All five jobs went green.

Phase 23: nine round trips

The next afternoon, 2026-09-06, I chose to extend the bundle to the other four installers. Research showed that BIND, Apache and cron were structurally the same as nginx, one `apt-get install` against the stock archive, and MariaDB needed its third-party repository replicated on the build host first. It also showed that four of nginx's five bundle functions were already generic, and the session extracted them into `lib/offline-bundle.sh` first.

The agent added eight scenarios. The commit `5fe31b7` landed at 15:41, and there were then six commits in the following 41 minutes, each one fixing something that could only appear once a real offline install ran and something else ran after it. The log counts nine CI round trips. The five bugs it lists:

A real product bug in BIND's installer. It created the include directory, with the placeholder from chapter 9, after the package install. BIND's postinst re-validates its config on every real reinstall, which `dpkg -i` always is, and it failed on a directory that did not exist yet. A live `apt-get install` on an installed package does nothing and never reruns the postinst, so this never showed on the live path. A production node doing its first offline apply after adding the include line by hand, as the documentation says to, would have hit it. `b45584c`, 15:55.

A runner-only break. An offline install that touches `libtinfo6` can install a newer version than the pre-installed dev packages on GitHub's runner allow, and that breaks `apt` for whatever runs next in the job. A bare production node never has this conflict. It is a result of one CI job chaining many scenarios on one loaded runner. Fixed with a repeated `apt-get install -y -f` after each scenario. `59833ff` at 15:48, extended in `fef073`.

A methodology bug that had been there since phase 22. `apt-cache depends --recurse` is not dependency resolution. It walks every alternative in an OR group as if each were required, and had pulled `libpq5` into Apache's closure although the real, live Apache install never touched it. It was

replaced with `apt-get install --simulate` against an empty temporary dpkg status file, so that apt's own solver reasons as a fresh node would. That fixed this and phase 22's `nginx-common` bug in one better place. 0ab6203 , 16:02.

A missing flag on the new solver call. Without `--no-install-recommends` , the simulation could not know that a recommended virtual package, `systemd-timesyncd` , was already satisfied by `chrony` on this host, so it resolved it and installing it conflicted. db59443 .

dpkg -i does not respect Pre-Depends. `mariadb-server` Pre-Depends on `mysql-common` , which sorts after it in a glob, and the first pass failed before `mysql-common` was configured. Apt always computes a valid order on the live path, and raw `dpkg -i` has no such guarantee. A second `dpkg -i` pass, run only if the first fails, went into a shared function that all five installers use. This one is general and not CI-specific. 270f571 , 16:22.

The last run went green across all five jobs. All five apt-based installers could now install offline from a bundle with verified checksums.

Phase 24: Q21, automated

At 17:23 the next commit was 8b63e83 . This was Q21, the include line that the nginx installer refuses to write into `nginx.conf` , because the manifest marks the file as both read-only and refused. I had chosen a manual prerequisite. It had cost something since: the CI workflow repeated the manual insertion seven times.

Two questions came up. Nginx alone, or all three services that share the constraint, nginx, BIND and Apache? I chose all three. How to write to a file marked refused? An explicit opt-in subcommand, `install.sh --prepare-config` , or folding the write into `--apply` ? I chose the opt-in subcommand. That was Q21's other option.

The research found something that made this smaller than it looked. `refused_roots` and `read_only_roots` are purely advisory. The contract validator checks only the JSON shape of those fields. No shell code anywhere consults them. So the new subcommand needed only a documentation carve-out, no code one, which closed off what might have been a large design question.

Before merging, the session did something I now expect. It ran the new insertion function under real `dash` with real GNU `sed` against five hand-built synthetic config files, one for each return path: insertion, idempotent no-op, refusal when there is no `http {}` block, append mode, and a missing file. All five behaved as designed.

CI failed in the new scenario's own last step, not in the shipped logic. The step called `install.sh --apply --web-server nginx` , which correctly failed its `conflicting_package` preflight, since Apache was still installed on the shared runner from an earlier scenario. It was switched to `--web-server both` , which is designed to coexist with Apache. a516959 . Seven duplicated shell snippets in the CI file had been replaced with real, shipped and tested code.

Phase 25: going back

The last phase of the chapter was the one that asked for the most. Phase 20 had given the agent binary generations and automatic rollback, and deferred the operating system packages, since apt has no rollback of its own. Once a package is upgraded and the old `.deb` is gone from the cache and no longer served by the repository, the old version generally cannot be recovered without downloading it again.

The bundle changed that. On the offline path, LEStas holds the exact `.deb` files, and that is the one lever a real rollback needs. So I chose to scope it to the `--offline-bundle` path only, and the live path stays unchanged for good, and to build it for all four services, nginx, BIND, Apache and MariaDB, in one pass.

The design kept the last two generations under `/var/lib/lesta/<service>/bundle-generations/` and rolls back with `dpkg -i --force-downgrade --force-confold`, then a systemd restart. `--force-confold` was a deliberate choice over `--force-confnew`. LEStas's own state lives outside any package conffile, so a stock conffile after a downgrade never matters to LEStas, but `--force-confnew` could silently throw away an operator's own edits.

Halfway through, the session hit a rate limit and the agent stopped, with the core changes done and the CI scenarios not written. The session did not start over. It read the agent's partial diff, confirmed that the shared functions and the four installers' wiring were complete, and wrote the remaining scenarios itself. It made one mistake in doing that and the log says so plainly. An early edit went into the main checkout's `tests.yml` instead of the worktree's copy. It was caught at once by `git diff --stat main`, which showed 437 added lines against a checkout that should have been untouched. It checked that the diff was purely the accidental addition, discarded it, and redid it in the right place.

Then it exercised the shared helpers itself, under `dash`, with a path-substituted copy of the library so that the real functions, which hardcode `/var/lib/lesta`, could run against a writable `/tmp` path, and with stubs for the logging calls. It covered a fresh install, a no-op re-apply, a real swap of generations, and an idempotent re-apply that left `previous/` untouched. It merged as `75ddf46` at 20:49.

The log counts six CI round trips after that merge and lists four bugs, each of which only a real swap and rollback could show. I list those four.

1. **Setup (`e0358a6` , 20:56).** In the session's own scenario, `apt-get purge` deletes `nginx.conf` as a conffile, so `--prepare-config` failed with `nginx_conf_missing`. It used `apt-get remove`, as an earlier scenario already did.
2. **A general gap (`5d0af2e` , 21:07).** nginx's own postinst tries to start the service, and `nginx.service` runs `nginx -t` before starting. A broken new generation can therefore make `dpkg -i` fail before the installer's later health checks ever run. A CI scenario with a deliberately broken nginx binary showed it. By then the last good generation had already been moved to `previous/`, so rolling back was as justified as after a health failure. All four installers' `dpkg_install_failed` paths were wired to their rollback wrapper. And the rollback function now restores the `current/` bookkeeping from `previous/` afterwards, because the failed swap might have cleared it, and the next `--apply` would then have treated the node as having no prior generation.
3. **Assertions (`0f8d2c1`).** The four scenarios had to accept either exit 20, caught at `dpkg -i`, or exit 21, caught at the health check. Both mean a correctly refused upgrade and a successful rollback.

4. **systemd (73b4618 , 21:19)**. The broken generation had already tried and failed to start its service during its own install, and that engaged systemd's start-limit throttling. The rollback's restart, immediately after, failed even though the binary being restarted was the restored good one. CI showed exit 22, rollback failure, after a successful `dpkg -i restore`. `systemctl reset-failed` before each restart fixed it.

The last run went green on all five jobs. Nginx, BIND, Apache and MariaDB could now roll back automatically on their offline install path.

What only a real run shows

Every one of the bugs in this chapter needed a real run to find. The reasons are all things that exist only on a real machine: how apt decides what to download, how `dpkg` orders packages, what systemd does after a failed start, what a runner image already has installed. The plan said what to build, and the checks said whether it worked. The gaps between them were behaviours of the real system that no document described.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: 46fb1c8 , 73b4618 , 38a3d51 , 5fe31b7 , b45584c , 59833ff , fefd073 , 0ab6203 , db59443 , 270f571 , 8b63e83 , a516959 , 75ddf46 , e0358a6 , 5d0af2e , 0f8d2c1 .

The test that had never run

After 2026-09-06 21:19, the repository is silent for more than three days. The transcripts show no requests at all on 2026-09-07 and 09-09 and three on 09-08. The next commit is at 22:22 on Thursday 2026-09-10. That is 82 minutes after my weekly limit reset at 21:00 Swedish time, as chapter 12 described. The record does not say why.

22:22, phase 26: a test that skipped itself for nine days

The session offered four things after the pause: wire Pebble into the PHP CI job, decide how the statistics service's database account works, wire the permission tables into real authorization, or design one of the three verticals that were still unbuilt. I chose Pebble.

That was the gap disclosed at the end of chapter 11. `IssueAcmeCertificatePebbleTest` had existed since 2026-09-01 and had skipped itself in the PHP job, because the runner had no Go, no Pebble and no nginx. It ran for real only in the Go job. The session added three steps to the PHP job: a Go toolchain, Pebble pinned to `v2.5.0` and not `@latest`, because chapter 11's compatibility gap with newer Pebble challenge types is written at the top of the test file, and nginx from apt.

The first CI run went red on exactly the path that had just been switched on. The DNS-01 failure test threw a database error, `1406 Data too long`, writing to `web_domains.last_certificate_error`. The code in the catch block already cut a failure message to 500 characters, sized for a realistic ACME error body. The column had been created in phase 11 as a bare `string()`, which in Laravel is a 255-character varchar. A real Pebble rejection routinely exceeds 255. The SQL statement truncated the message before the PHP limit mattered. It was invisible for four phases, because the test that exercises that path had never actually run.

A new migration widened the column to 500, matching the existing cap exactly and not switching to `text`, since nothing suggested that the 500-character design had been wrong. It went out as `25209fa` at 22:22 and `df5423b` at 23:00.

A different problem came up in the middle of it, unrelated to the code. The first push failed with a 403 that named another of my accounts, although `gh auth status` showed the right one as active. The cause was that git on this machine takes its credentials from the macOS keychain helper, which held a stale token, and switching the active account in `gh` did nothing to it. `gh auth setup-git` re-registered `gh` as git's credential helper and the push went through. The log adds that it is worth remembering for any push that keeps failing after `gh auth switch`. It comes back in the chapter on the audit.

23:44, phase 27: an account nothing used yet

The statistics service does not exist. The backlog item was a design question open since phase 12: how does the statistics service get to read tenant databases? The requirement itself had been fixed on the first day, in the Foundations log: statistics reads tenant database state directly, through a dedicated least-privilege read-only account, in addition to web logs.

Two questions settled the design. The account is scoped per tenant database, not per account and not platform-wide, which matches "least privilege" and follows every other credential in the project being tied to one resource. And it lives inside `TenantDatabase` itself, not as a separate model. One resource, one lifecycle: create, rotate, suspend, unsuspend and delete manage both accounts together. The session implemented the account lifecycle for real inside the existing database capability and stopped short of the statistics service itself.

CI caught one thing at once. The MariaDB installer's own self-test payload had the old shape. It was fixed within nine minutes, `570badc` at 23:53.

07:34, phase 28: what a permission table is for

At 07:34 the next morning the log records a commit that changed how authorization worked. The tables for roles, permissions and their pivot had existed since phase 1, with a seeder holding a 16-entry catalog. Nothing in the application ever asked a permission question. Every admin check was a raw comparison on a role name, or the blanket bypass from chapter 7.

The research checked the catalog against what the policies actually did, and found the catalog stale. It lacked `packages.create` and `packages.delete`, the node create, delete and list permissions, and had the wrong name for the support view. Wired in naively, it would have locked phase 19's node interface out of creating and deleting nodes.

A real policy question came up. The catalog listed permissions for a provider admin to update, suspend, unsuspend and delete an arbitrary account, and to manage its memberships. No code path allowed any of that. All that existed was viewing, the read-only support view, and impersonation. Wiring the catalog in would grant new power. The session put it to me and I said yes, wire the full catalog.

The session designed the rest without asking. A corrected list of 21 permissions became the one source of truth for both the seeder and the test factory, so that there was no second copy to drift. The blanket bypass got an exclusion list, `Package` and `Node`, whose policies now decide everything by permission. `accounts.view` was renamed to `accounts.view_as_support`, and it deliberately has no plain `view` twin, since a permission-gated blanket view would have quietly bypassed the dedicated, audited support action from the first day.

One correctness gap was found before it shipped, and it was of the kind a bare-account unit test cannot show. Granting an admin permission to suspend or delete an account meant the cascade into that account's web domains and DNS zones now ran as that admin, and those two policies checked only the owner role. The first real account with any web domain would have thrown an authorization exception in the middle of the transaction. The session found it by writing an integration test with an admin

suspending an account that really owns a web domain. The existing tests had passed before and after the fix, since they never reached the cascade. It fixed it by letting the two policies recognise the same `accounts.*` permission that authorised the parent action, and not by extending the catalog, which would be the much larger, still-deferred revisit.

A test that asserted the old behaviour, that an admin cannot suspend an account without impersonating, was rewritten to assert the new one. The suite was 382 tests. It passed first time in CI. `eebce92`.

18:20, phase 29: the review before the mail

On Saturday 2026-09-12 the choice was between the three unbuilt verticals. The session noted that statistics was the easiest to build, since its account had just shipped, and that backups needed new external infrastructure. I chose mail. The session's own note called it the option flagged as the highest risk. The mail README says the capability "cannot be enabled until the mail threat model and operational readiness review pass". Two documents had named that gate since the first day, and no one had written it.

The session wrote a mail threat model and readiness review first. It rests on the facts captured in the capability matrix from chapter 6, including the MD5 hashes and the password overwritten with `SUSPENDED`, and on abuse patterns that need no legacy check to be real: open relay, backscatter and credential stuffing. It names a sixth trust boundary that no other capability had needed, between the node and the external mail ecosystem, since mail is the first capability that starts traffic toward third parties instead of only answering requests. It covers every gate item from the ADR and adds prohibitions. The strictest is that a DKIM private key must never reach the control plane, not even once, which is stronger than the general threat model's language for any other secret.

The session put it to me and did not assume sign-off, since the ADR treats it as a real gate. I approved it as written.

The build then mirrored DNS: `MailDomain` and `MailAccount`, with the domain as the one provisioning resource and every account embedded in its payload, since a real Exim configuration renders one artifact per domain. Two decisions came straight from the review. `dkim_enabled` is recorded as intent but cannot be set, because switching DKIM on with nothing yet enforcing it would be a lie in the interface. And the payload takes an explicit parameter pair for the one account whose password was just created or rotated, extending the database rule that only what was just generated is passed through and never decrypted.

20:52, phase 30: mail, and a checkmark that lied

I asked for mail and backups together. The session pointed out that the mail capability alone was the size of the BIND one, the largest so far, and that backups had no design at all. It proposed two separate units in the same session. And it asked, before spending real effort, whether this pass should cover only core routing and authentication, as the review recommended, or everything at once. I chose everything at once.

It began by installing real Exim 4.100 and Dovecot 2.4.5 on the laptop and proving the mechanics by hand.

- A real Exim relay rule refused a domain it does not host, "relay not permitted", and refused an unknown local part on a hosted domain. Along the way it found that Exim's `local_parts = lsearch;...` matches only the bare local part, never a full address, which would have let "sales" succeed on any domain. A condition that looks up the full address avoids that.
- A real 2048-bit RSA key and Exim's own DKIM options produced a valid signature header against a throwaway listener.
- `doveadm pw -s SHA512-CRYPT` did not work with this machine's libc. It chose `BLF-CRYPT`, bcrypt, which Dovecot implements itself. The log notes it is a good permanent choice on its own merits and not a local workaround.
- `sieve` compiled a vacation script and rejected a broken one.
- One limit was disclosed and not worked around. Dovecot 2.4's config format is an incompatible rewrite of 2.3's, and Ubuntu 24.04 and 26.04 ship 2.3. No 2.3 instance was available locally. That blocks testing Dovecot's own config parsing here, but not the capability's write surface, which is the password data file. Real delivery testing waits for CI.

The capability follows BIND's design, with a difference. Exim has no glob include, so the capability keeps aggregate lookup files, rewritten in full on every apply by scanning its own state directory for every sibling domain.

Two bugs came out of the local test harness before CI saw the code. A suspended account's password hash is left out of the rendered live file, as with every capability. But Laravel never re-sends a password on suspend or unsuspend, by the ADR's rule. So the first suspend would have destroyed the only copy of the hash and left unsuspend with nothing to restore. A durable credentials archive, kept apart from the rendered file and saved before activation, fixed it. The other was local only. Exim refuses to run as a daemon with DKIM options under a non-root `-C` invocation. In production Exim starts as root from systemd and never has this problem. DKIM signing was tested with a separate `-bS` run.

The checkmark

Then came the finding the chapter is named for. The first two CI pushes for this phase both went green, and both were wrong.

