From GitHub App to your first baseline

Guide · Watchdog **Version 1.1 - 2026-08-04** Author: Canine Development

Who this is for: the engineer or engineering lead connecting a codebase to Watchdog for the first time. It covers what the first survey costs and what it commits you to, what the analysis looks at, how to read the result, and which report to open first.

Watchdog reads your code and never writes to it: no pull requests, no commits, no branches. You connect a repository, and the first survey starts automatically. Here is that path, and what to do with the report that comes back.

---

1. Install the read-only GitHub App

Start at watchdog.canine.dev and connect GitHub. What you are installing is a GitHub App with **read access only** - enough to list the repositories your installation can reach and to clone them for analysis. The clone uses a short-lived read token that is never persisted. There is no write path: Watchdog comments on none of your pull requests and changes none of your settings.

When GitHub asks you to choose repositories, you can grant the App access to a single repository or a wider selection. The installation grants read access only to what you select, and you can narrow or widen that selection later.

If you would rather not connect the App, you can add a public repository by typing its `owner/name` instead. Connecting the App means you choose from the repositories available to the installation rather than entering one manually.

---

2. Track a repository - and know what that starts

Choose a repository and track it. A good first choice is one you know well, because you can compare the report with your existing understanding of the code.

Tracking starts two things.

The first is the baseline. You do not need to start a separate scan: tracking queues the first survey immediately, and that survey is free.

The second is a recurring schedule. If the repository belongs to a team, it is surveyed at the end of each of that team's sprints; a repository with no team of its own sits on a default 14-day calendar. **Those later surveys are metered against your plan.** Only the baseline is free.

The accurate commercial summary is therefore: the first report costs nothing and requires no card, while tracking also activates a recurring paid cadence that you can pause. If the baseline is all you want, open the repository's Settings tab and pause analyses after the report lands. Watchdog is priced by lines scanned per month rather than per seat, so what the survey measures is also what determines the price of the surveys that follow. See the current terms on the pricing page.

The free baseline is granted **once per repository identity**. It is keyed to the repository's permanent identity at the provider, not to your account, so untracking and re-adding it does not create another free baseline. If you choose a public repository that somebody else has already tracked, its free report may already have been used. One of your own repositories is therefore the safest first choice.

---

3. What the survey looks at

The survey clones your code read-only, runs the analysis and produces a report. It is not a reduced version of a paid survey: **depth is never gated.** The baseline computes the full index across every lens and dimension that applies to the repository, exactly as a paid survey does. A plan buys breadth and cadence, not depth.

Three things govern how much of the model can be measured on your code:

- **Languages.** As published on 4 August 2026, the production catalogue held **13 languages**: 12 analysed deeply and JavaScript analysed structurally. The catalogue is a live production record and may change, so check the current version at watchdog.canine.dev/api/public/language-support.
- **Dimensions and lenses.** The published rubric contains **127 entries across ten lenses**. Forty-two are citable dimensions, identified as `D1` to `D42` - the codes a finding cites. The remaining 85 are meta-dimensions that contribute to a lens without being cited directly on a finding. Of the 127, 111 are measured by deterministic tools and 16 carry an advisory language-model read. Each survey records the rubric version it used, and the authoritative catalogue for that run is at `cai.canine.dev/api/rubrics/<rubric-version>/catalog`; the current one is browsable at cai.canine.dev/dimensions.
- **Applicability.** Not every dimension applies to every repository, and the survey scores the ones that do. A dimension that cannot be measured reports *not measured* with the reason attached rather than a zero it did not earn. The report also states how many applicable dimensions returned a result.

Time to first report. Surveys run asynchronously, and expect minutes rather than seconds - longer on a large or complex codebase, and longer again at a busy moment, because a survey is served from a queue. You can close the tab while the survey runs and return to the repository page when the report is ready.

---

4. Read your Codebase Assurance Index

The survey returns a **Codebase Assurance Index (CAI)**: a single 0-100 headline showing how the repository performs against the CAI rubric for software build quality. It maps onto five bands at fixed cutlines of **90 / 70 / 50 / 25**:

Band	CAI
Exemplary	90-100
Strong	70-89
Adequate	50-69
Weak	25-49
Critical	0-24

The headline is where you start; the per-lens breakdown underneath it shows where to investigate.

The ten lenses, and which apply to you

Five are always on: Code Health, Architecture, Maturity, Readiness, and Security & Compliance. Every codebase is scored on all five.

Five are conditional: Domain Modelling, Event-Driven, Event Sourcing, Accessibility, and Performance. These contribute only when they apply, and the weights re-normalise so a repository is not penalised for an inapplicable lens.

