CLAROVIEW

Zulip

How sustainable is this system for what its owner needs from it, and what would it take to close the gap?

CLIENT
None: an example on a public open-source project

SYSTEM
Zulip, open-source team chat (server and web app)

ASSESSED
23 September 2026 · system version main · commit e94efe61feb06d67ba6088929673348264644b1b (2026-09-23)

ISSUED
5 November 2026 · version 1.3

SCOPE
The product and one deployment of it: the zulip/zulip repository at commit e94efe61fe with its full history, and Zulip Cloud's configuration in it (puppet/kandra)

EVIDENCE
18 scanners and 394 findings, checked against the code; a dependency audit; CI history

ASSESSOR
Claroview · claroview.io

STATUS
Example report. The scoring is a first pass and was not validated with Zulip's technical team, so individual capabilities may be scored wrongly.

MODEL
AI Readiness Maturity Model v1.1

© 2026 Claroview ApS. All rights reserved.

CLAROVIEW

AI READINESS ASSESSMENT · ZULIP · WHERE IT STANDS

Zulip: a mature project that people hold together

Mature engineering, held together by **people rather than guardrails**.

SYSTEM	Team chat, self-hosted and run as Zulip Cloud
SIZE	about 370,000 lines · mostly Python and TypeScript
BUILT BY	1,455 contributors since 2012 · 65 in the last 90 days
EVIDENCE	18 scanners · 394 findings · full git history

Overall

L1

Fragile

Quality depends on **luck**. Set by the weakest dimension.

0 of 7 dimensions meet what this system needs.

What it needs (proposed)

Criticality	medium
Always on	essential
Protected	essential
Correct	important
Fast at scale	essential
Easy to change	essential
Connected	essential
Volume (measured)	large

Rated by Claroview, owner to confirm. It holds organisations' private conversations around the clock, and apps and integrations depend on its API.

How solid is this result?

98 capabilities checked: **35** evidenced, **55** absent, **8** unknown or awaiting the owner. Unknowns never count as a pass. All eight are for the owner, in one conversation.

Each dimension: the level now, and the level it needs ◇

Code	**L1 Fragile** · needs L4	gap of 3	**Held at L1** by import cycles between the Django apps.
Verification	**L2 Workable** · needs L4	gap of 2	**Next:** a gate under ten minutes, with business rules tied to tests.
Delivery	**L2 Workable** · needs L4	gap of 2	**Next:** every merge deployed automatically, not started by hand.
Operations	**L2 Workable** · needs L4	gap of 2	**Next:** a timeout, a backed-off retry and a limit on every call.
Data	**L1 Fragile** · needs L4	gap of 3	**Held at L1:** nothing maps where personal data is stored.
Security	**L1 Fragile** · needs L4	gap of 3	**Held at L1** by findings that are not published.
Knowledge	**L2 Workable** · needs L4	gap of 2	**Next:** the docs checked against the code automatically.

L1 L2 L3 L4 L5

A level says what quality depends on: L1 luck · L2 people · L3 defaults · L4 guardrails · L5 feedback.

WHY IT MATTERS, IN ONE FINDING

What level 4 would catch: a test step switched off since 2023

One step of the end-to-end message-basics suite, navigation from the home view, begins with `return; // No idea why this is broken.` It landed in February 2023. The suite runs in CI on every change, and that step has passed ever since without checking anything.

At level 4 a test known to be unreliable is marked or listed as quarantined, so it is visible as switched off instead of counting as a pass. For AI agents, which read a green run as permission to go on, that difference matters.

An example on a public open-source system. The scoring is a first pass and was not validated with Zulip's technical team; individual capabilities may be scored wrongly.

Next: why, per dimension →

© 2026 Claroview ApS. All rights reserved.