The session checked instead of trusting the checkmark. The Go job's second `go test ./...` step, after installing nginx, BIND, Apache, Exim, Dovecot and Pebble, reported every package as `(cached)`. It was replaying the first step's result: the run where nothing was installed and every real-capability test skipped itself. Go's test cache keys on the literal string value of the `PATH` variable. It does not look inside the directories on it. Every install step in the job put its binary into a directory already on `PATH`. The session reproduced it locally to be sure. `-count=1` forces a real run.

That exposed a second gap that the cache had hidden. With real execution restored, the disposable `named` harnesses for BIND and for the ACME Pebble test failed, because nothing in this job relaxed AppArmor before pointing `named` at a config under `/tmp`. The installer job already had that step. The

session copied it.

The log's summary is worth quoting in full: "this job's own 'nothing skips' real-capability test step has, in all likelihood, never genuinely exercised nginx/bind9/apache/Pebble-backed ACME here across this project's entire history". The other jobs, which run `go test ./...` once each, were never affected, since there was no earlier run in the same job to cache against. They are the project's real proof for those capabilities. The gap was in one step of one job.

The four commits were `db2314d` (the capability) at 20:52, `9601ac4` at 20:53 (installing Exim and Dovecot in CI), `528af18` at 21:07 (the cache fix), and `b6d47bd` at 21:19 (the AppArmor fix). Run `34713843002` was green with real, non-cached execution in every package, checked by elapsed time.

Two things were left out and disclosed. DKIM's public key is not published to DNS yet, because the result message had no channel for a capability to send back a piece of non-secret derived data, and that needed its own design. And the real spam and virus daemons and Dovecot's own structural config stay with the installer.

A pattern

Chapter 11 left a test skipping itself in one job. Chapter 16's finding is the same idea a level up. A green check does not say that the tests ran, and the way to find out is to read what the check actually did. It happened twice in the project, eleven days apart, in two different places, and both times the fix was to look at the elapsed time and the cached markers, not at the color.

SOURCES

- Commits, each at github.com/mikho/LESt/commit/ followed by the hash: `25209fa`, `b6d47bd`, `df5423b`, `570badc`, `eebce92`, `db2314d`, `9601ac4`, `528af18`, `5295833`, `5cab3bf`.

The day of thirty-eight commits, part one

Sunday 2026-09-13 is the busiest day in the record. The transcripts show 2,226 requests, 1.37 million output tokens and 10.4 hours of machine activity. The repository has 38 commits stamped that day, from 03:20 to 21:15. Phases 31 to 45 were built in those hours. This chapter takes the first half, up to the middle of the afternoon. The next takes the second.

The day did not start with a plan for fifteen phases. Each phase ended with the session offering a few choices, and I picked. Phase 31 is dated 2026-09-12 and 2026-09-13 in the log, so its first choices were made just before midnight.

03:20, phase 31: backups, and a hole in the protocol

Phase 30 had ended with a disclosed gap. A capability had no way to send derived, non-secret data back to the control plane: neither a DKIM public key nor a backup's metadata. The session asked whether to extend the protocol now or defer it again. I said now.

The change is small and touches everything. `ResultEnvelope` got an optional `data` field, and so did the JSON schema. The same field was threaded through Laravel: the result object, the action that completes an operation, the request validation, the controller and a new JSON column on `provisioning_operations`. Backups became the first real user of it. DKIM's use of it waited.

Four questions settled the backup design before any code.

- **Where to store it.** Local disk only for the first version. No S3-compatible remote target.
- **How to upload.** Through the Go agent, on the daemon path from phases 17 and 18, since a backup can take arbitrarily long to archive and encrypt and the one-shot protocol was never built for that.
- **Who can see it.** Provider admin only, one snapshot of a whole node. The session raised the reason before asking. No capability tags its own on-disk state by account, so any "your backup" for a tenant would hand back everyone's data, commingled. I confirmed provider admin only.
- **The rest** followed from precedent: a `Backup` model scoped to a node, with view, create and delete abilities and no update, since a backup is made or deleted and never edited. The key is encrypted at rest and follows the explicit-parameter rule from chapter 13.

The Go capability, `backup.encrypted-artifacts.v1`, finds which of the node's other capabilities have real rendered state, archives them into one gzip and tar stream, seals the whole thing with AES-256-GCM using the 32-byte key that Laravel generates, and writes it to

`/var/lib/lesta/backups/<uuid>.tar.enc` . The key is used directly, with no key derivation step, since `bin2hex(random_bytes(32))` is already the size that AES-256 needs. It was tested with real crypto and real archives and needs no external binary and no root.

Then came a limit that the session found while wiring the state roots and chose not to ship around. The `backs_up` list in the backup manifest had been carried over from a placeholder in phase 0, and it included both database capabilities. A raw copy of a live InnoDB data directory, with no consistent snapshot step, is not a backup. A restore could bring back something that was internally inconsistent. The session removed both from the list and scoped the first version to the configuration capabilities that can be safely copied as they are: nginx, Apache, BIND, mail and cron. It wrote down a real `mysqldump`-based capture for the databases as backlog. It was not silently missing.

The commit series started at 03:20 with `a204786` (the protocol field) and `e7c0382` (the capability).

03:40 to 07:27: two installers, and two things found by reading

The next choice was to close two installer gaps at once: mail and backups had real capabilities and no `install.sh` . The session did the installers first.

The backups installer is the thinnest yet: no package, no daemon, no firewall phase, no include line. It creates the artifacts directory. A stale entry in the manifest for a cache directory that nothing ever wrote to was removed. It was green on the first real push, `31b2080` at 03:40.

Writing the mail installer turned up a bug before it shipped. The real Exim daemon, running as the `Debian-exim` user, has to read the DKIM private keys that the capability generates. They were written mode 0600, owned by the agent's own user. Signing happens in a different process. Every real signing attempt would have failed with a permission error. Phase 30's tests never caught it, because they ran `openssl` and `exim` directly against the file, and never through a second, unprivileged identity. The mode became 0640, and the installer adds `Debian-exim` to the `lesta` group, as BIND does for `named` . `649a170` , 03:49.

A second constraint showed up before it was written wrongly. The capability had a separate LESto-owned Exim file, included from the distribution's main config, by analogy with nginx and BIND. Exim's format does not allow that. It has exactly one `begin acl` , one `begin routers` , one `begin transports` and one `begin authenticators` section, and Debian's split config already defines all four, so a second file gives duplicate-section errors and not a working config. The fix was to point at `/etc/exim4/exim4.conf` itself and have the installer take complete ownership of it, using Debian's own switch for manual control, `dc_use_split_config='false'` . `b5fe4d0` , 03:55.

The mail server needs a TLS certificate for its own hostname, and nothing in the project issued one for a standalone hostname. The session asked me, and did not invent new integration. The installer takes a required `--mail-hostname` flag and fails closed unless the certificate files already exist, which they do once the operator creates a web domain for that hostname and lets the phase 11 machinery issue it. No new code.

Two more bugs came from CI in the next hours. The installer wrote its own config after `apt-get install` and not before, an ordering mistake of the kind BIND had already taught (`0c73647` , 04:15). And a preflight rejected a healthy re-apply because its own Dovecot was listening on port 993 (`61cc39c` , 07:27).

10:01, phase 33: nothing was logging traffic

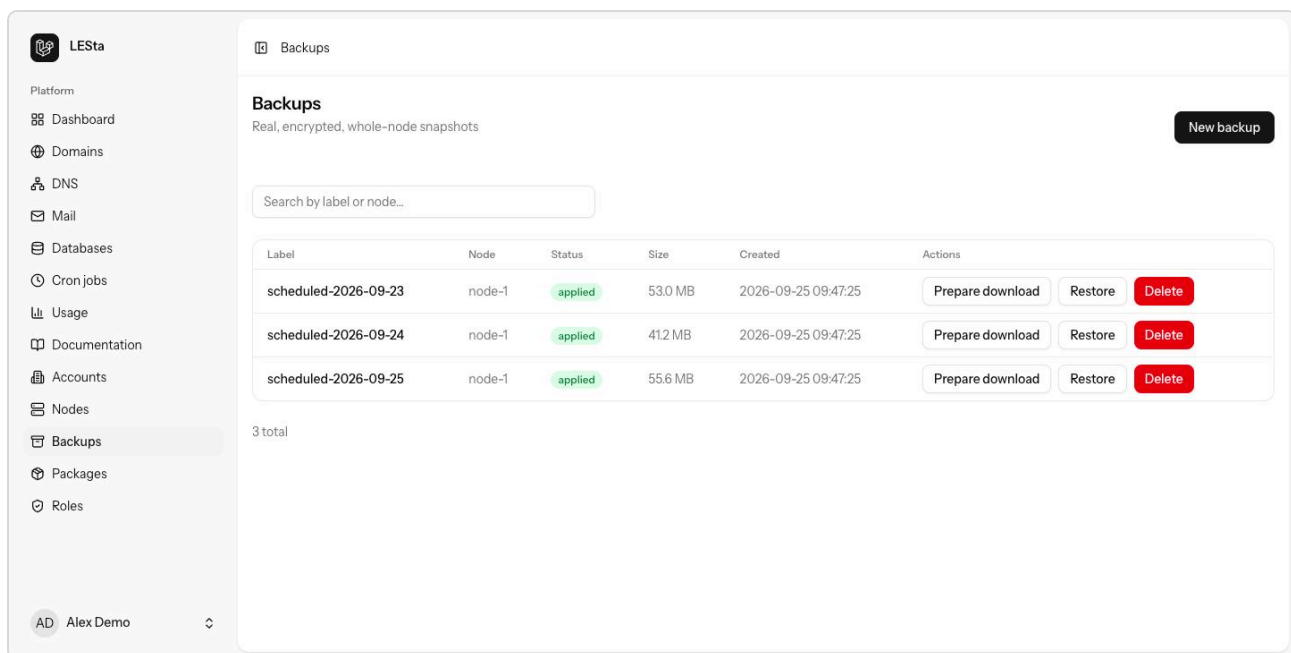
The statistics vertical was the last one with no design. Scoping it turned up a prerequisite nobody had noticed. Neither nginx nor Apache was configured to log access at all. Bandwidth and request counts per domain could not be collected from the first day.

The session added an `access_log` and `CustomLog` directive to every vhost template in both capabilities, written to a LESt-a-owned log directory and named by `resource_id` , never by the domain string, which a tenant controls. It gives each domain unambiguous attribution with no parsing of Host headers. Both installers create the directory before their config validation, the same ordering lesson as the mail installer's config file, and install a `copytruncate` logrotate policy, chosen because it never has to signal a web server to reopen a file.

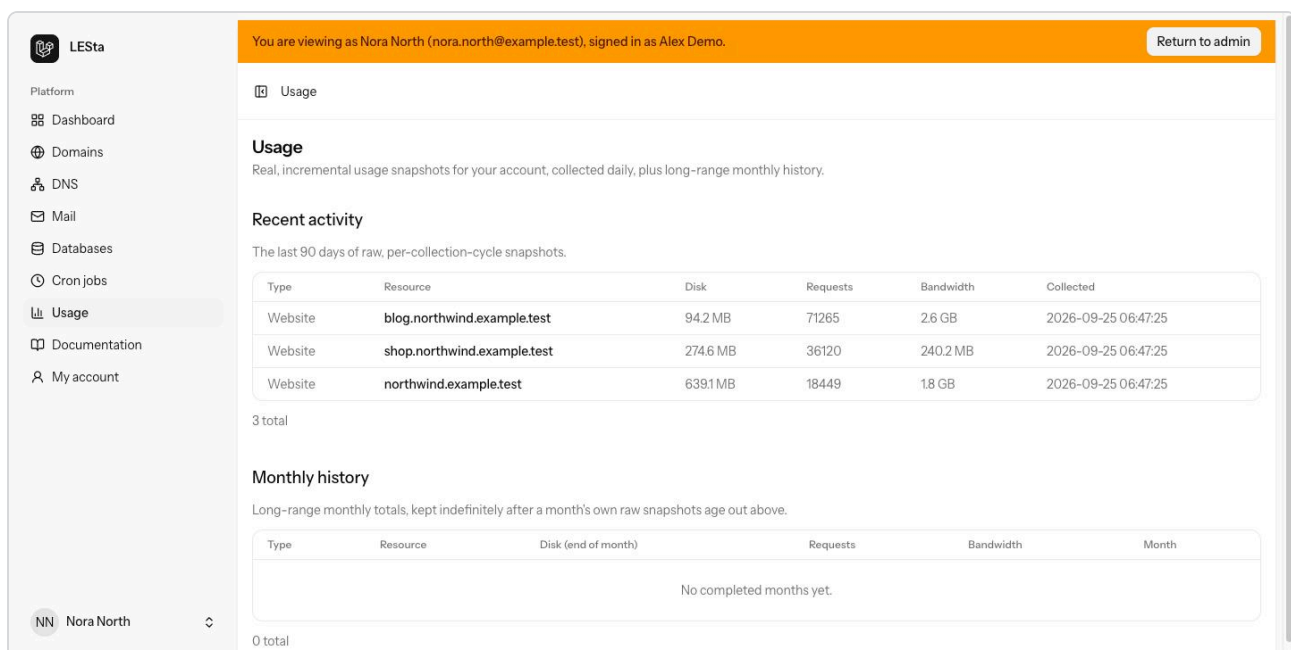
The protocol shape needed a decision. Usage collection does not fit the create, update and delete pattern. It is read-only telemetry. The session confirmed with me before coding: one `observe` operation per node per day, with a `MetricsCollection` resource as a purely internal anchor, a payload that Laravel builds from its own database, since the Go agent has no independent way to discover mail accounts and databases on the node, and results reported per resource UUID through the new `data` field. The Go capability sums real maildirs, queries real database sizes through each database's own stats account from phase 27, and parses access logs with a saved offset, tested against a simulated rotation.

Commits `a2b736b` at 10:01 (logging), `cdec34` at 10:18 (the capability), `e27756c` at 10:33 (a second test configuration that had missed the new log directory), `d06bae4` at 10:34 (the Laravel side) and `d5acef4` at 10:46 (Pint's strict import rules in the new tests). Then `b7b4539` at 11:44, phase 34, the admin interface for backups and usage.

For that interface, the session found a bug that had nothing to do with it. Regenerating the Wayfinder route helpers had been done without a `--with-form` flag, and every existing page's `.form()` call broke under a clean TypeScript check. The log calls it a pre-existing, project-wide bug. There was no browser tooling in that session, so the pages were checked with HTTP-level Inertia assertion tests instead, and I accepted that.



Backups as an admin sees them. Restore is the phase 45 button.



Usage snapshots from phase 33, with the monthly history table that phase 43 added.

12:25, phase 35: three bugs before a single button

The next choice was an interface I had asked for in plain words: "user ui". It meant an admin page for one account, showing its memberships, web domains, mail domains, databases and cron jobs, with the actions of its lifecycle.

Before wiring anything, the session traced what suspend and delete already did, and reproduced three bugs with `tinker` scripts. None was hypothetical.

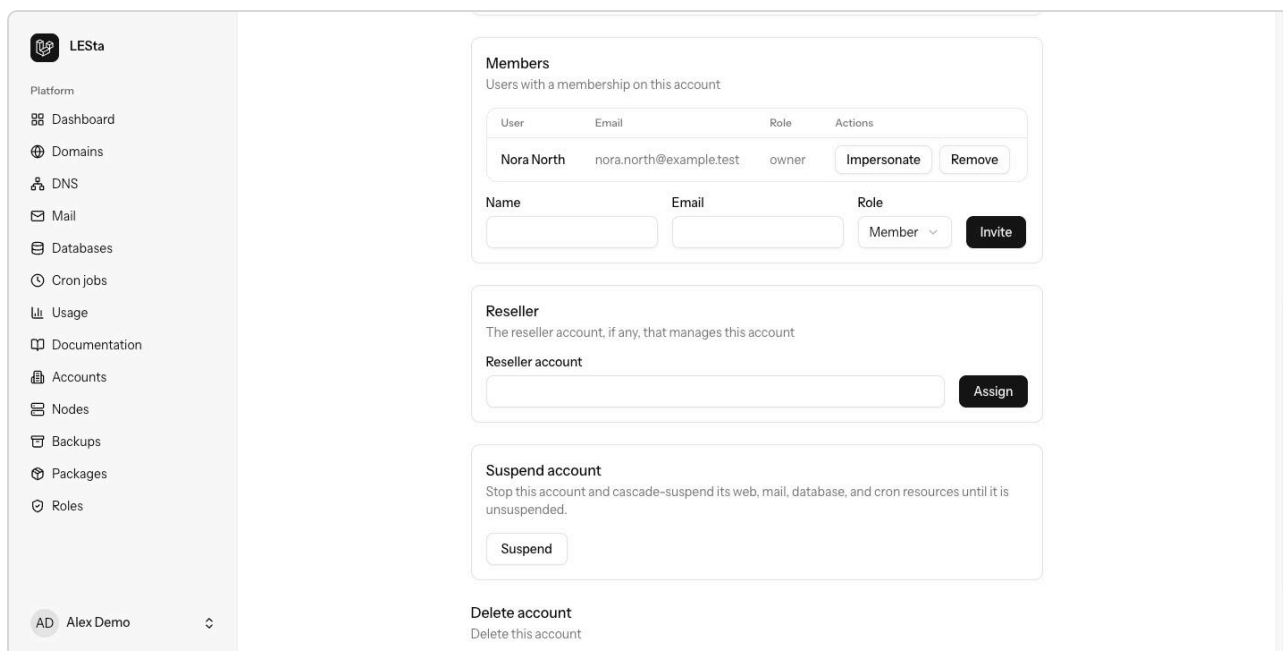
1. **DeleteAccount** **never deleted the account's databases and cron jobs**. Both tables' foreign keys to accounts are `restrictOnDelete`. Deleting any account that still had either threw a raw `QueryException` about a foreign key violation, when the other resource types got a clean transactional cleanup.
2. **SuspendAccount** and **UnsuspendAccount** **never touched databases or cron jobs**. A suspended account's database and cron jobs stayed fully active. That is a security and business-logic gap, not only a missing audit line.
3. **The two policies had no `accounts.*` permission fallback**, the exact bug from chapter 16 that had been fixed for web domains, DNS and mail and never applied to these two. Fixes 1 and 2 would have thrown an authorization error mid-transaction the first time a provider admin, not the owner, suspended an account with a database.