Two mechanics are worth knowing before you read your own number:

- **A critical lens caps the headline.** The roll-up cannot read Strong while a lens reads Critical, so one serious failure cannot be averaged away by strong scores elsewhere.
- **An advisory language-model read cannot move the number.** Where one is present, it can help explain a result, but it does not contribute to the score.

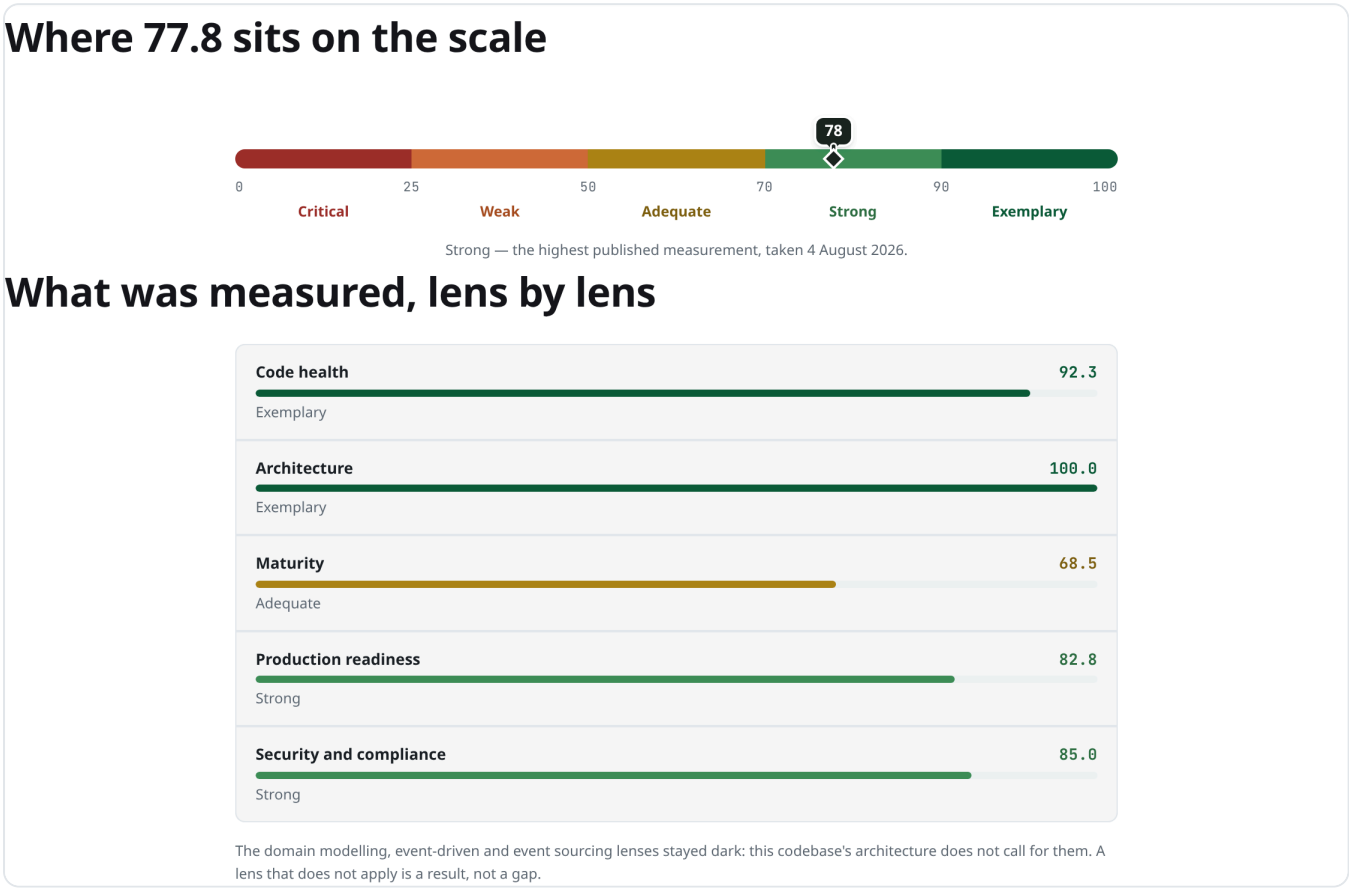
A worked example you can open

Watchdog publishes a corpus of measured open-source projects that you can read without an account: the whole corpus at cai.canine.dev/surveys, one language at a time, and a page per project carrying its score, its band, its trend and its per-lens gauges, with a link to the full report.

Take the published survey of the Python dependency-injection library `vshulcz/injex`, as it stood on 4 August 2026:

> **77.8, Strong.** Architecture 100.0 and Code Health 92.3, both Exemplary. Security and compliance 85.0 and Production readiness 82.8, both Strong. Maturity 68.5, Adequate, and the weakest lens in the report.

