Engineer

Quality gate

a set of checks code must pass before it moves on

What it is

A quality gate is a set of conditions a change has to meet before it may move to the next stage, with one result at the end: pass or fail. The stage can be a commit, a merge into the main branch, a release or a deployment. The conditions are anything a machine can decide: a clean lint run, passing tests, a coverage figure, a build that completes.

The word doing the work is gate. A check reports; a gate decides, and the decision is enforced by whatever controls the next step. A linter that prints forty warnings into a log is a check. The same linter wired so that the merge button stays disabled until its output is clean is part of a gate.

SonarQube Cloud, where the term is a named feature, describes a quality gate as a set of conditions against which code is measured during analysis. Each condition pairs a metric with a comparison operator and an error value, so “reliability issues greater than 0” is a condition, and if any condition is met, the gate fails. A condition applies either to new code or to the code as a whole.

The hosting platforms build the same idea into their merge flow:

- **GitHub** has a branch protection setting, “Require status checks before merging”. A required check must report a successful, skipped or neutral status before changes reach the protected branch.
- **GitLab** has a merge check called “Pipelines must succeed”. Once it is selected, a merge request cannot merge while its pipeline is failing, and it also cannot merge when there is no pipeline at all.
- **On a developer’s own machine**, pre-commit installs checks as Git hooks that run at commit or at push, and a task runner such as Task gives the whole set one name.

Use the concept in a decision

Put “What is a quality gate?” to work in a real decision. We can help relate the concept to the system you are building or buying. Tested, released and reversible, so changes reach customers on a Tuesday afternoon rather than late on a Friday night.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated GIS SaaS deployment with GitHub Actions — deployment, data and reporting, all off the hands

Automated GIS SaaS deployment with GitHub Actions Get in touch →

Why it matters

A standard that lives only in a document gets negotiated one exception at a time. A standard that fails the build is followed by everyone, including the person who wrote it, on the afternoon when they would rather skip it. A gate makes agreed rules executable, so keeping them stops depending on memory and mood.

It also changes what human review is for. When a machine has already caught formatting, unused imports, broken links and leaked secrets, the reviewer can spend their attention on intent and design, which a machine cannot judge. The bar is also the same for every author.

The new-code idea makes a gate adoptable on an old codebase. Sonar way, the default gate in SonarQube Cloud, focuses on new code rather than on remediating old code, with conditions such as at least 80.0% test coverage and at most 3.0% duplication in new code. On a pull request, only conditions about new code are used. A project with thousands of existing issues can switch the gate on and hold every new change to the bar without first paying off the backlog.

Speed decides whether people keep the gate switched on. Task runs a task’s dependencies concurrently, so a gate built from many independent checks can run them side by side instead of one after another. Splitting the checks by cost helps too: the cheap half on every commit, the full set before code leaves the machine.

This site has its own gate: `task check`, run on the machine the site is written on rather than on a hosted build service. Every diagnostic counts as an error, the site generator runs with `--panicOnWarning` so that even a deprecation notice stops the build, and inline suppressions such as `# noqa` or `stylelint-disable` are prohibited, so a failure gets fixed at its cause instead of silenced. A pre-commit hook runs the file-level checks on every commit, and a pre-push hook runs the

full `task check`, which takes minutes rather than seconds. A green run from a clean tree stamps that tree, and the deploy script refuses a tree without a stamp. Its one override is an environment variable that prints a loud warning, so whoever skips the gate in an emergency has to say so.

What it looks like in practice

The site's credits page lists the tools inside that gate: vnu for HTML, Stylelint, axe-core across a page per template, Lighthouse, lychee for links, Prettier, markdownlint, ESLint, Ruff, basedpyright, yamllint, ShellCheck, codespell and gitleaks. Task runs all of it under one name per job, and pre-commit runs the file-level half again on every commit. The page also names what stays outside the gate: two of the browser checks driven by Puppeteer, viewport overflow and the visual regression, which a person runs on demand.

A one-person project has no reviewer at all, which is the situation one portfolio page starts from. The standards had to be executable, so that breaking one fails the commit instead of waiting for someone to notice. The answer, at the documented August 2026 implementation stage, was a commit gate of 22 linters with no warning tier, plus project-specific checks that no off-the-shelf tool has an opinion on, such as that the deployed tree holds no private path or hostname and that the copy kept its tone rules in three languages. A failure already present is fixed in the same pass rather than accepted. The page names the cost plainly: committing is slow, and the gate is the only reviewer the project has.

A gate can also disagree with itself. Another page describes a project with a commit hook and a full gate, a document saying the hook ran the gate, and two lists of checks kept by hand in two files. A test that resolves both lists to the checks they actually invoke showed five checks running only in the full gate, and they were exactly the ones that read the prose. Every check protecting code ran automatically; every check protecting the writing ran only when someone remembered. The five moved into the hook, two of them were made fast enough to fit its time budget, and the comparison test stayed to catch differences between the lists.

Where it goes wrong

Most gates fail by letting things through while still showing green.

Warnings pile up behind a pass. A gate that fails only on errors lets warnings accumulate until the log is too long to read, and then a real problem arrives as warning number two hundred and one. Drop the warning tier: either a rule is worth failing on, or it is switched off.

A skipped check counts as a pass. On GitHub, a skipped status satisfies a required check, and a job skipped by its `if:` condition reports success. A job that depends on a failed job is skipped too, and may not block the merge. GitHub's advice is to use `always()` with `needs`: require one job that always runs and passes only when everything it needs succeeded.

```
# .github/workflows/check.yml (excerpt)
jobs:
    gate:
        if: always()
        needs: [lint, test, build]
        runs-on: ubuntu-latest
        steps:
            - run: |
                  for result in ${{ join(needs.*.result, ' ') }}; do
                      test "$result" = success || exit 1
                  done
```

The opposite trap also exists: a required workflow skipped by a path or branch filter leaves its checks pending and blocks every merge. Avoid requiring workflows that can be skipped.

Two lists drift apart. The hook runs one set, the server another, and a document claims they match. Give the checks one definition and call it from every place, or keep a test that compares the lists and fails when they differ.

The gate gets slow and people go around it. A flag that skips hooks is at hand, and on GitHub branch protection does not apply to repository administrators by default; "Do not allow bypassing the above settings" changes that. Measure where the time goes, run independent checks concurrently, keep the commit-time half small, and move a genuinely slow check to a documented manual step.

A check passes because it checked nothing. A linter pointed at a glob that matches zero files, or an ignore pattern that swallows a whole directory, exits cleanly, and a clean exit looks the same as clean code. Test the gate itself: give

each check a file that is known to be wrong and expect it to fail.

Further reading

- Introduction to quality gates — conditions, the Sonar way default and the focus on new code.
- About protected branches — required status checks and who may bypass them.
- Troubleshooting required status checks — which statuses count as passing, and how skipped workflows and jobs behave.
- Auto-merge — GitLab's "Pipelines must succeed" check and how it treats skipped pipelines.
- pre-commit — checks declared once and installed as Git hooks for the commit and push stages.
- Task dependencies — how Task runs a task's dependencies concurrently and its commands in order.



Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a quality gate?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/quality-gate/