Strong on tests and build, short on guardrails

CLAROVIEW

DIMENSION	WHAT ALREADY HOLDS	WHAT IS MISSING TO REACH THE NEED
Code Can we change it at reasonable cost? L1 · needs L4 · gap of 3 · 8 short L2 ■□□■ L3 □□■□□ L4 □■□□ L5 □□□	Six components with one job each (five Django apps and the web client), matching the top-level folders. Strict static checking (ruff, mypy strict, ESLint) is green on every push to main, and every runtime and framework is in support.	**The apps import each other in cycles** at module level: zerver with analytics and confirmation, corporate with zilencer (import graph of the production code). Maintainability Cloud rates it 1.96, below the 2.5 a system of its size needs; nothing in the pipeline measures it or proposes updates.
Verification Would we know if a change broke it? L2 · needs L4 · gap of 2 · 6 short L2 ■■ L3 ■■□□ L4 □□□□ L5 □□	160 backend test files against real PostgreSQL, 178 frontend unit test files, 20 end-to-end suites and fixture tests for every webhook integration, on every pull request and every push to main.	**The gate takes 32 minutes** wall-clock (median of the last 20 green CI runs), and required checks bind non-admins only. Apart from Markdown rendering and the API schema, no document names the business rules and the tests that guard them.
Delivery Can we ship a change safely and repeatably? L2 · needs L4 · gap of 2 · 3 short L2 ■■■ L3 □■■■■ L4 □■■□ L5 □□□	Full history since 2012, builds and CI install strictly from the lockfiles, a scripted install and upgrade, tagged releases, and a CI job that installs and upgrades the real release tarball on five OS images.	**Deploys are started by people, not by the merge:** main reaches chat.zulip.org several times a week and Zulip Cloud one to two weeks later (release-lifecycle docs). Review is rarely recorded: two of the last ten merged pull requests had a formal approval. Reversing a migration means restoring a backup.
Operations Do we know when it is unwell, and does it survive faults? L2 · needs L4 · gap of 2 · 8 short L2 ■■■ L3 ▨■□□□ L4 □□■□□ L5 ⋰⋱□	Failures are contained: a catch-all at the request boundary, supervised processes that restart, a time limit per event for most queue workers. Zulip Cloud's own configuration pages on-call and exports per-endpoint metrics.	**Some calls can wait forever:** 6 of 27 connection kinds set no timeout (read in the code and the locked libraries). 8 of 21 external services retry with backoff, and 11 of 33 list endpoints return whole sets, most realm-sized by design. No service-level objectives or load test.
Data Can we evolve, protect and restore the data? L1 · needs L4 · gap of 3 · 6 short L2 ▨■□ L3 ▨■■▨ L4 ▨□□ L5 □□	Migrations run in order (952 files across six apps), an empty database is installed from scratch in CI on every push to main, deleted messages stay restorable by an admin for 30 days, and export and deletion paths exist.	**No document maps personal data to where it is stored** (the privacy policy lists categories only). Backups are scheduled once configured; only the owner can show that production has them and restores them. 13 of 65 multi-step writes lack a transaction or compensation (read in the code).
Security Is it protected in proportion to what it holds? L1 · needs L4 · gap of 3 · 9 short L2 ▨■■□ L3 □▨□■□ L4 □□■□ L5 □□	Every non-public route (about 320, counted from the URL files) needs a login, API key or server credential. No critical or high advisory reaches code that is served or run; 33 sit in build tooling (lockfile audit, traced by dependency path).	*Security findings are not published.*
Knowledge Could someone new, human or AI, pick this up? L2 · needs L4 · gap of 2 · 8 short L2 ■■ L3 □□■□□ L4 ■□□□□ L5 □	About 64,000 lines of maintained developer documentation, an OpenAPI contract checked in CI, and context files for AI agents. The documented setup worked first time on a clean container (assessor-run, 23 September 2026).	**Nothing checks the docs against the code:** 8 of 71 architecture statements in the docs are stale, and half-done renames have no recorded target state. 77% of changes reach an issue or link through their message or pull request (bar: 90%).

■■□ level now ◆ level needed ■ evidenced ■ owner-attested □ absent ▨ unknown ⋰⋱ not asked: above the need one square = one capability

explained on page 5

An example on a public open-source system. The scoring is a first pass and was not validated with Zulip's technical team; individual capabilities may be scored wrongly.

Next: what to do →

© 2026 Claroview ApS. All rights reserved.

Four moves close most of the gap

Each move builds on the one before; the last turns good habits into guardrails.

CLAROVIEW

1

Clear the level-2 blockers.

Break the three import cycles, and write down which personal data lives where. *Security measures are not published.*

Code reaches L2; with the owner's confirmation of production backups, so does Data.

2

A faster gate, tied to the rules.

Bring the gating checks under ten minutes, list the core business rules with the test that guards each, and deploy every merge to main automatically to a test server such as chat.zulip.org.

Verification and Delivery reach L3.

3

Bound every call.

A timeout on the six connection kinds without one, backed-off retries for queue clients, paging or limits on the lists that grow with use, error handlers in the browser client and the scheduled jobs, and the owner's confirmation of what calls the health check.

Operations reaches L3.