The fixes went in first as a separate commit, `af5221d` at 12:25, ahead of the page. The page followed at 12:39, `bed2e63`, and with it the read-only support action from chapter 7, `ViewAccountAsSupport`, got a caller for the first time. It was also the first time `Account` got a route key, since no route had ever bound `{account}`.

The screenshot shows the LESta web interface. On the left is a sidebar with a 'Platform' menu containing links to Dashboard, Domains, DNS, Mail, Databases, Cron jobs, Usage, Documentation, Accounts, Nodes, Backups, Packages, and Roles. At the bottom of the sidebar is a user profile for 'AD Alex Demo'. The main content area is titled 'Accounts' and shows the details for 'Northwind Hosting'. It includes input fields for 'Name' (filled with 'Northwind Hosting') and 'Contact email' (filled with 'nora.north@example.test'). There is a 'Package' dropdown menu set to 'Business' and a 'Save' button. Below these is a 'Resources' section with a 'View usage' link. The resources are listed in a table:

Resources	
Web domains	Mail domains
3	1
Tenant databases	Cron jobs
1	1
Created 2026-09-25	

The provider-admin account page of phase 35.



Members, reseller assignment and lifecycle actions on the same page. Impersonate arrives in phase 53.

13:22, phase 36: a TXT record that could not fit

The next choice was DKIM DNS publication, closing the gap chapter 16 disclosed. The session researched before writing code, and found a bug in BIND, not in DKIM.

The TXT rendering in the BIND capability wrapped a value in exactly one quoted string. DNS limits one character-string to 255 bytes. A DKIM public key for RSA-2048 is around 400 bytes. Publishing one would have either failed `named-checkconf -z` or served a truncated key, and every message the domain sent would have failed DKIM verification. The bug was in every TXT record the project ever produced. It had simply never met a value past 255 bytes.

The fix splits a value into adjacent quoted strings, which BIND supports natively. It was proven against a real `named` process: a 398-byte TXT was accepted by `named-checkconf -z`, and a real `net.Resolver` query returned it exactly, since Go's resolver joins the strings with no separator, which the session confirmed against Go's own source.

The mail capability now reads its public key back off disk, never the private one, and reports `{selector, public_key}` in the new `data` field. A new `PublishesDkimDnsRecord` action, the fourth hook in the completion class, publishes `<selector>._domainkey.<domain>` as a real DNS record on the matching zone of the same account. If no zone matches, it logs and does not throw. The session had checked and found that nothing links a DNS zone to a mail domain. It deliberately bypasses the account's own DNS record quota for this one record the infrastructure requires. 24d5217, 13:22.

14:13, phase 37: how much of a restore is safe

I chose four backlog items in one go: a restore command, an admin usage view, database backups by `mysqldump`, and offline bundles for the mail and backup installers. This entry covers the first.

Before designing it, the session read what an archive contains. The findings were bad enough that it brought them back to me. The backup only archives each capability's state root, which is generation bookkeeping. It never copies the live directories that nginx, BIND, Apache, cron, Exim and Dovecot read at runtime. Untarring the archive over the node would not change what is being served until some new reconciliation re-derived live config from the restored state. It also observed that the generation store had no rule for numbers going backward, that the protocol had no non-lifecycle verb like "restore", and that a backup is node-wide and not consistent with whatever has been created or deleted since.

It offered three scopes: decrypt and download only, full automated restore for nginx as a proof, or full restore across all five capabilities. I chose decrypt and download, the one that sidesteps all of it. The Go capability gained an `Observe` that reports the sealed bytes back, and a new hook decrypts them with the already-stored key and serves a single-use, time-limited plaintext copy. `cb0eabe` , 14:13.

Four minutes later the second item went in, `6e381dc` : an `?account=` filter on the usage view. The permission for a provider admin to see usage across accounts, `usage.view_any` , had existed since phase 33 and nothing had used it.

Half a day

By 14:17, phases 31 to 38 were done, on top of everything in the earlier chapters. The next chapter covers the second half of the day: the rest of the four items, a virus scanner that could not read its own files, a spam filter that called everything spam, and the restore that had been declined at 14:13.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `a204786` , `6e381dc` , `e7c0382` , `31b2080` , `649a170` , `b5fe4d0` , `0c73647` , `61cc39c` , `a2b736b` , `cdec34` , `e27756c` , `d06bae4` , `d5acef4` , `b7b4539` , `af5221d` , `bed2e63` , `24d5217` , `cb0eabe` , `be9d278` .

The day of thirty-eight commits, part two

This is the second half of Sunday 2026-09-13. It starts where chapter 17 stopped, with the third of the four backlog items I had chosen at 14:13. It ends with the log stating that the backlog is empty.

14:33, phase 39: a backup of a database

Phase 31 had refused to back up the two database capabilities, since a raw copy of a live InnoDB directory is not a backup. Phase 39 was the safe way.

Both database instances already have fixed unix sockets that root can authenticate against with no password, using `unix_socket` auth. The installer's own health checks used the same mechanism. The one-shot agent binary that runs every real capability always runs as root in every path built so far, so there was no privilege gap to solve. The backup capability could run a real `mariadb-dump --all-databases --single-transaction --quick` against each socket, with no tenant credentials sent or even needed. `--single-transaction` is the non-locking InnoDB snapshot that phase 31's concern had been waiting on.

It dumps every schema on each instance, system schemas included, and does not enumerate individual tenant databases. That needed no new plumbing, since the generation store has no way to list all resource IDs, and it matches the capability's existing scope of a whole-node snapshot. It was proven end to end against a real disposable MariaDB. `b678b13`, 14:33.

14:43, phase 40: one installer needed it, one did not

The fourth item was offline bundles for the mail and backup installers. The session checked before starting and found that the backup installer has nothing to bundle. Its own comment says there is no external binary to install, since the capability is pure Go. Even after phase 39 made it run `mariadb-dump`, that binary belongs to the MariaDB installer. The scope shrank to mail alone. The mail bundle follows MariaDB's multi-package pattern, with all four mail packages kept and rolled back together as one generation. `c407bec`, 14:43.

Then four commits in 40 minutes went on the last CI scenario, number 26, which built the mail bundle. Each of them was a guess disproved by CI, and the last was a diagnostic.

- `b44b713`, 14:57. A first guess: the runner had a PGDG entry in `sources.list.d`. Wrong.
- `cce7c1`, 15:10. A second guess: rebuild the apt list cache. Wrong.
- `fc3c4db`, 15:22. Not a fix. A real diagnostic step in place of a third blind guess.

- **b10d707 , 15:36.** The diagnostic found the cause. The runner's own pre-installed PostgreSQL development tooling left a `libpq5` in its `dpkg` status that pointed to a version no repository could fetch. It was the third time in the project that a runner-image quirk of this kind had to be fixed, after `libtinfo6` twice. The scenario now purges it.

The log's summary of the episode is that two wrong guesses were disproved cleanly by real CI before a diagnostic found the cause. It is the same rule as chapter 10, applied faster.

16:17, phase 42: backups on a schedule

Scheduled backups took one commit. A `backups_scheduled` flag on `Node`, deliberately not mass-assignable, so it can only be set by a dedicated action. A `CreateBackup::handleSystemInitiated` that mirrors the one for deleting backups: it skips authorization and records an audit event with no actor. A daily `backups:create-scheduled` command for every node that is switched on, not suspended, and has an active backup capability.

One bug is an Eloquent gotcha. `Node::factory()->create()` returns an in-memory model with only what was passed plus the primary key. A default computed by the database is not filled in without a `refresh()`. A test that asserted the default directly on that model was trivially true, whatever the column default actually was. The fix was to assert on the refreshed model. The same mistake comes back in phase 44.

The other was already a known one. `pint --dirty` passed locally and CI's `pint --test` failed on an inline fully qualified class name. The log had recorded this divergence in phase 33 and treats it now as a standing thing to check for by hand. `d2b922a` at 16:17 and the follow-up `35456d4` at 16:22.

16:18, phase 41: a virus scanner that could not read the file

Antivirus and antispam for mail was the largest of the afternoon's items, and one commit, `d7ccd4c` at 16:18, was followed by seven more before CI was green.

The design is in the Exim configuration, which the installer already owns. A per-message accumulator, `$acl_m_av` and `$acl_m_spam`, is set by the per-recipient step whenever any accepted recipient's domain opted in, because Exim has no single `$domain` at DATA time when recipients span several hosted domains. The scanning conditions use `/defer_ok`, so a briefly unreachable scanner defers the message with a retried 4xx and neither rejects good mail nor accepts it unscanned. That has a sharp edge, and the log says so.

Exim never connects eagerly to the scanner or the spam daemon. A successful `exim4 -bv` or restart says nothing about them. So two protocol-level health probes were added: a raw `PING` over the clamd Unix socket and a raw `PING SPAMC/1.5` over spamd's port. The proof in CI is the strongest test in the project so far. It creates a domain with both scans switched on and opens a raw SMTP connection. It sends the standard EICAR test string and expects a virus rejection. It sends the GTUBE string and expects a spam rejection. It sends a clean message and expects acceptance, to prove that the ACL does not simply reject everything.

Six bugs followed, all from CI and none from local testing.

1. **ShellCheck's version (6f50757 , 16:22).** CI pins 0.9.0, older than the local copy, and it flagged the probe functions as unreachable because it cannot follow dynamic function-name dispatch. Explicit disable directives fixed it. The lesson written into the log includes a comedy: a comment line that starts with the literal word "shellcheck" is parsed as a malformed directive.
2. **spamassassin.service does not exist on Ubuntu 24.04 (deb49b1 , 16:33).** The package installs fine and ships its unit under a different name. The installer now asks `systemctl list-unit-files` for `spamd.service` or `spamassassin.service` and does not hardcode either.
3. **spam = Debian-exim:true classified every message as spam (8a0f593 , 16:38).** The first version of the rule used that exact string, taken from an online example without checking what it means. Appending `:true` to the user in Exim's `spam` condition makes it always true, whatever the real score is. Its purpose is to tag headers unconditionally in a `warn` statement, not to classify. The bug could not show until the new proof step ran for the first time, since every earlier CI attempt had failed before reaching it. The fix was `spam = Debian-exim/defer_ok`, matching the malware condition's pattern.
4. **clamd could not read the message it was asked to scan (be04fa7 , 16:48).** Over a Unix socket, clamd is handed the path of a spool file, and not the content. clamd runs as `clamav`, and Exim's spool belongs to `Debian-exim`. And `/defer_ok` treats a scan error, including a permission error, as if no virus was found. So EICAR sailed through with a plain `250 OK` and no visible failure. This is a decades-old integration gap, Debian bug 250335 and others, and not specific to this project. The fix was the reverse group grant, `usermod -aG Debian-exim clamav`, and `AllowSupplementaryGroups true` in `clamd.conf`, because clamd drops all its supplementary groups at startup by default. It is another case of the ownership class from chapter 14, with a worse symptom: the scanner reported success without having scanned.
5. **A fresh offline dpkg -i does not refresh systemd's unit cache (6ff43cb , 17:00).** The session first thought this explained a remaining failure. It did not. It added `systemctl daemon-reload` to the shared function anyway, because it closed a real gap, and said so.
6. **spamd is a separate Ubuntu package (7d9b658 , 17:29).** Installing `spamassassin` pulls in `spamd` neither by Depends nor by Recommends. The live install had never named it, and earlier CI runs found a working `spamd.service` regardless, which the log calls unexplained and explicitly not relied upon. The offline path excludes Recommends on purpose, so it never had a chance to include it. Naming `spamd` explicitly in every package list closed it. A diagnostic commit, `1703e11` at 17:14, sat between the fifth and the sixth.

CI ended green on all five jobs, with the EICAR and GTUBE step and two offline scenarios that now vendor all seven mail packages.

19:07, phase 43: a date that was not a date

Usage history had a 90-day limit on the raw snapshots. Phase 43 added monthly rollups that survive past it: one row per resource per calendar month, with the disk value as the last reading of the month and the request and byte counts as sums. A column stays null when every raw snapshot left it null, and

does not collapse to a misleading zero. A daily command aggregates every finished month, scheduled right before the pruning command, and is safe to run every day.

Its bug was caught locally before CI. Eloquent's plain `date` cast stores a full `2026-08-01 00:00:00` string, not `2026-08-01`. The command upserts on a key that includes the period. A search array compared a bare date string against a stored datetime string, never matched, and would have inserted a duplicate row on every daily run until it hit the table's unique constraint. The cast `date:Y-m-d` stores the bare date. The session found it with a sqlite run, since the local MariaDB test database was not reachable with the credentials that `phpunit.xml` expects on this machine, which the log calls a pre-existing environment gap. CI's own MariaDB service remained the authority and was green. `33db047`.

20:02, phase 44: rotating a key without breaking mail

DKIM keys should rotate, and a naive rotation breaks mail. If the DNS record for the new key is not visible yet when the mail server starts signing with it, receivers fail every message. Two questions came before any code.

How careful should the rotation be about DNS propagation? A one-shot swap, or the three-phase state machine that the existing code comment had already promised: publish the new key's record, wait, switch signing, wait, and only then retire the old record and key. I chose the three-phase version. And how should it be triggered? The session had found that no HTTP controller, route or page existed for `MailDomain` anywhere in the app. Only the model, actions and policy existed, and tests exercised them. It offered an on-demand Artisan command against automatic rotation on a schedule. I chose automatic, which removed any need for an interface in this phase.

The design rotates every 90 days per domain and computes each wait from reality. A new class looks up the domain's own DNS zone and uses its configured TTL, with a one-hour floor, and falls back to a conservative 24 hours when the control plane does not manage the domain's DNS. Three scheduled commands run the phases: start, promote and retire, and a domain can be in only one rotation at a time. On the Go side the capability stores one keypair per domain and selector, and Exim's `dkim_selector` became a per-domain lookup, like the key path already was.

The gotcha from phase 42 came back in `CreateMailDomain`: the factory result did not reflect a computed default. It was caught locally. `23d9d64`, 20:02.

21:15, phase 45: the restore, reframed

The last item was the one that phase 37 had declined at 14:13, and it came back with a different shape. Two rounds of research before any code overturned most of the framing.

The claim that the generation store forbids numbers moving backward was not true. `Activate` has no such check, and nginx's own recovery already re-activates old content under a new number. The missing protocol verb was a small additive change, since dispatch is per capability and every capability already returns `unsupported_operation` for a verb it does not know. And the question about capturing live state had been the wrong one. LESt's database is the source of truth. A node's configuration, the

nginx and Apache vhosts, BIND zone files and cron sidecars, can always be rebuilt losslessly from it by a normal update. Only two things on a node cannot be rebuilt: the mail that people have received, and each database's own table data.

So there are two mechanisms of the right size. **Resync** re-sends a normal update for every web domain, DNS zone, cron job and mail domain on a node. It has no new verb and reuses the existing dispatch.

Restore is a new `restore` verb for the backup capability alone. It puts back the mail directories and each database's dump from an archive already on that node's disk. The archive never leaves the node. Only the decryption key travels down, the reverse of the download flow.

A second correction came mid-design, and I agreed to it. Mail passwords need no forced reset during resync. The mail account's password is stored with Laravel's recoverable `encrypted` cast, not a one-way hash. The agent's own sidecar redacts it, a different, node-local copy, but Laravel's database always has the current plaintext. Resync reuses each account's known password, and a mailbox owner notices nothing.

The design is one Restore button, gated by a new `backups.restore` permission, and a sixth hook in the completion class that triggers a resync when the restore completes. On the Go side, `Decrypt`, which had been written in phase 31 and unused since, finally got a caller. Mail directories come back through a per-domain staged directory swap and never as a blind overwrite. Each database dump pipes back into its live socket, the reverse of the dump.

The test is a real Go end-to-end run against a real disposable MariaDB: a seeded row and real maildir bytes, a simulated disaster with the database dropped and live mail wiped, then a restore. Both came back. The scope limits are disclosed. Only resources that still exist in Laravel today are resynced, so nothing deleted after the backup returns. And a restore only targets the node the backup was taken on. `94b61ca`, 21:15.

The last line of the phase's log is this: "This was the last remaining item. No open backlog candidates remain in the feature request list as of this phase."