Check that against the band table above and it reconciles directly: 100.0 and 92.3 clear the 90 cutline, 85.0 and 82.8 sit in the 70-89 band, and 68.5 falls in 50-69.



The same survey as the page renders it: 77.8 pinned on the fixed 0-100 scale, and the five lenses underneath it. The note below the lenses is the conditional-lens rule in the product's own words - three lenses stayed dark because this codebase does not call for them, and a lens that does not apply is a result rather than a gap.

Nothing is being averaged away: a well-architected library is carrying a documentation and process gap, and the number says so. That survey shows only the five always-on lenses because no conditional lens applied. Compare it with the Rust survey of `jj-vcs/jj`, where Domain Modelling appears as a sixth scored lens because the code has a domain model to assess. Same rubric, different applicable surface.

The practical reading is the same on your own repository: start with the weakest applicable lens rather than treating the headline as the diagnosis.

Which report to open, and what to do next

A survey produces four views of the same run. **Open the Engineering report first** if you are reviewing the code and deciding what to improve - it is the one written for the people who will change it. The Executive

report is the summary to hand upwards, the Security report collects the security lens, and the Audit report records the run parameters and scoring scope described in the next section.

Then work in this order:

1. **Go to the weakest lens the report names**, not the headline. On the example above that is Maturity at 68.5.
2. **Read the findings within that lens.**
3. **Use each finding's dimension code** to consult the catalogue for the rubric version recorded in the Audit report.
4. **Separate measured findings from measurement limits.** A dimension marked *not measured* includes its reason; it is a gap in what could be assessed, not a fault in your code, and it does not count as zero.
5. **Choose a manageable first improvement**, rather than trying to respond to the entire score at once.

Three things to hold in view

- It is a **measurement, not a certification**, and the report states that scope directly.
- It reads your source, history, configuration and manifests. Outside-in penetration testing and DAST are a separate and complementary exercise.
- It needs nothing else installed, and complements the engineering controls you already run rather than replacing CI, linters or scanners. Where you have those, it looks at what they do not - architecture, change-safety, knowledge concentration - and leaves their checks to them. Those history-derived signals describe the codebase and where its knowledge sits, not the people who wrote it.

---

5. What makes the report trustworthy

Every run records what is needed to identify it: the **commit analysed**, the **start and finish timestamps**, the **rubric version**, and the **analyzer image**. The Audit report prints those parameters, lists the lenses included in the run, states whether any lens or dimension was disabled, and carries a section on what changed since the previous scan - on a first run it says so, because there is nothing yet to compare against.

Two different checks sit behind the word "verifiable", and they are worth keeping apart:

- **Repeating a survey.** The same commit, under the same frozen rubric and analyzer image, is expected to produce the same number. The recorded parameters identify the conditions required for that comparison, and the method is published at cai.canine.dev/verify. The open reference scorer needed for an external party to reproduce the complete survey independently is not currently distributed, so independent end-to-end reproduction is not yet publicly available.

- **Verifying a shared delivery.** When a result is packaged and sent to somebody, that package is content-hashed and Ed25519-signed. Checking the signature against the issuer key shows where the package came from; recomputing the hash over the manifest shows whether its contents changed after signing. That check anyone can run today.

Every Watchdog report carries a **"NOT ATTESTED"** verdict rather than a pass mark. This states the scope of the report: Watchdog measures the repository against the CAI rubric but does not certify it against agreed acceptance criteria. Binding a report to declared criteria is an Assay capability, not part of a Watchdog survey.

---

6. Where to go next

- **Pick one lens and improve it.** Use the Engineering report to identify a focused first action; the next survey's Audit report will tell you what changed against this baseline.
- **Decide the cadence you want.** Tracking has already set one - your team's sprint end, or the default 14-day calendar. Keep it if it is useful, or pause analyses on the repository's Settings tab if the baseline was all you wanted.
- **Use later surveys as comparison points.** A later report gives you another measured result to compare with the baseline.
- **Widen access** to more repositories from the same read-only installation when appropriate.

Survey a repository - the first report for an eligible repository is free and requires no card. Later scheduled surveys are metered against your plan.

Learn more / verify

Start here

- The product and your free first survey: watchdog.canine.dev
- What surveys cost after the first: watchdog.canine.dev/pricing

Real examples

- The measured open-source corpus: cai.canine.dev/surveys - each project page carries its score, trend, lens gauges and full report
- Public Watchdog reports: watchdog.canine.dev/publicreports

The live catalogues

- Languages and analysis tiers: watchdog.canine.dev/api/public/language-support

- Every dimension, per rubric version: cai.canine.dev/dimensions, and for one survey `cai.canine.dev/api/rubrics/<rubric-version>/catalog`

The standard, and checking it

- How the lenses roll into one number: cai.canine.dev/spec
- How a CAI is checked: cai.canine.dev/verify