4

Turn review into guardrails.

Required checks and reviews that bind administrators too, dependency and secret scanning on every change, updates proposed automatically (which would also clear the build-tool advisories), an import rule between the apps, and decision records for the renames in flight.

Moves Code, Security and Knowledge towards L3 and L4, and Verification and Delivery towards L4.

READY FOR AI AGENTS

How far an AI agent can be trusted to change this system

The same five levels, read for the agent harness: what an agent depends on to work safely.

L1 — ZULIP TODAY	L2	L3	L4	L5
Fragile	**Workable**	**Dependable**	**Resilient**	**Self-sustaining**
An agent works from what someone types: nothing written down guides it, and nothing stops it.	An agent can find its way: context to read, one command for the tests, no secrets in reach. Somebody still checks everything it does.	An agent works against written rules, including what must never happen, and the tests tell it when it broke one.	An agent cannot do harm unnoticed: its reach is limited, every change is reviewed, a failing check stops it.	The guidance follows the system: an agent's context is regenerated from it, and every defect leaves a test behind.

Held at L1 by a security finding that is not published.

An example on a public open-source system. The scoring is a first pass and was not validated with Zulip's technical team; individual capabilities may be scored wrongly.

Next: how to read this report →

© 2026 Claroview ApS. All rights reserved.

How to read this report

A level says what quality **depends on**. The further right, the less it depends on anyone remembering, and the safer the system is to change at AI speed.

CLAROVIEW

L1	L2	L3	L4	L5
Fragile	**Workable**	**Dependable**	**Resilient**	**Self-sustaining**
Quality depends on **luck**.	Quality depends on **people**.	Quality depends on **defaults**.	Quality depends on **guardrails**.	Quality depends on **feedback**.
It works today. Nobody can say why it would tomorrow.	Good practice exists, and stops when the person who does it is busy or leaves.	Doing it right takes no extra effort, so it happens every time.	Doing it wrong is blocked, so it cannot happen silently.	The system notices its own decline and starts the repair.

Reading a dimension

Code 

L1 · needs L4

L2 ■□□■ L3 □□■□□ L4 □■□□ L5 □□□

- **Filled segments**: the level now. **Yellow diamond**: the level needed. Short of it is a *gap*, on it is *met*, beyond it is *sufficient* and never offsets a gap elsewhere. *6 short*: six capabilities are still missing up to the needed level.
- Each **small square** is one yes-or-no capability, grouped by level. A level is reached only when all its squares, and all before it, are filled. Faded groups lie beyond the need.

■ **Evidenced** – an artefact proves it

□ **Absent** – looked for, not there

⬚ **Not asked** – only the owner knows, and it lies above the need

Attested – owner states it; counts, flagged

Unknown – never counts as a pass

How the needed level is set

- **The owner's answers decide**, not a generic ideal. **Criticality** sets a floor for every dimension; each **need** marked important or essential lifts the dimensions it depends on; measured **volume** lifts those that size makes harder.
- The highest that applies wins. No weights, no formula: the *Set by* column names the one answer behind each requirement.

NEED	ISO/IEC 25010:2023 CHARACTERISTIC
Always on	Reliability
Protected	Security
Correct	Functional suitability
Fast at scale	Performance efficiency; Flexibility (scalability)
Easy to change	Maintainability; Flexibility (adaptability)
Connected	Compatibility (interoperability, co-existence)
Criticality	ISO/IEC 25019:2023 Freedom from risk

Business questions, traceable to the standards rather than renamed after them. Interaction capability and safety are out of scope.

How the result is established

- **The overall level is the weakest dimension.** Strengths are shown but never average away a gap.
- **Capabilities, not opinions.** 98 yes-or-no capabilities, checked against the system's own artefacts. Mechanisms, not tidiness.
- **Levels are cumulative.** One missing L2 capability holds a dimension at L1.
- **Architecture is measured where it shows:** its structure in Code, its fit in the dimension each need drives (e.g. Operations).
- **Stars ★** are reserved for the Maintainability Cloud score, which anchors Code.

Basis

Maintainability Cloud scoring based on Sokrates analysis with git history and the Claroview AI scanners, every finding checked against the code, assessed against the rubric. The system owner attests what artefacts cannot show.

This report describes the system as it stood on the assessment date. It is not a certification, a penetration test or a legal opinion.

The capability rubric behind these levels is maintained by Claroview and applied identically to every system it assesses.

© 2026 Claroview ApS. All rights reserved.