What a day of this was

The day's commits run from 03:20 to 21:15. Fifteen phases went in. A mail scanner reported "clean" when it had not scanned. A DNS limit from 1987 turned up in a feature written in 2026. Two guesses at a runner problem were disproved, and the third step was to stop guessing.

The transcripts say the machine was active for 10.4 hours. By the log, what I did on the day was choose from the options that came up, each with a stated cost, and answer the questions that could not be settled by research.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `b678b13`, `94b61ca`, `c407bec`, `b44b713`, `cce7cf1`, `fc3c4db`, `b10d707`, `d2b922a`, `35456d4`, `d7ccd4c`, `6f50757`, `deb49b1`, `8a0f593`,

be04fa7, 6ff43cb, 7d9b658, 1703e11, 33db047, 23d9d64 .

PART FIVE

A product, not a demo

What Vesta never had

After the backlog was declared empty on 2026-09-13, the questions changed. Until then the sessions had mostly been offering what to build next from a list. From 2026-09-14 the phases started from requests that I wrote, and they began with looking at the product as a user would.

Phase 46: the premise that did not survive

I asked a question about the frontend: what was there for users, resellers and admins? I framed it as something Vesta had, and that the rewrite should keep. The session went to look before planning anything. What it found is written in the log's second heading: "The VestaCP premise didn't survive research."

Vesta's reseller support was a stub behind a paid plugin. The function that was meant to find an account's owner was dead code. Its role model was binary, admin or user. There was no multi-node concept at all, since Vesta was a single-server panel. There was nothing to preserve. Both reseller accounts and node-level administration were new, and were built on their own merits. The original plan had said reseller nesting was reserved until the account and membership model was proven, and by phases 28 and 35 the model was.

Two rounds of questions shaped it, and after the plan was first drafted I made a third correction. On node administration, I chose infrastructure only: no reach into tenant resources, not even on a node the admin manages. For resellers, I chose a special account that owns other accounts, reusing the shape that already existed instead of a new top-level model. And then I cut something from the plan. It had granularity per capability and per service for node admins. I said one grant means full admin of the whole node, nothing narrower. That reduced `NodeAdminGrant` to a plain user and node pair with no role and no capability column.

Part one: a tier below the platform admin

A new `node_admin_grants` table, and not a third meaning added to `memberships.account_id`. The node policies gained a fallback for a delegated admin, and the fleet list for such a person shows only their own nodes.

A failing test caught a privilege escalation. `GrantNodeAdmin` and `RevokeNodeAdminGrant` were first authorised through `NodePolicy::update`. After the new fallback, a delegated node admin passes that ability for their own node. They could therefore grant or revoke further admins of their own node, which is the chain the plan prohibited. It was fixed by checking `nodes.update` as a direct permission in both actions, deliberately bypassing the widened ability.

A second mistake was in process, and the log says it plainly. A file, `NodeCapabilityPolicy.php`, was overwritten without reading it first, on the assumption from incomplete research that it was a new file. It was an existing all-false stub from phase 19. `git status` showed `M` where `??` was expected. The file was restored to its original shape and only the needed changes were kept. `5c77cdf`, 2026-09-14 09:38.

Part two: the reseller

An account can now point at another account through a nullable `reseller_account_id`. The whole feature rides on one central change: `User::hasAccountRole()` gains a fallback that recurses onto the reseller account when a direct membership check fails. Every policy that already called it got reseller support with no change. A new test proved that the owner of a reseller account can manage a managed account's web domain with no edit to that policy file.

Nine view checks did not go through `hasAccountRole()`. They queried memberships directly, since there is no role name for "any role". A new `hasAnyAccountMembership()` with the same shape replaced all nine.

The nesting limit is not in the recursion. Neither method limits its own depth. The one-level rule holds only because the assignment action refuses to create data that would need a second hop. A candidate reseller cannot itself be managed, and a candidate managed account cannot itself manage others.

The log is careful about what was not built: no account creation flow of any kind. Nothing in the app created an account for anyone. Accounts came out of band, from factories and seeders. `8859c84`, 13:25.

Phase 47: I correct the premise

The next question was whether to build self-service account creation. I answered that there is no public self-registration for a hosting account in this product at all. Administrators and resellers create accounts. Resellers create hosting accounts, and admins create resellers or hosting accounts. Reseller-initiated creation would wait: "let's start first version without resellers, as decided earlier".

In the same answer I pointed at two gaps that nobody had disclosed. No admin path created a hosting account or a user for one. And a regular hosting user could not see or manage their own account, since `AccountController::show()` demanded the admin-only support permission, while `AccountPolicy::view()`, already reseller-aware, had no callers at all.

I also set a security requirement, and it is a good example of what the requirement did when it met the code. The `{account}` value in the URL must be a random 12-character string, never derived from the row's own sequential ID, so that no one can enumerate every account on the server. And a logged-out request for `/accounts/{account}` must reveal nothing about whether that account exists.

The session verified it against the real framework and did not assume. In Laravel 13.26.1, the `auth` middleware is prioritised ahead of route-model binding. A guest hitting any `/accounts/{account}` URL, real or garbage, is redirected to login before binding runs. No anonymous enumeration gap existed. But

it found a gap for a logged-in stranger. A nonexistent token got a 404 at the binding layer, and a real but inaccessible one reached the controller and threw a 403, a signal that says the account exists. The controller now returns 404 whenever both abilities are denied.

A new `public_id` column replaced `uuid` as the route key and display identifier. It could have reused `uuid`, but a 12-character value in a column named `uuid` would mislead the next reader. The rename went further than the first estimate. A generic action used by every provisioned resource, and several tests, had used an account as a convenient stand-in for any model with a UUID. The session checked every real caller and none passed an account. It moved those tests to a real production example.

`CreateAccount` is platform-admin only. It binds an existing user by email or creates one with a random unusable password, marks the email verified at once, and sends a real Fortify password reset link, so no new mail infrastructure was needed. Before shipping, the session caught that `email_verified_at` is not fillable on `User`, so passing it to `create()` silently dropped it. An explicit `forceFill()` fixed it.

baa518a , 2026-09-15 15:12.

Name	Contact	Package	Members	Status	Actions
Contoso Web	carl@contoso.example.test	Business	1	Active	Manage
Fabrikam Studio	fiona@fabrikam.example.test	Starter	1	Active	Manage
Northwind Hosting	nora.north@example.test	Business	1	Active	Manage
ergwrg	ewrgwerg@ss.ss	Starter	1	Active	Manage

4 total

The admin view of every tenant account, the page that phase 47's Create account button hangs on.

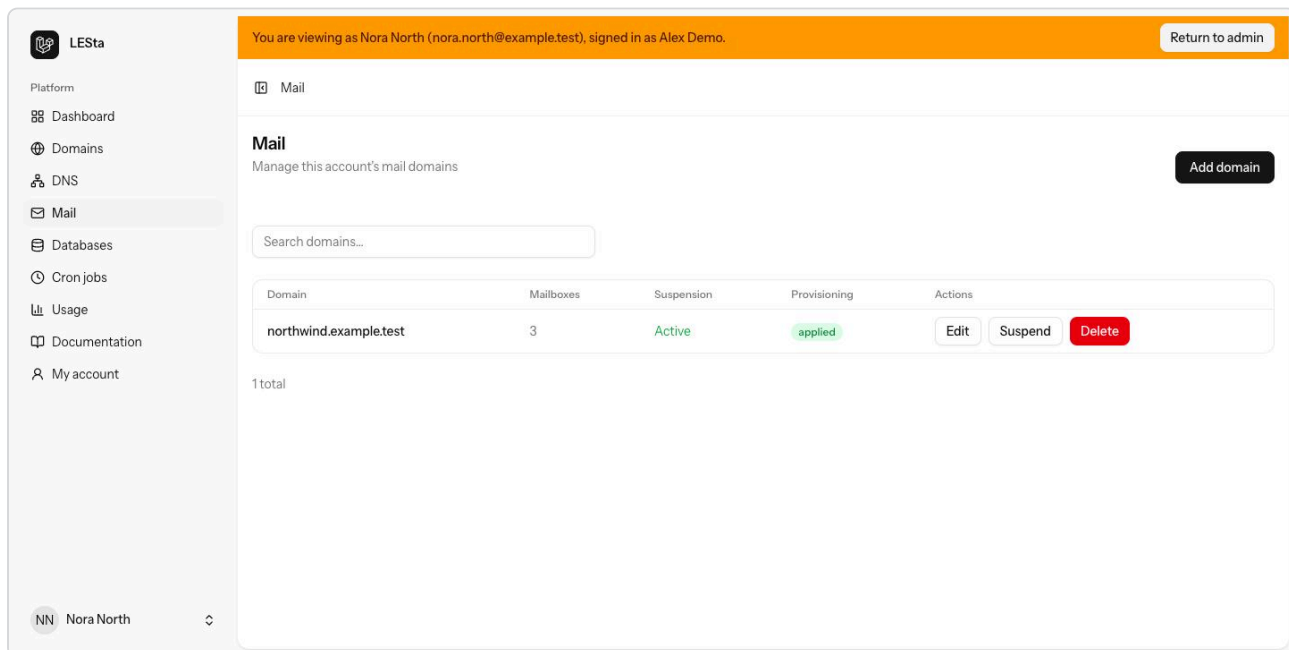
Phase 48: a mail interface for a mail system that had none

The next request was in plain words: "let's build the graphical interface for the user". The session found that most of it existed already. Domains, DNS, databases, cron, usage and account settings were all real self-service pages. Two gaps were real. Mail had no interface at all, and the phase 44 log had already said so, even though phases 29 to 44 had built the mail capability all the way to DKIM rotation and virus scanning. And the dashboard was still the untouched Laravel starter-kit placeholder.

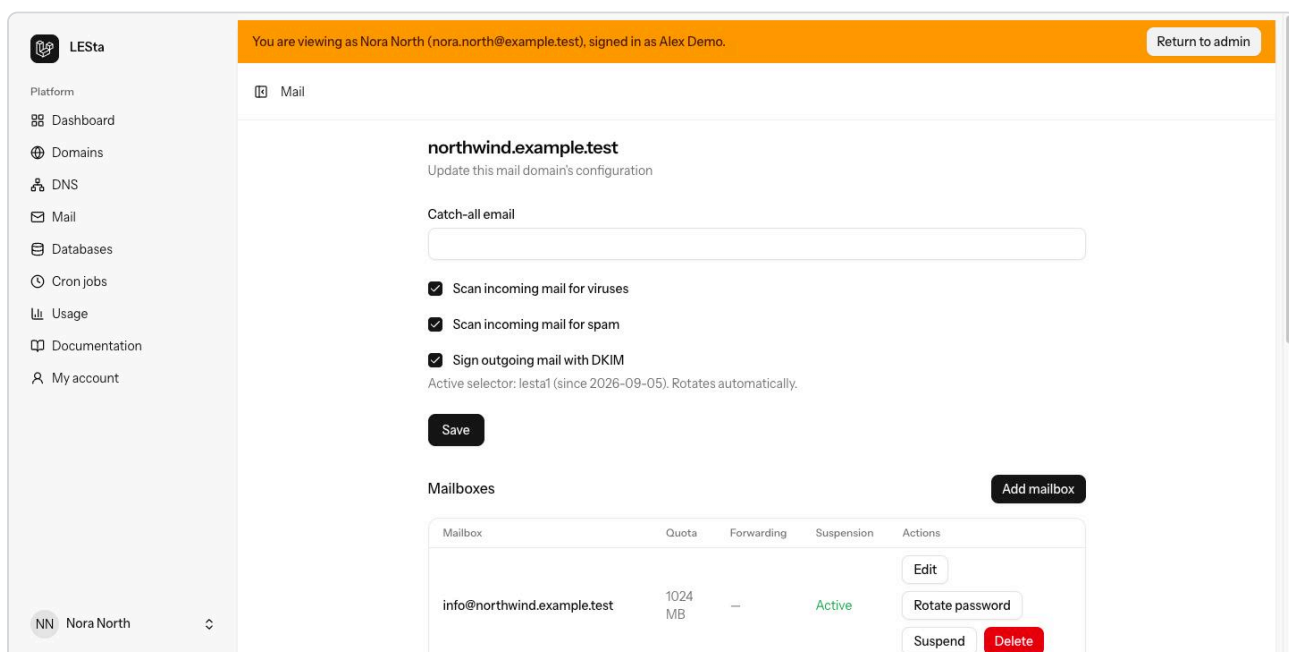
I chose both, mail first. And I confirmed that verifying on the laptop alone was acceptable for now: sqlite, HTTP-level assertions, and my own eyes on the real pages at a local address, which the session confirmed live through Herd with a plain `curl`. A server for real Exim and Dovecot testing stayed a future item.

The mail actions and policies turned out to be structural mirrors of DNS's own, so no backend work was needed. It was a controller and frontend layer copied almost line for line. There was one deliberate divergence. Creating or rotating a mailbox password returns a one-time plaintext password, and the field is encrypted and never readable again. The interface reuses the pattern for node enrollment

tokens: flash the password on the server, catch the flash event in the browser into local dialog state. It is not a page prop, so it cannot come back on a later GET or on back-navigation. A test proves that the password never appears in the domain edit page's props. 98fe324 , 16:01.



The tenant mail list of phase 48.



Mail domain settings. The DKIM line shows the active selector and says it rotates automatically, the result of phase 44.

Phase 49: removing the button, and the route

I asked to remove the register button, since accounts are not created by users, only by admins or, later, by an API.

The session did not stop at the link. It disabled `Features::registration()` in `config/fortify.php`. Fortify registers `/register` only when the feature is on, so the route itself is gone. That was confirmed with `php artisan route:list --name=register` returning nothing. The view wiring and the unreachable page were removed as dead code.

The front page was the starter kit's own demo splash, a Laravel logo, links to Laracasts and a button for deploying to Laravel's cloud, unchanged since the first phase. It became a real page in the app's own visual language: the name, one line and a login button. And the session found that `APP_NAME` was still the literal default "Laravel" in `.env` and `.env.example`, which showed in the browser tab and in the sidebar everywhere. It became "LESta".

I also asked for an admin user to log in with. The session created it directly in the local database with the `provider_admin` role and the full permission catalog, and generated a real password reset link, so that no one ever set or saw a plaintext password. With mail set to the log driver locally, it read the link out of the Laravel log. 47ec34c , 16:24.

Phase 50: the documentation that found two bugs

I asked for two markdown guides, an Admin Guide and a User Guide, with a chapter per function, and with anything missing or unclear marked in the text and not fixed silently. The session wrote them from the code: a full route list and every controller, policy and page it described.

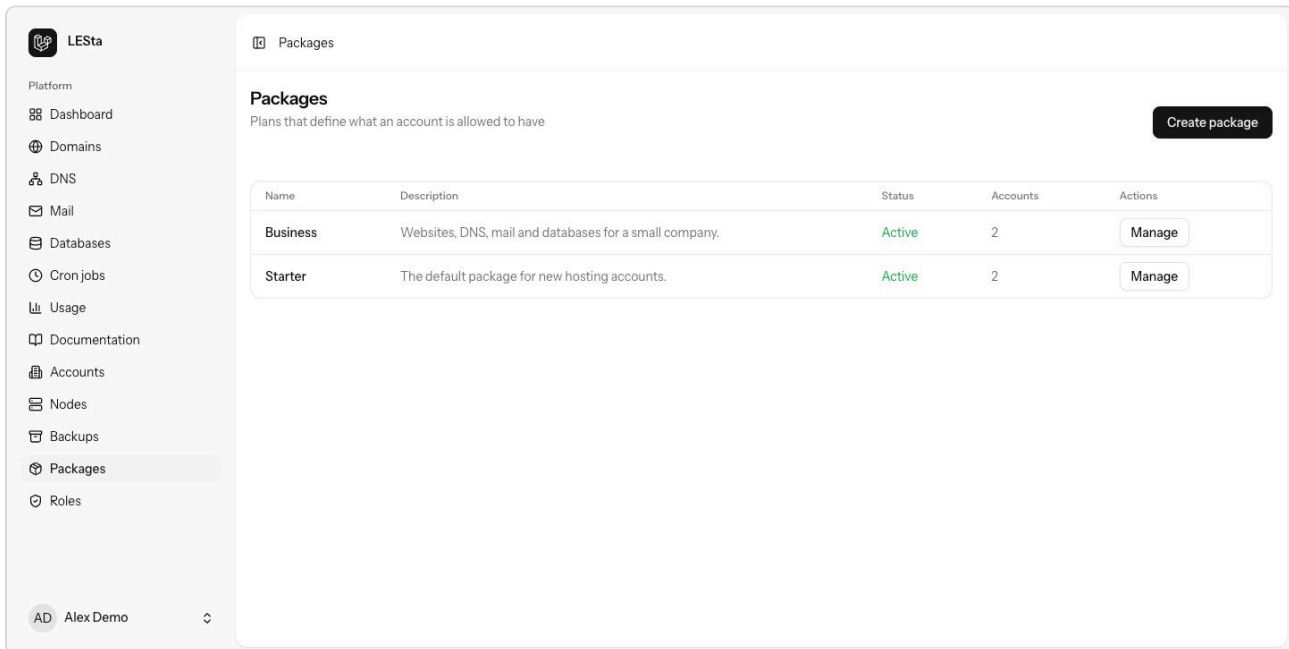
Writing the documentation is what found the two most important gaps of the day.

Three capabilities could never be switched on. The node capability dropdown, and the enum that validates it, lacked three strings that already existed: `mail.smtp-imap.v1`, `backup.encrypted-artifacts.v1` and `metrics.usage.v1`. Mail, backups and usage could not be declared on any node. The whole Mail interface of phase 48, the largest interface of the session, and the backups interface of phase 34, had no way to be enabled. The dropdown had been written in phase 19 and never updated as capabilities were added. The session checked the Go dispatch switch and each PHP gate, and did not assume that all five candidate strings in the draft guide were equally missing. Exactly three were. The other two were correctly excluded. `system.account-identity.v1` is sent automatically the first time an account gets a cron job, so there is nothing for an admin to declare. `database.control-plane.v1` has no Go dispatch case at all and is only a lookup key for the application's own database.

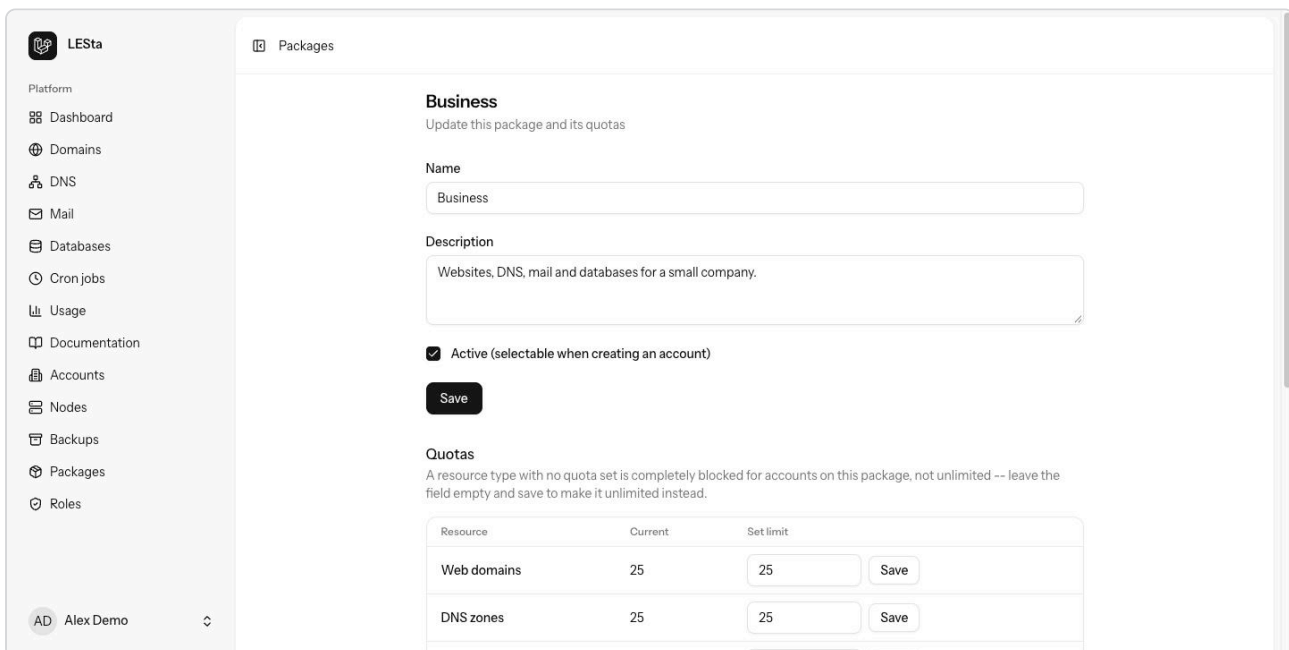
Packages had no interface. No controller, route or page existed for packages and quotas, although a full policy and an `UpdatePackageQuota` action existed, with no callers. It is the same pattern as `AccountPolicy::create()` before phase 47. I chose to fix both, in that order, before anything else.

The package work found a third bug through the practice that the project had followed all along, a full local run. `migrate:fresh --seed` failed as soon as `Package` needed a UUID. `DatabaseSeeder` uses `WithoutModelEvents`, which silently switches off the `creating` event that `HasUuid` relies on. The package seeder inserted a row with no UUID at all. The fix was to set the UUID explicitly and to make sure a re-seed never overwrites a real one, since every link to it depends on it. The log flags it in both guides as a pattern: any other model seeded from inside `DatabaseSeeder` that later gains a UUID will hit the same bug.

The quota interface offers only the seven resource types that a real create action anywhere in the codebase enforces, checked by grepping every `Create*` action. It does not offer `memberships`, which `UpdatePackageQuota` special-cases but which no real path ever checks. Offering it would imply an enforcement that does not exist. 288b677 , 18:57.



Packages, which had no interface until phase 50.



Quotas per resource type. A type with no quota is blocked, not unlimited.

Three days, five phases

Between 09:38 on Monday 2026-09-14 and 18:57 on Tuesday 2026-09-15, five phases went in. All of them started with a question from me and not from a list. Three of the five phases fixed something that already existed: a premise that was wrong, a permission that was too wide, a dropdown that was out of date. The documents I asked for are the reason two of those were found. In both cases the finding was made by writing down what the application does, and checking it against what the application does.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `5c77cdf` , `288b677` , `8859c84` , `baa518a` , `98fe324` , `47ec34c` .

Uninstall

For twenty-two days the project had installed things. It had never removed anything. The installer contract said so, in a sentence that the session quotes in the log: "Uninstall and destructive migration are separate operator workflows and are never implied by `--apply`." On 2026-09-17 I asked for it anyway, and I did it by rejecting the session's plan outright.

The guide that asked a question

The chapter starts with documentation. I had asked whether the installation documentation was clear enough to stand alone, and whether a third markdown guide, next to the Admin Guide and the User Guide, was warranted. The session checked and found the answer was no to the first.

`.install/README.md` and `.install/INSTALLER-CONTRACT.md` both still described themselves as written before implementation, "no executable installer yet", while eight real, CI-tested installers existed. It wrote an Installation Guide from each service's own manifest, installer and the contract, and marked the gaps: the two stale top-level documents, stale README text for mail and backups, no documented way to get `.install/` onto a fresh server, and four services that had manifests and no standalone `install.sh`. It fixed the two stale documents in the same phase.

A combined installer, and the rejection

Separately, I asked for a way to select several services and install them from one script. The research said this needed no new architecture. Seven services had a real, independently runnable installer: nginx, Apache, BIND, MariaDB, cron, mail and backups. Firewall and node health bootstrap inside every one of them. ACME and statistics had no standalone script. Every manifest already declared `depends_on` and `provides`, and `validate-contract.mjs` already built and cycle-checked that graph in CI. Nothing needed inventing, only walking at install time too.

The session presented `install-selected.sh` as a plan: POSIX `sh`, a `--services` list, install order resolved from the manifests, a failure before any change if the selection is incomplete, and no automatic rollback across services. I rejected it.

The rejection carried the instruction. I wanted a `--prune` parameter: uninstall whatever is on the node and not in the new selection, unless it is a security package. The log calls it "a fundamentally different kind of change". No uninstall capability existed anywhere in the codebase. The contract had deferred it explicitly. And the risk is real data loss: a live tenant's mail or database can be destroyed by a flag.

The session asked two questions before designing anything, on the two unknowns that mattered most.

How far should removal reach? The recommended option was LESt's own services only. The alternative was a broader trim of the operating system. I chose the broader trim.

What happens when a service being removed still holds real tenant data? The recommended option was to refuse outright. The alternative was to allow it, with one extra explicit confirmation. I chose to allow it with confirmation.

The log states what those two answers were: "Both of the user's choices are the more aggressive option in each pair, contrary to the recommended default." Then it states what the session did about it. It implemented both literally. It added one safety mechanism that overrides neither. Data that would be destroyed is quarantined, moved and timestamped, and not deleted. That is how the project already treats old state elsewhere, in generation-based rollback and in five-generation backup retention. The removal still proceeds on one confirmation, as I had asked. The only addition was what happens to the bytes.

The design

A list, `.install/base/protected-packages.txt`, of what `--prune` will not touch under any circumstance: SSH, `sudo`, the package manager and its trust chain, `systemd`, core networking, the firewall and fail2ban baseline, unattended upgrades, and the running kernel and bootloader by prefix. A library, `.install/lib/uninstall.sh`, with a lookup table per service of packages, units, owned directories and whether the service is stateful.

The plan had assumed the packages could be read from the manifests. Checking showed that all seven manifests declare `packages: []`. So a small hardcoded table became the single source of truth, as reviewable as the protected list.

Three services are stateful: MariaDB with real tenant databases, mail with real received mail, and backups with real backup artifacts. It is the same distinction that phase 45 made for restore and resync. Dropping one requires a repeatable `--confirm-data-loss <service>` flag, or the whole run fails before any change.

The bug that would have deleted the control plane

While designing the CI scenario for a confirmed prune, the session found a real bug before it existed in any commit. `mariadb-server` and `mariadb-client` are shared between the tenant instance, which can be dropped, and the application's own control-plane instance, which never can and is not even a node capability. The per-service removal path already handled it. It never purges those packages for a `mariadb` removal, and only stops `mariadb@tenant.service` and removes the tenant's directories. But the separate pass for the broader trim had no idea. The moment `mariadb` was absent from `--services`, it would have purged both packages, and taken the control-plane database down with them.

The fix went into the one place both paths consult: both packages were added to `protected-packages.txt`, with a comment that explains why.

A second correction was made mid-implementation. `install-selected.sh` originally printed a synthetic "would run" line per service for `--dry-run`, and did not run each installer's real dry run. An operator who always dry-runs first would then never have performed the real preflight the contract promises: the include-line check, the port conflicts, the capacity check. Both modes now invoke each real installer and differ only in whether `--apply` and `--yes` are passed.

The contract validator gained a check that the script's hardcoded list of seven services matches the real `services/*/install.sh` files on disk. It is the same kind of drift that phase 50 had found in the capability dropdown. Four disposable-Ubuntu scenarios, 28 to 31, went into CI: a successful combined run of three services, a fail-closed missing dependency, a fail-closed unconfirmed prune, and a real confirmed prune. The last removes BIND and the MariaDB tenant instance, and proves that the control-plane instance and the protected baseline survive and that a canary package installed for the purpose is really gone. It runs for real on a GitHub runner that is single-use and disposable.

The combined installer and the prune are commit `66d77d3`, stamped 04:11 on 2026-09-17. Three more commits, `ec07663`, `b184a08` and `3394f3d`, stamped 04:11 to 04:12 the same morning, carry the first three items of the batch below.

A batch of backlog

Phase 53 was a set of items closed in the same session, and one line in it says something about how the backlog had built up.

- **Honest capability status.** A capability showed as "Active" the moment an admin declared it, with no proof that the service had ever been installed. It now has `not_installed`, `running` and `stopped`, driven only by the agent's own heartbeat. An admin can never set "running" by hand. Install stays a manual step over SSH, by the boundary in the ADR against remote installation. If the agent is unreachable, every capability collapses to "unknown" instead of trusting old data. It also found that mail and backups had real state directories and were missing from the Go agent's own map, so neither could ever be confirmed by a heartbeat. And it found the same trap as phases 42 and 44: `AddNodeCapability` returned a model that did not reflect the database's own default until refreshed.
- **Members, buttons and search.** Invite and remove a member, with a guard against leaving an account without an owner. A plain member saw Update, Suspend and Delete on their own account page although only an owner's click would work. The policy was right and the bug was in the interface, which now receives the real per-ability booleans. Reseller assignment took a raw ID typed by hand, and got a search. While writing that endpoint the session found that the eligibility filter used the wrong side of the assignment rule, which would have excluded a valid reseller that already managed another account.
- **Custom admin roles.** Full CRUD for platform roles with exactly the permissions granted to them. A browser test showed that a role holding only `packages.view_any` gates access correctly end to end. It also found a bug in the test itself: Radix's checkbox is a button and not a native input, so Playwright's `check()` cannot find it. One limit is disclosed and not fixed: the navigation links still

show for the literal `provider_admin` role and not by permission, so a custom-role holder can reach every route directly and every ability is enforced, but will not see the menu item.

- **Graceful empty states.** Six tenant pages threw a raw 404 for any user with no account membership. The session noted in the log that the duplicated `resolveAccount()` function had to be edited by hand in seven places for this, and that a later phase removes the duplication. That is chapter 21.
- **Real impersonation.** `StartImpersonation` and `StopImpersonation` had existed since chapter 7, built and audited, with no caller. They now have a controller, routes, a persistent banner on every page while active, and an "Impersonate" action with a required reason next to each member, gated by a policy. It is a real session swap, and it is distinct from the read-only support view.
- **Documentation inside the app.** The two starter-kit sidebar links, "Repository" and "Documentation", were replaced with a `/docs` section built from the project's own guides, with a chapter index and a switcher. The Admin and Installation guides are shown only to provider admins, in the listing and on a direct URL.
- **An installer that fetches itself.** A repository-root `install.sh` uses a git partial clone with sparse checkout to fetch only `.install/` and the one prebuilt agent binary, and never the application source. It is an explicit fetch-then-run and never `curl | sh`, per the contract's own supply-chain rule. CI shellchecks it and runs a real sparse checkout against the commit under test. A README, the first the repository had, documents it.

The last four are `c9ffba7`, `8589629` and `217d197` at 20:21 and 20:22, and `e44fcac` at 21:47, all on 2026-09-17.

The screenshot shows the LESta management interface. On the left is a sidebar with navigation links: Platform, Dashboard, Domains, DNS, Mail, Databases, Cron jobs, Usage, Documentation, Accounts, Nodes, Backups, Packages, and Roles. The main content area displays a table of capabilities with columns for Capability, Status, and Actions. The capabilities listed are: backup.encrypted-artifacts.v1 (Running), database.tenant.v1 (Running), dns.bind9.v1 (Running), mail.smtp-imap.v1 (Running), metrics.usage.v1 (Not tracked), scheduler.account-cron.v1 (Running), web.apache.v1 (Running), and web.nginx.v1 (Running). Each row has a 'Suspend' button and a 'Mark as stopped' dropdown menu. At the bottom of the table is an 'Add capability' button. Below the table is a section for 'Recent provisioning operations' showing a single operation: 'web.nginx.v1 create' with a status of 'applied'.

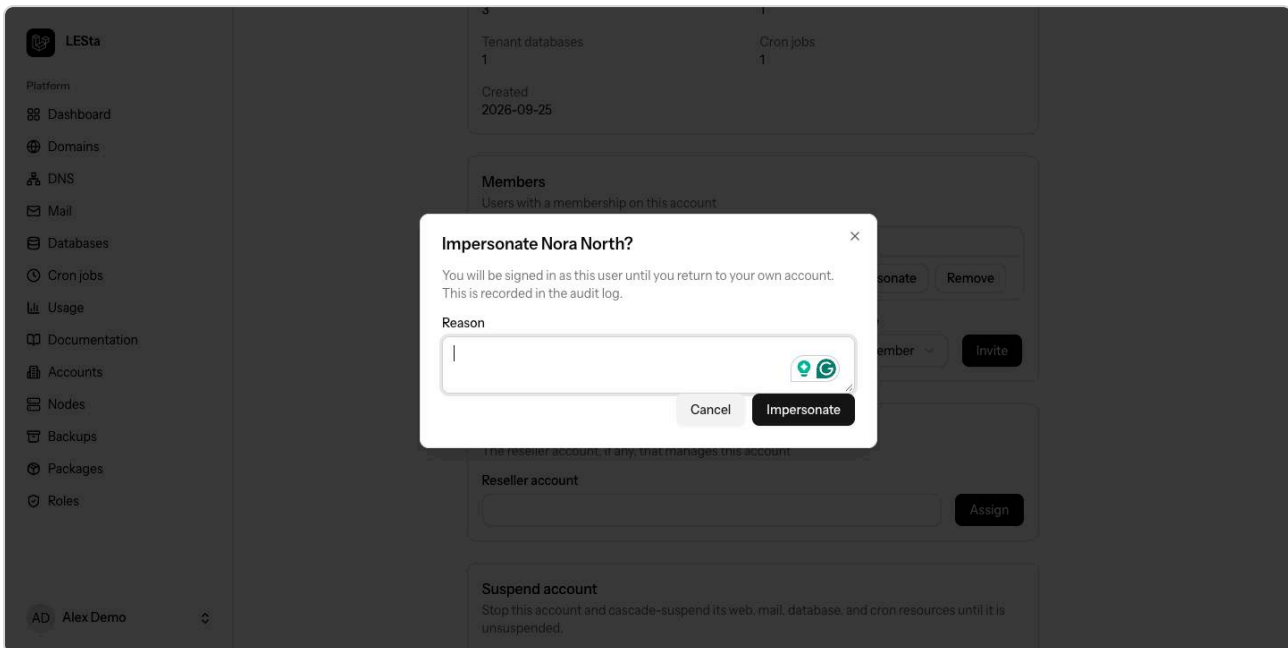
Capability	Status	Actions
backup.encrypted-artifacts.v1	Running	Suspend, Mark as stopped, Set
database.tenant.v1	Running	Suspend, Mark as stopped, Set
dns.bind9.v1	Running	Suspend, Mark as stopped, Set
mail.smtp-imap.v1	Running	Suspend, Mark as stopped, Set
metrics.usage.v1	Not tracked	Suspend
scheduler.account-cron.v1	Running	Suspend, Mark as stopped, Set
web.apache.v1	Running	Suspend, Mark as stopped, Set
web.nginx.v1	Running	Suspend, Mark as stopped, Set

Capability: `tls.acme.v1` Add capability

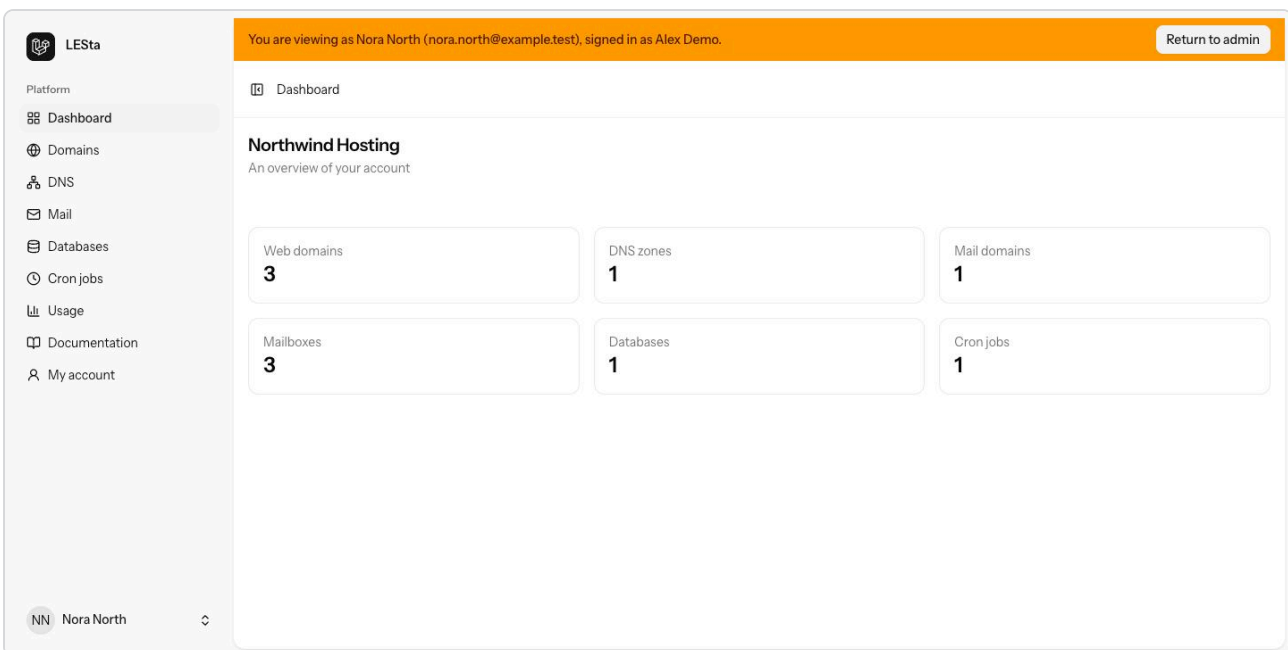
Recent provisioning operations
The most recent operations dispatched to this node

web.nginx.v1 create	applied
---------------------	---------

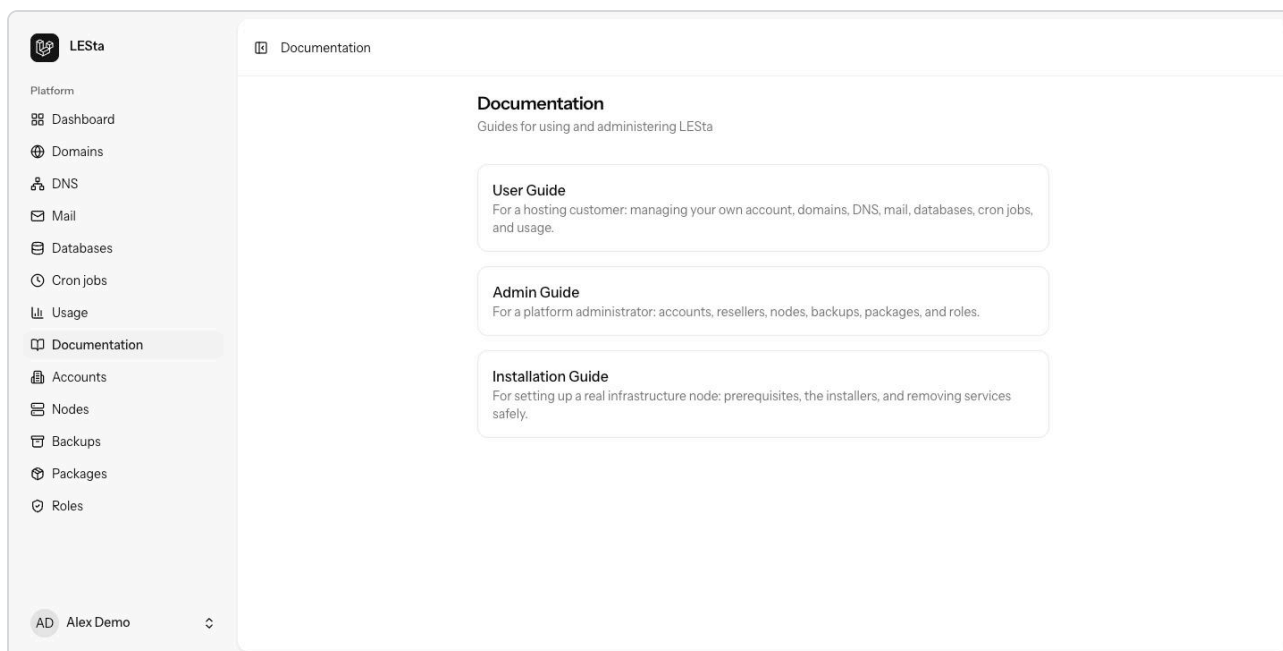
Capability status on a node. Running and stopped come from the agent's heartbeat. An admin cannot set running by hand.



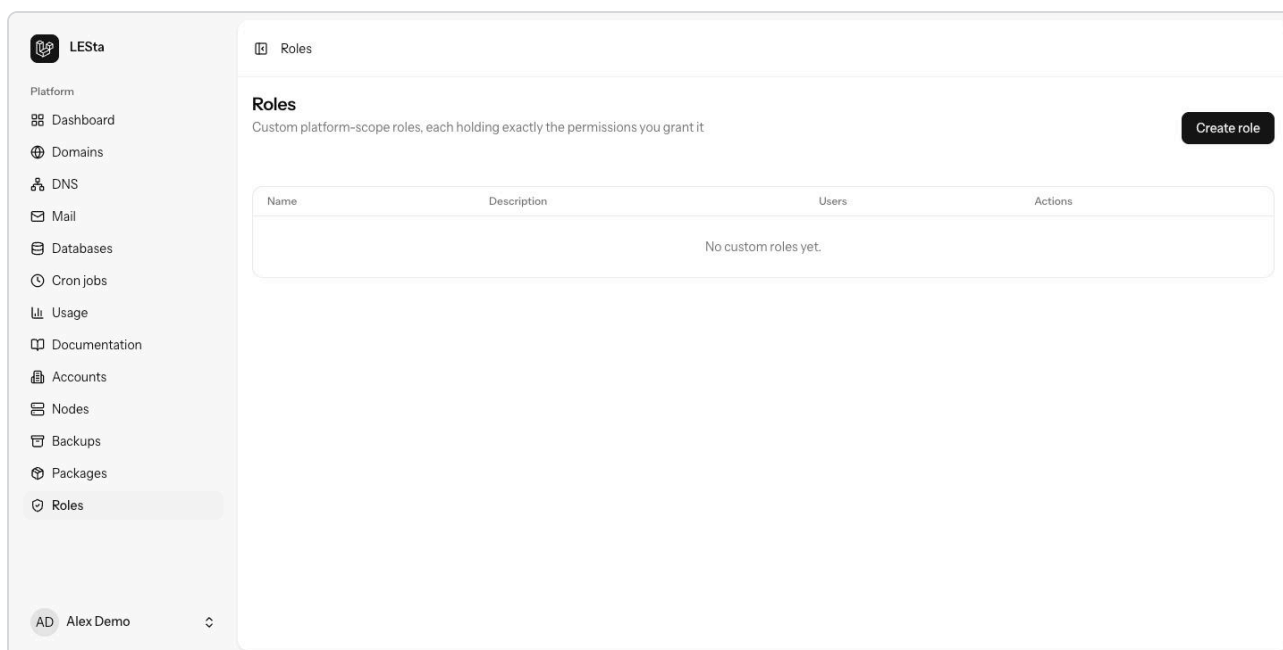
Impersonation asks for a reason and is audited.



While impersonating, a banner is on every page.



The in-app guides as an admin sees them. A customer sees only the User Guide.



Custom platform roles, empty on a fresh install.

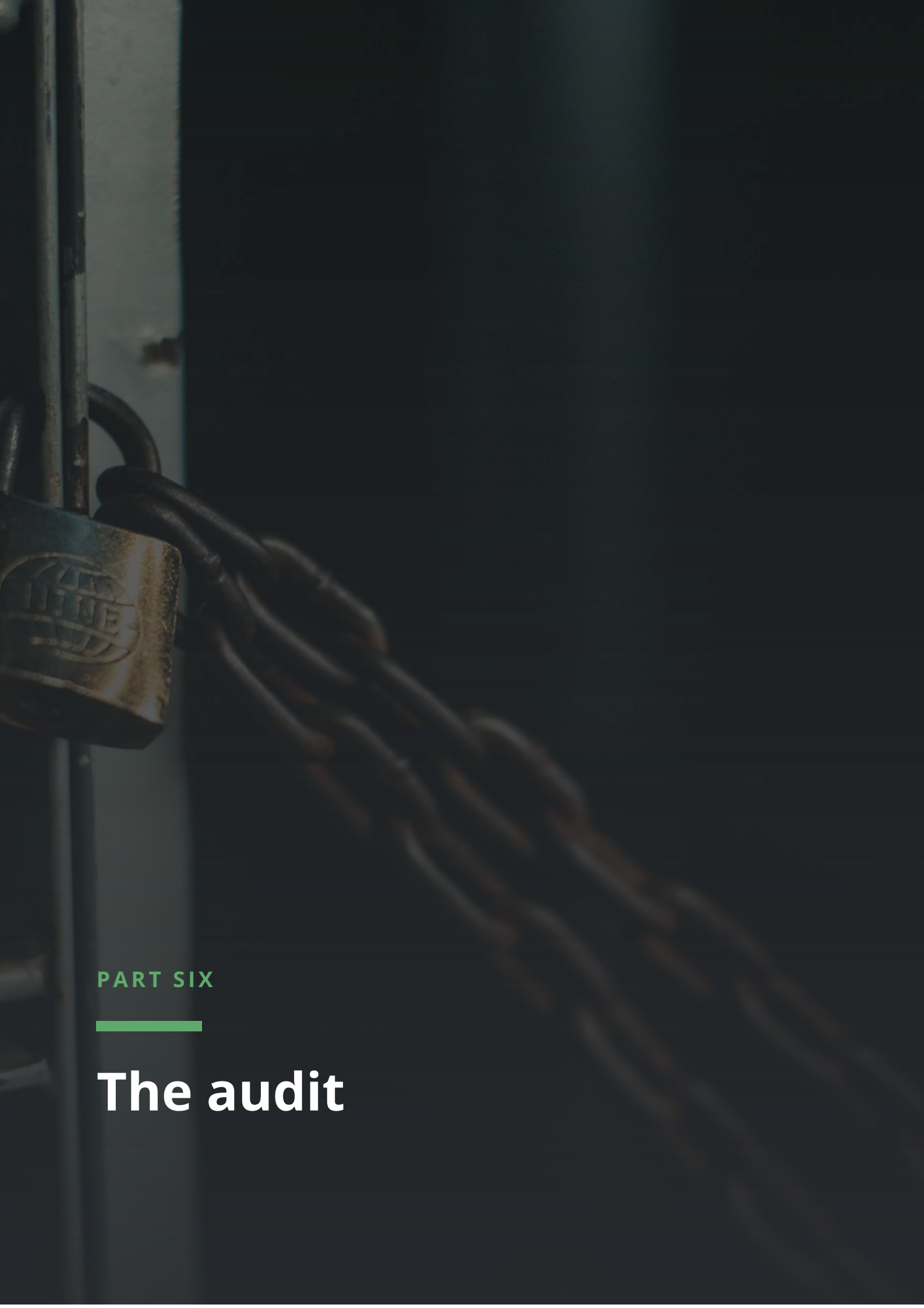
What "aggressive" did

The log calls the prune command a much larger and riskier addition than the combined installer, and I asked for it against the recommendation. The design that came back kept both of my choices and added quarantine. The prune scenario ran for real in CI on the same day, and it found the bug that would have taken the control plane's database with it, before that bug reached a commit.

I do not read that as evidence that the recommendation was wrong. It is evidence that a risky choice got an unusually close look, precisely because it was risky. That is how the session treated it.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `ec07663` , `e44fcac` , `66d77d3` , `b184a08` , `3394f3d` , `c9ffba7` , `8589629` , `217d197` .



PART SIX

The audit

The machine reviews itself

The last phases of the log are about security. They were run by the same machine that wrote the code, and that is the point of this chapter. Phase 52 is an audit in the style of a senior security engineer, of the codebase that the session had built over the previous three weeks. Phase 54 is its follow-up, and it corrects parts of phase 52.

The audit

The scope was the control plane: authentication and session security, the protocol between the agent and the control plane, the handling of node secrets, and the risk in the API, the database, the frontend and the dependencies. The log lists what was fixed first, then what was identified and not fixed, then what was confirmed clean. I will keep that order.

Fixed: two highs and a medium

HIGH: secrets in plaintext job payloads. The JSON column that carries a provisioning operation's payload down to a node held real secrets in the clear: a backup's AES-256-GCM key, an ACME certificate's private key, a freshly generated mailbox password. Each secret's own storage column, `Backup::encryption_key`, `AcmeAccount::account_key` and `MailAccount::password`, was already encrypted by Laravel. The payload copy was not.

The fix was a migration. It widens the column from `json` to `text` first, because a native JSON column's check constraint rejects an encrypted string, which is not valid JSON. The session had confirmed that this is why the two other encrypted columns are already `text`. Then it encrypts every existing row in chunks of 200, and has a symmetric `down()` that decrypts before narrowing again. The model's cast changed to `encrypted:array`.

HIGH: a suspended node still authenticated. The middleware that checks a node's credential looked only at the credential hash and the enrollment status. It never looked at `suspended_at`. When a provider admin suspended a node in phase 19, the node stopped appearing active in the interface, but it could still send heartbeats and submit operation results. A `whereNull('suspended_at')` on the lookup fixed it, and a new test proves that an otherwise valid credential from a suspended node gets a 401.

MEDIUM: no proxy configuration, and a cookie that was never forced to Secure. The application is documented as always deployed behind nginx that terminates TLS on the same host or the same private network. But `bootstrap/app.php` trusted no proxy, so `$request->secure()` evaluated the internal hop from nginx to PHP-FPM, plain HTTP, and not the real client connection. The session cookie's `Secure` flag could therefore fail to be set on a genuinely HTTPS install. Three changes. `trustProxies()` trusts loopback and the RFC 1918 private ranges only, never a public address.

`URL::forceScheme('https')` in production, together with a hard `RuntimeException` if `APP_DEBUG` is ever true in `APP_ENV=production`, which the session found as a direct extension of the same review. And the session config sets `secure` to true in production.

That last one hit a Laravel constraint. Calling `app()->isProduction()` inside a config file throws `Target class [env] does not exist`, because config files load before the container's `env` binding is guaranteed to exist. Reading `env('APP_ENV')` directly avoids it, as every other config file does.

Identified and not fixed

The log lists nine, and states for each that its remediation status was not confirmed at the time of writing.

- A **quota race** that was systemic. Every creation action does a `count() >= limit` check inside a database transaction with no row locking. `grep -rn lockForUpdate app/Actions/` returned nothing. Two concurrent requests could both pass the same check and both succeed.
- An **untested bypass invariant**. A global authorization bypass for provider admins has an exclusion list of models that must be kept in step by hand, with no test asserting that it is complete.
- **N+1 queries** in four controllers' list pages.
- **No rate limit** on the password reset routes.
- An **audit log that is fully mutable**. No route deletes from it today, but nothing stops a future one from doing so silently.
- `.env.example` defaulting to `APP_DEBUG=true`.
- `resolveAccount()` **duplicated** across seven controllers.
- Per-request authorization checks that issue a fresh query each time.
- And a note on database-backed queue and cache defaults as a scaling ceiling.

Confirmed clean

The section that is easiest to skip, and the one the log says a future session should not have to redo. There is no raw or concatenated SQL anywhere in the app. Mass assignment is airtight, with explicit `# [Fillable]` attributes and no validated request fed straight into `create()`. Authorization on tenant resources is enforced twice, in the controller and in the action. Two-factor, passkeys and password reset are correctly implemented, with encrypted TOTP secrets, real WebAuthn sign-count replay protection, hashed reset tokens and a strong production password policy that includes a `HavelBeenPwned` check. The agent protocol has no `RunCommand` or `Install` operation, enforced by ADR 0002 and by the JSON schema's `additionalProperties: false`. The shell installers use real quoting discipline with no injection found. On-node secret files are consistently 0600 or 0640. The frontend has no real XSS surface. `npm audit` and `composer audit` reported zero vulnerabilities.

Two of the items in that list can be checked against earlier chapters. The absence of `RunCommand` is the first item of the plan's prohibited list from chapter 3, and the audit finds it held. The file modes are the DKIM lesson from chapter 17.

The follow-up, and what it had to correct

Phase 54, on 2026-09-18, closed every item that phase 52 had left open. It also went back and corrected two of phase 52's own fixes, and the log says it records the corrections here and does not rewrite the earlier entry.

The encryption migration had a serialization bug. Verifying it against real MariaDB and not only SQLite showed that the backfill used the global `encrypt()` and `decrypt()` helpers, which serialize by default. Eloquent's `encrypted:array` cast calls the encrypter without serialization. Data written by the migration would have been undecryptable by the model. It now uses `Crypt::encryptString()` and `decryptString()`, and a full up, down and up round trip was proven on MariaDB with real data.

A node credential needed rotation, and not only a suspension check. The long-lived bearer credential had no revocation at all. A leaked one stayed valid indefinitely, and the only recovery was to delete the whole node record. `Node::issueEnrollmentToken()` now clears the credential hash and moves to the `Revoked` status, an enum case that had existed and never been used. On an already-enrolled node the "issue enrollment token" action is now the revocation path, records a distinct audit action, and relabels itself "Rotate credential" with a warning that the node will stop authenticating immediately. It was checked with a real browser test and a real HTTP request showing that the old credential is rejected afterward. The installer for the agent daemon also gained a format check on `--node-uuid`, as defence in depth.

The password-reset limiter was added by a route event, not middleware. The routes for reset had no rate limit, unlike login, two-factor and passkeys. The session confirmed that this application's routing bootstrap finishes loading route files, Fortify's included, only after `Application::booted()` callbacks have already run. So attaching middleware to the routes would have silently attached nothing. A `RouteMatched` listener does it.

Then the items that phase 52 had left.

- **The quota race:** a new `EnforcesPackageQuota` trait takes `lockForUpdate()` on the real concurrency boundary, the account for account-wide resources or the specific parent row, a DNS zone or a mail domain, for the two counted per parent, before evaluating the count. A second request waits for the first to commit and then reads an accurate count. It is applied to all seven creation actions that had the bare check.
- **The bypass invariant:** a test reflects over every model implementing `ProviderAdminManaged` and fails if one whose policy calls `hasPermission()` is missing from the exclusion list. The session verified that the test works by deliberately breaking the list first and confirming the failure and its message.
- **N+1 and duplication:** the four controllers now eager-load, and the identical `resolveAccount()` from seven controllers is one `ResolvesCurrentAccount` trait. The log remarks that phase 53 had just paid for the duplication by editing all seven by hand.
- **The mutable audit log:** each `AuditEvent` row now chains to the one before by a SHA-256 hash of its own fields and the previous row's hash, computed in the model's `creating()` hook, so none of the roughly twenty places that create an audit event had to change. A migration backfills the chain.

Changing or deleting any row breaks the chain from that point on, and `php artisan audit:verify-chain` detects it.

- **Memoized authorization:** five user methods each ran a query on every call, and the shared Inertia props alone call three of them on every page load. They are memoized per user instance and keyed by method and arguments, since a single flag per method would have been wrong, not merely stale: two of them recurse onto the same instance with a different account for the reseller fallback. The session first checked that no code path or test changes a membership and then re-checks the same method on the same loaded user within a request.

And one item that was not an audit finding at all. `install-cp.sh` is a guided setup for the control-plane application itself, distinct from the node-facing `install.sh`. It handles `.env`, dependencies, a production build and migrations. It detects and merges the database credentials from the MariaDB installer when they are on the same box, and prints copy-and-paste guidance for the web server, systemd and cron and never touches them. Testing it end to end found a bug: `DatabaseSeeder` created a "Test User" through a factory that needs `fakerphp/faker`, a dev-only package that a `composer install --no-dev` production install does not have. It is now guarded behind the local and testing environments.

The push that would not go

One process fact in this phase has nothing to do with the code, and it is not in the decision log. It comes from the session records. The session had been blocked on `git push`. The local GitHub CLI authentication was on an account that did not have write access to the repository, and the push failed with a 403. It is the same class of trouble as chapter 16's keychain. On 2026-09-20 the blocked session told a second, concurrent session about it, and that second session, with the correct account, pushed all eight commits at about 22:22 Swedish time. That is two days after the work was done.

The repository shows the eight commits stamped between 12:32 and 12:34 on 2026-09-18, in a burst: `034f714`, `907e408`, `5504717`, `af4cada`, `7a03179`, `85d0aa0`, `b46c7ab` and `2ecfb6c`. They are the work of the session, and they reached GitHub together. The burst is the last work commit in the repository. The only commit after it is the README correction from chapter 22.

What a self-review does and does not show

It found two high-severity problems in code that the same tool had written and that had passed tests and CI for weeks. Both were the kind that tests do not find, because they are about what should not be possible: a secret that should not be in a column, a credential that should not work. It then corrected two of its own fixes, one of them because a real database was used and not a substitute.

It also found things clean, and the list is specific. The chapter's claim is smaller than "this system is secure". It is that a second pass by the same tool, told to be a security engineer and told to look for what the first pass could not see, found real problems and fixed them, and that the findings are written down with their status. Chapter 23 is about what has still not been checked by anyone.

SOURCES

- Commits, each at github.com/mikho/LESta/commit/ followed by the hash: `034f714` , `2ecfb6c` , `907e408` , `5504717` , `af4cada` , `7a03179` , `85d0aa0` , `b46c7ab` .

The manual nobody had

The last entry in the decision log is a documentation phase. It has no commit in the repository, since the file it rewrites lives in the vault. The file dates put it at 2026-09-22, four days after the last commit. It is a short chapter of the story, and it is here because the documents were the one thing in the project that found bugs without running any code.

Three guides

By the middle of September the project had three guides. The User Guide is for hosting customers. The Admin Guide is for platform administrators. The Installation Guide is for whoever sets up nodes and the control plane. About 6,600 words in all. They live in the vault, and since phase 53 they are also served inside the application at `/docs`, from `resources/docs/*.md`. The Admin and Installation guides are shown only to provider admins.

I asked for them, and I asked for them in a particular way. Each guide has one chapter per function. Anything that is missing or unclear is marked in the text, and is not fixed silently. That rule, more than the writing, is what made them useful.

Three of the findings in earlier chapters came out of writing them:

- Writing the Admin Guide is how phase 50 found that mail, backups and usage could never be switched on for a node, because the capability dropdown was missing three strings. It is also how it found that packages had no interface.
- Writing the Installation Guide in phase 51 is how the two stale top-level documents were found, still describing themselves as written before the installers existed, and four services with a manifest and no installer.
- Phase 55 found the last one, below.

The guides check the code, and the code does not check the guides. Every value in them was pulled from the codebase and not copied from the prose of an earlier draft. That method found things that reading code for bugs had not, because the questions are different. A reviewer asks whether the code is correct. A guide asks what an administrator would do next, and the answer to that is sometimes "there is nothing they can do".

The question about one server

Phase 55 started with a question that I asked while reading the Installation Guide. It never said that the control plane and the hosting services can run on the same server. The ADR's own default is one node running everything: one MariaDB installation providing two instances, on ports 3306 and 3307. The

guide described only how to set up a hosting node. It did not mention `install-cp.sh` at all, the script for the control-plane application itself, and it still carried four unresolved gap markers for things that the codebase had since answered.

My prompt for the rewrite required zero remaining gaps, zero placeholders and no "ask the developer" language. The result had to be one document that an administrator who had never seen the project could follow from beginning to end.

It covers both halves of a real deployment. Part A is the control-plane application, set up with `install-cp.sh`. Parts B and C are one or more hosting nodes. It is explicit throughout about the single-node default and the split alternative.

The log lists what was pulled from the code, some of it for the first time written down anywhere in the vault:

- The full table of `.env` variables, cross-checked against `.env.example` and every config file that calls `env()`. It calls out the AWS, Redis and Memcached keys as inert leftovers from the starter kit that this application never reads.
- The production hardening that the application actually enforces: forced HTTPS, forced secure cookies, the refusal to boot with `APP_DEBUG` on in production, the reset limiter, and `DB::prohibitDestructiveCommands`. This became the guide's Security chapter, and it is chapter 21's list, read back from the code.
- The exact system package names per service, read from each installer's `install.sh` and not from the manifests, since every manifest declares an empty `packages` list. That is the same fact as chapter 20's `--prune` table.
- A tested procedure for creating the very first `provider_admin`. No installer, seeder or artisan command does it. The session traced `CreateAccount::resolveOwner()`, with its random unusable password, immediate email verification and real reset link, and wrote it up as a `tinker` recipe with a platform-scope membership and a null account. It is what phase 49 had done by hand, with no procedure to follow.
- A confirmation, from the Go agent's `dump.go`, that the backup capability already covers the control-plane database and not only the tenants', with `--all-databases` against every reachable socket. The previous draft's silence on that would have left an administrator guessing.

Scheduled commands, artisan commands, ports, systemd units, the directory layout and every troubleshooting entry were verified the same way, against `routes/console.php`, `php artisan list`, each manifest and the installers' own preflight error strings.

What it does not cover

The log says so at the bottom. High-availability and multi-instance control-plane deployment are not covered, because the ADR's default is a single instance. Docker, Kubernetes and Windows are not covered either. The project is Ubuntu and systemd end to end, and the repository has no Dockerfile or compose file anywhere. Neither exists as a real deployment path today, and the guide does not pretend otherwise.

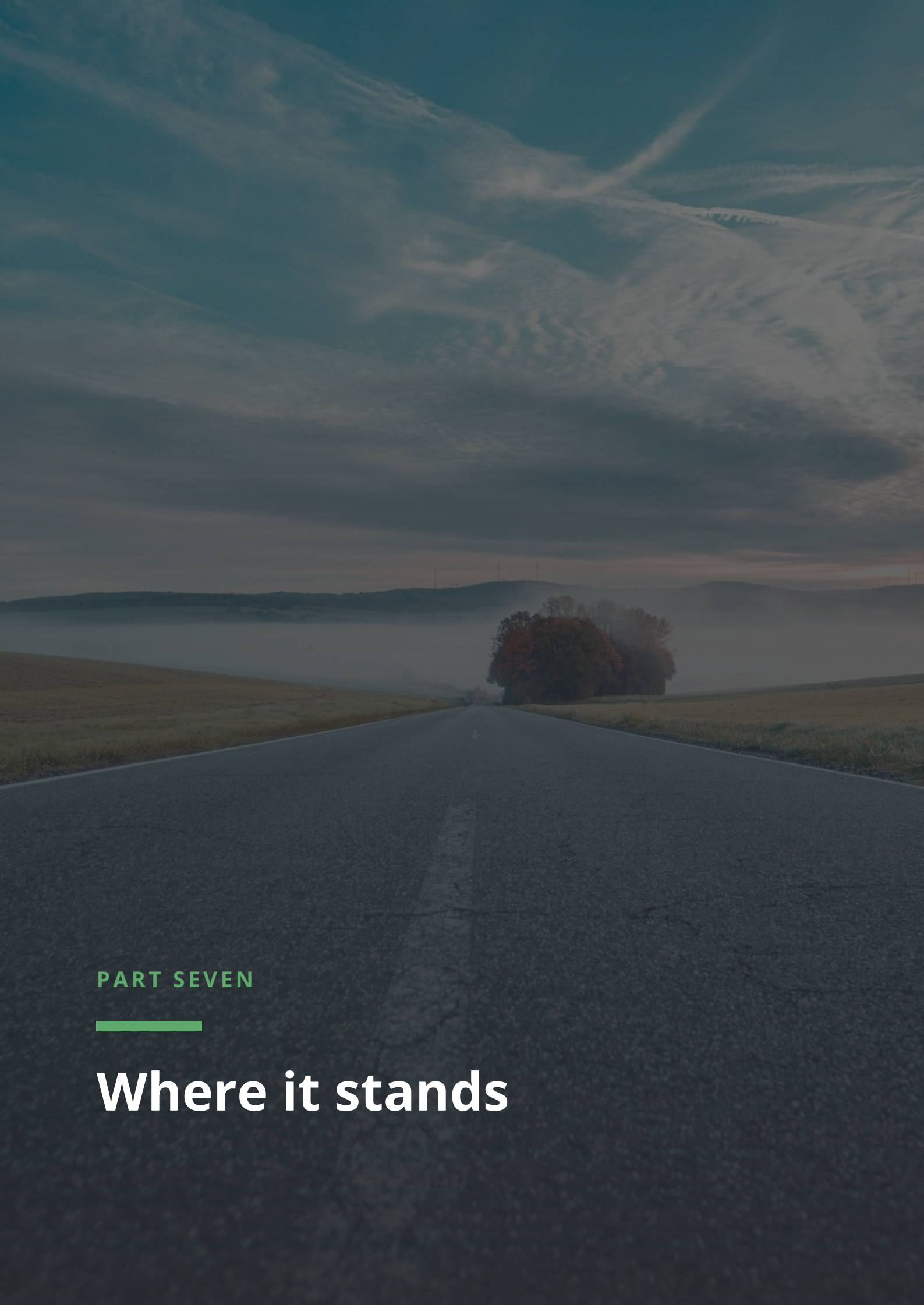
A last inconsistency

One error in the repository's own documentation is worth putting on the record, because it was found while checking this chapter and not by the process above. The repository's `README.md` says twice that LESta is "a standard Laravel 12 + Inertia v3 + React application". `composer.json` requires `laravel/framework ^13.17`, and the decision log names 13.26.1 in phase 47. The book uses 13. The README was wrong, and no session had caught it. It was corrected on 2026-09-25 in commit `e8289ab`, after this chapter was first drafted.

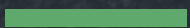
The point I take from it is the one that the guides made throughout. A document is a claim about the system, and it stays true only while someone checks it. The three guides were checked against the code in the last week of the project. The README was not.

SOURCES

- The three guides as shipped in the repository: github.com/mikho/LESta/tree/main/resources/docs.
- The README correction: github.com/mikho/LESta/commit/e8289ab.



PART SEVEN



Where it stands

Where it stands

This is the last chapter for now. The book stays open. More chapters get added as the work continues, and the first one is going to be about an install that has not happened yet.

The bet, checked

The challenge was made on 2026-08-25. The warning was that even with agentic code it takes weeks or months to turn everything over and over again, and that a day or two would not do.

The first commit is dated 2026-08-25 and the last one, at the time of writing, 2026-09-18. That is 24 days, in which 160 commits and 55 decision-log phases went into the repository. The transcripts show 24 calendar days with activity and about 58 hours of machine time. So "weeks" was right. It is not yet known whether "months" was, because months would begin with the first real install, and that has not happened.

Figure 6. Fifty-five phases in 29 days

One square per phase, on the day of its commit. Phase numbers are those of the decision log.

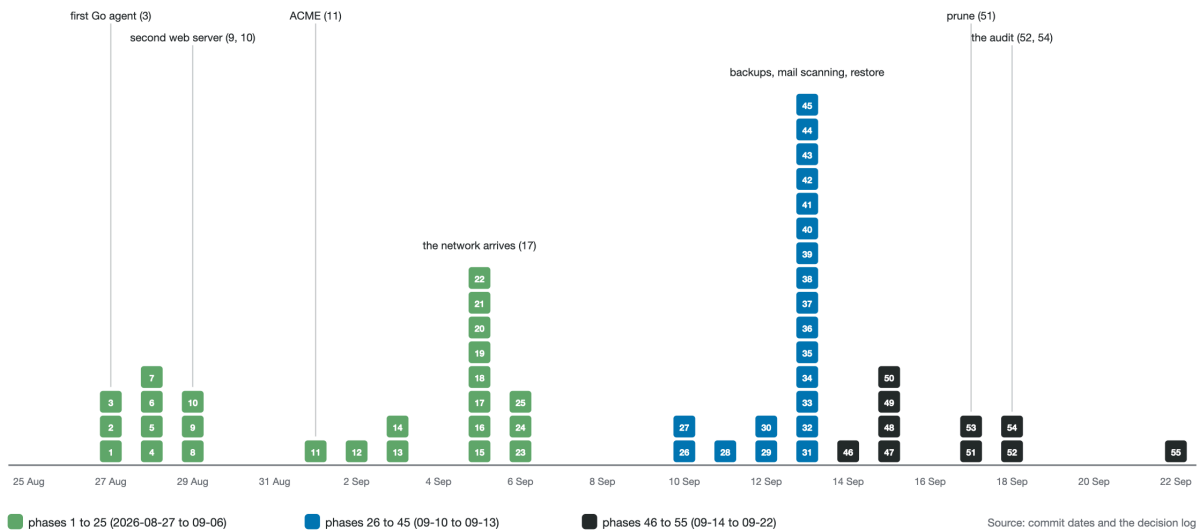


Figure 9. Two panels, in facts

Lines of code are not comparable across the two, so they are listed and not charted.

Vesta (pinned commit 5f4ee2ef)	LESt (2026-09-24)
102,000 lines in bash, PHP and JavaScript	16,900 lines of PHP, 25,300 of Go, 10,400 of shell
379 bash scripts holding all business logic	10 real capabilities behind one protocol
136 PHP endpoints that build shell commands	No RunCommand and no exec from a controller
105 sites that eval a stored record	Mutations go through policies, audit events and idempotent operations
0 flock calls across 103 mutating scripts	Per-resource serialization in the protocol
error_reporting(NULL) in 133 API files	201 test files, and real Ubuntu CI for every installer
4,666 duplicated lines in 12 list screens	Payloads encrypted, audit log hash-chained
An arbitrary file write behind login	Never installed on a real server yet

What exists

By measurement, on 2026-09-24:

- About **16,900 lines of PHP** in the application, **25,300 lines of Go** in the node agent, and **10,400 lines of shell** in the installers. The TypeScript is about 41,700 lines, and part of it is generated route helpers, so it is not all written.
- **165 PHP test files** and **36 Go test files**. The last full Pest run recorded in the log is phase 50: 658 of 664. The log names one failure, a known and unrelated timestamp artifact, and two tests skipped on purpose. It does not account for the other three.
- **Eleven capability strings** in the agent's dispatch, ten of them real capabilities: web on nginx and on Apache, DNS on BIND, mail on Exim and Dovecot, tenant databases, account-scoped cron, ACME certificates, encrypted backups, usage metrics and per-account system identities. The eleventh, the control-plane database, is a lookup key.
- **Twelve service directories** in `.install/services`, seven with a standalone installer (nginx, Apache, BIND, MariaDB, cron, mail and backups) and an eighth for the agent daemon. Firewall and node health bootstrap inside each of them. A combined installer with a prune mode, offline bundles for the five apt-based services, generation-based rollback, and two setup scripts: `install.sh` for nodes and `install-cp.sh` for the control plane.
- A control plane with accounts, memberships, roles and permissions, packages and quotas, resellers, node-scoped admins, impersonation with an audit trail, a hash-chained audit log, and interfaces for domains, DNS, mail, databases, cron, backups, usage, nodes, packages and roles. Three guides, about 6,600 words, served inside the app.
- **Continuous integration** that runs each installer on a fresh Ubuntu runner: a real apply, an idempotent re-apply, offline installs, tampered artifacts, upgrades that fail and roll back, and a confirmed prune.

What has not happened

This list is as important as the one above, and it is the one the record is clearest about.

- **A real install on a real server.** Every installer has been run on a GitHub-hosted runner, which is a disposable Ubuntu machine. None has been run on a server I could log into, with a real domain, a real network and real mail arriving.
- **Mail against the real thing.** Mail passes tests against Exim and Dovecot on runners and on the laptop. Real delivery and real IMAP against a real node were deferred by me in phase 48, where I accepted verification on the laptop alone for now, with a server for the purpose named as a future step. The log records a further gap: the laptop had Dovecot 2.4, the Ubuntu targets ship 2.3, and the two configuration formats are incompatible, so Dovecot's own config parsing was never tested locally.
- **Ubuntu 26.04 for MariaDB.** The installer fails closed on it, because MariaDB Foundation's repository does not publish for it. The log flags it for re-checking. The ADR asked for 24.04 and 26.04 both.
- **More than one node.** Every resource is assigned to a node, and the protocol is built for several. It has been tested with one.
- **An outside security review.** The audit in chapter 21 was done by the same tool that wrote the code, told to review as a security engineer. It found and fixed real problems and lists what it found clean. It is not the same as a person who did not build it trying to break it.
- **Load.** The queue and the cache are database-backed, which the audit notes as a ceiling. Nothing has been measured under traffic.
- **Backups off the node.** Local disk only, by my own choice in phase 31. A restore only targets the node the backup was taken on.

Several things were deferred on purpose and are recorded in the feature request list and the logs. Changing a node's web profile after install. Reseller-initiated account creation. Reseller nesting beyond one level. Node admins reaching tenant resources. Automatic rotation of node credentials on a schedule. `.env.example` still defaults `APP_DEBUG` to true, overridden by `install-cp.sh` and refused in production by the boot check from chapter 21, and still a risk for someone who copies the example by hand.

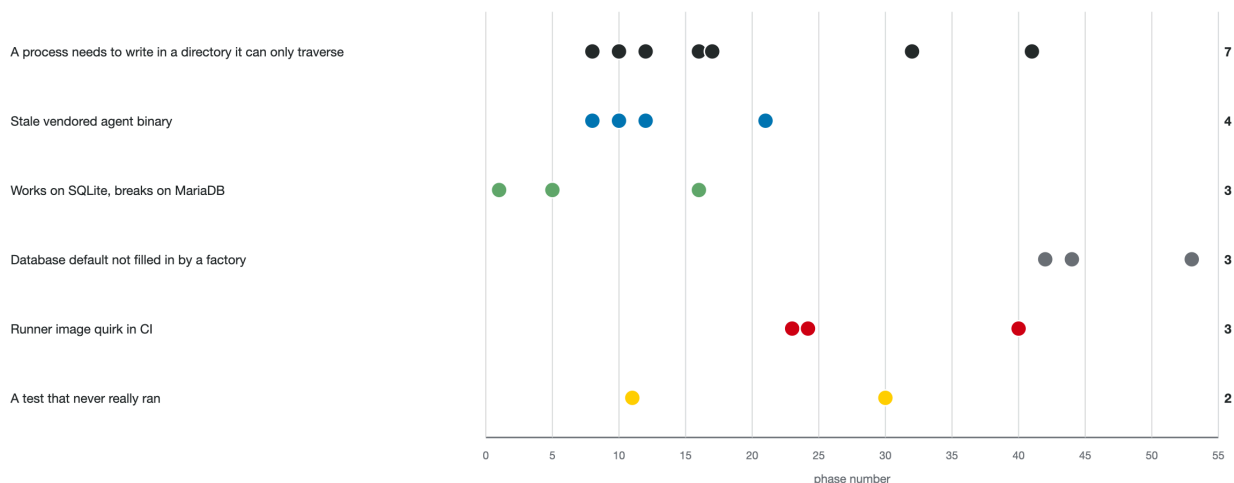
What is next

In order, and in my words at the start of the book: the first install and a test of what happens. Then code checks, optimization, and closing vulnerabilities.

The first install will produce the chapter that this one cannot write. Some of the things in the list above will turn out fine. A few will not. I expect the record to be what it has been so far, that most of what breaks will be a behaviour of the real system that no document described. Whatever comes out will be added here in the same way as everything else, in the order it happened.

Figure 7. Bugs that came back

Each dot is an occurrence, placed by phase number. Only cases named in the decision log are counted.



Counts on the right are occurrences. The ownership class has seven, and the log itself counted the first five before mail added two more.

What the record teaches

Here are seven points that the chapters support. Each is tied to the place where it happened, and none is advice that I could not point to.

1. **Say what you mean in the prompt, and read what came back.** The Apache mistake in chapter 4 was mine, and it was caught the same day by reading a line of the log and asking whether I had said that.
2. **Separate the bugs from the decisions.** In chapter 6 the session adopted engineering corrections directly and put four policy questions to me. The read-only support view exists because that question was asked.
3. **Re-run what the agent reports.** Chapters 7 and 8 have the session repeating every check itself, and finding a serious hole and a stray file that the agent's report had not mentioned.
4. **Check what the green means.** Chapters 11 and 16 are two tests that skipped for a long time under a green mark, and the second was hidden by a cache. Look at the elapsed time.
5. **Do not guess twice in a row.** Chapter 10 states it, and chapters 15 and 18 apply it. When the second guess fails, add a diagnostic and no third guess.
6. **Write the limit down where the user will see it.** Chapter 13's shared cron runner was undone in two days and cost one phase, because the limitation had been written down and not hidden.
7. **Keep the state where the next session can read it.** Chapters 6 and 12. A run that can be cut off by a rate limit, a sleeping laptop or a cloud sandbox with no local files needs its state on disk, or it starts over.

SOURCES

- The discussion thread: lowendspirit.com/discussion/11268/vibe-coding-a-vestacp-alternative.
- The repository: github.com/mikho/LESts.

The numbers

Every figure that the chapters quote, with how it was measured and how far to trust it. Measured on 2026-09-24. Nothing here is an estimate unless it says so.

From the Claude Code session transcripts

Source: the transcript files that Claude Code keeps for the repository, 106 files (3 main sessions and 103 subagent transcripts). Each unique assistant message counted once, using its last usage record.

Measure	Value	How far to trust it
Span	2026-08-26 20:44 UTC to 2026-09-22 08:46 UTC	Exact for these sessions
Calendar days with activity	24	Exact
Active machine time	about 58 hours	A floor for machine time, counting gaps under ten minutes. Not my hands-on time
Assistant requests	11,790 (4,106 by subagents)	Exact
Output tokens	7,474,981	Exact
Fresh input tokens	about 62 million	Exact
Cache reads	about 4.7 billion	Exact, but mostly the same context read again on each request. Not new work
Model recorded on each request	claude-sonnet-5	As recorded
Prompts typed by me	58	Real prompts only
Of those, 25 characters or shorter	18	Exact

Not covered: the first three prompts (another tool), any session outside the repository folder, the other projects that shared the plan's limit, and any dollar cost. See chapter 12 for the reasons.

From the repository

Measure	Value
First commit	2026-08-25

Measure	Value
Last commit	2026-09-18
Commits	160 up to 2026-09-18, plus one README correction on 2026-09-25 (161 in all)
Busiest days	2026-09-13 (38 commits), 2026-08-29 (27)
PHP in <code>app/</code>	about 16,900 lines
Go in <code>agent/</code>	about 25,300 lines
Shell in <code>.install/</code>	about 10,400 lines
TypeScript in <code>resources/js</code>	about 41,700 lines, including generated route helpers
PHP test files	165
Go test files	36
Capability strings in the agent's dispatch	11 (10 real)
Service directories in <code>.install/services</code>	12 (7 standalone installers plus the agent daemon's)
In-app guides	3, about 6,600 words

From my project notes

- Decision log: 55 phases, about 393 KB.
- Phase plans: 25 files, about 193 KB.
- The governing plan: 2,238 words, 7 phases (0 to 6), 28 numbered steps.
- Questions asked and answered in the Q&A file, first 24 entries: 18 answered as recommended, 6 differently.

About Vesta, from the report

Pinned commit `5f4ee2ef` . 102,000 lines in three languages. 379 bash scripts, 34,600 lines. 136 PHP endpoints, 42,900 lines. 105 sites that read a record back with `eval` , 191 uses of `eval` in the tree. Zero `flock` calls across 103 mutating scripts. `error_reporting(NULL)` in 133 files under `web/api/v1/` (the report said 136, a recount on 2026-08-26 gave 133).

Weekly windows (see chapter 12)

Window starting (Thu 21:00 Swedish time)	Requests	Output tokens	Fresh input tokens	Active hours
2026-08-20 (partial)	763	0.59 M	5.8 M	5.0

Window starting (Thu 21:00 Swedish time)	Requests	Output tokens	Fresh input tokens	Active hours
2026-08-27	3,122	2.11 M	23.7 M	14.9
2026-09-03	2,344	1.25 M	12.2 M	10.9
2026-09-10	4,899	3.12 M	16.5 M	22.4
2026-09-17 (partial)	703	0.41 M	4.0 M	1.8

A note on commit hashes

On 2026-09-25 the repository's history was rewritten to change the author and committer identity on every commit, and force-pushed. Every commit hash in this book is the hash after that rewrite. The vault's decision log and plans still carry the older hashes. Dates, times and subjects are unchanged, and the hashes in this book are the new ones.

Glossary

Short definitions for the terms the chapters use. Each points at the chapter where it matters.

ADR. Architecture decision record. A short document that states one decision, the alternatives and the reasoning. LESt's ADR 0002 covers installation and build order (chapters 4 and 5).

ACME. The protocol that certificate authorities such as Let's Encrypt use to hand out certificates. HTTP-01 proves control of a domain by serving a file. DNS-01 proves it with a DNS record (chapter 11). Pebble is Let's Encrypt's own test server for it.

Agent, node agent. The small Go program that runs on each hosting server. It is the only thing allowed to change that server, and it does so only through named, validated operations (chapters 3, 6 and 8).

AppArmor. A Linux mechanism that confines a program to a list of allowed files, whatever user it runs as. It is why a permission that looked correct still failed for BIND, Apache and others (chapters 9 and 10).

Capability. A versioned name for one thing the agent can do, such as `web.nginx.v1` or `dns.bind9.v1`. A message names exactly one capability, and an unregistered name is rejected before its contents are read (chapter 6).

Control plane. The Laravel and React application that holds what should exist: accounts, quotas, the audit log and the queue of jobs. It never touches a server directly.

DKIM. A way of signing outgoing mail so that receivers can check it. The public key is published as a DNS TXT record (chapters 17 and 18).

Eval (in Vesta). A shell command that runs a string as code. Vesta reads its records back this way (chapter 2).

Generation. A numbered, stored version of a rendered configuration. A failed change is rolled back by re-activating an earlier one (chapters 8, 9 and 15).

Idempotent. Safe to run twice with the same result. A repeated operation with the same key returns the first result (chapter 6).

Installer contract. The written rules every installer must follow: no interactive prompts, no unpinned downloads, a dry run, a preflight, structured output and deterministic exit codes (chapters 4 and 14).

Manifest. A small JSON file per service that declares what it provides, what it needs, its ports and its directories. Install order is worked out from the manifests (chapter 5).

Outbox. A table of jobs to be sent to nodes, written in the same database transaction as the change that caused them, so that none is lost and none is sent early (chapter 7).

Pest, PHPStan, Pint. The PHP test runner, static analyzer and code style tool that run on every push.

Phase. One numbered unit of work in the decision log, 1 to 55. The plan's own seven phases are called "the plan's phases" (chapter 3).

Plan mode. A state in Claude Code where the tool will not let the model change files until someone exits it (chapter 7).

Prune. The installer mode that removes services and packages not in the selection, except a protected list (chapter 20).

Resync and restore. Two ways of recovering a node. Resync re-sends the configuration from the control plane's database. Restore puts back the two things that cannot be rebuilt, received mail and database contents (chapter 18).

Vertical slice. Taking one service all the way from installer to interface and tests before starting the next (chapters 4 and 9).

Vibe coding. Building software by describing what you want and letting the model write the code. In this book: all changes made by prompt, with hand edits only when a prompt asks for one (chapter 1).

Where to read more

- **The code.** github.com/mikho/LEStA (public).
- **The discussion thread on LowEndSpirit.** "Vibe coding a VestaCP alternative", lowendspirit.com/discussion/11268/vibe-coding-a-vestacp-alternative. Members' questions, pushback and the running commentary. Members are not named in this book. To find a quote, search the thread for the phrase.
- **The build-log thread on LowEndSpirit.** "LEStA - the vibe coded version of VestacP", lowendspirit.com/discussion/11288/lestra-the-vibe-coded-version-of-vestacp. Closed. The same project told part by part as it was built. This book is the whole story in one place, and it goes further than the thread does.
- **The thread where the bet was made.** The comment chain ending in "Challenge accepted", dated 2026-08-25, LowEndSpirit comment 240245.
- **Vesta's history.** The VulnCheck advisory for CVE-2018-25117, Hackread's coverage of the 2018 attack, SSD Secure Disclosure's advisory of 2021-03-20 for CVE-2021-30462 and CVE-2021-30463, Outroll's announcement of 2024-03-16, and the Vesta site's page on the 2.0 rebuild. Full links are at the end of the preface.
- **The Hestia fork.** A LowEndTalk thread from July 2019, "Hestia Control Panel (VestaCP Fork)", and the Hestia 1.0 release of June 2019.

Image credits

The seven part-opener photographs are from Pexels (pexels.com) and are free to use under the Pexels license. They were chosen from search results and used in a reduced size.

- Part one: minimalist hourglass, pexels.com/photo/hourglass-8573370
- Part two: floor plan, pexels.com/photo/a-photo-of-a-floor-plan-4458205
- Part three: motherboard, pexels.com/photo/selective-focus-photography-of-motherboard-1432673
- Part four: light trails on a road at night, pexels.com/photo/long-exposure-photography-of-road-at-night-1699588
- Part five: cargo containers, pexels.com/photo/stack-of-different-colors-of-cargo-containers-12770545
- Part six: padlock on a metal gate, pexels.com/photo/close-up-of-a-secure-padlock-on-metal-gate-30844016
- Part seven: empty road under a blue sky, pexels.com/photo/empty-asphalt-road-under-blue-sky-14145081

All diagrams, charts and the screenshots (which show fake demo data) were made for this book.