GitHub Actions & Automated Checks in Mesa

Boyu Wang

2026.01.11

Outline

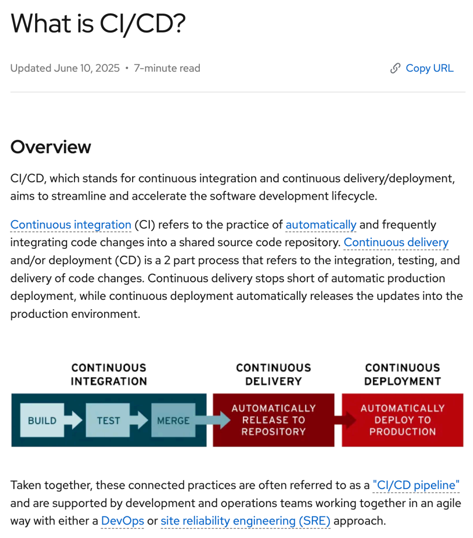
Continuous Integration & Continuous Delivery/Deployment

Motivation:

- Catch bugs early (before merge) and keep `main` branch releasable.
- Make reviews faster: checks provide a baseline for correctness and style.
- Standardize "works on my machine" into "works in a controlled environment".

Key terms:

- **CI (Continuous Integration):** automated build + tests + lint on each PR/push.
- **CD (Continuous Delivery/Deployment):** automated packaging/release once code is approved.



What is CI/CD?

Updated June 10, 2025 • 7-minute read

Copy URL

Overview

CI/CD, which stands for continuous integration and continuous delivery/deployment, aims to streamline and accelerate the software development lifecycle.

Continuous integration (CI) refers to the practice of automatically and frequently integrating code changes into a shared source code repository. Continuous delivery and/or deployment (CD) is a 2 part process that refers to the integration, testing, and delivery of code changes. Continuous delivery stops short of automatic production deployment, while continuous deployment automatically releases the updates into the production environment.

Taken together, these connected practices are often referred to as a "CI/CD pipeline" and are supported by development and operations teams working together in an agile way with either a DevOps or site reliability engineering (SRE) approach.

Source: What is CI/CD? *by Red Hat*

Environments

An **environment** is a deployment target with its own *config*, *dependencies*, and *data* (so you can test changes safely before they reach users).

Common environment types (vendor / org names vary):

- **Local dev:** your machine; fastest feedback (debugging, quick experiments).
- **Test / QA:** automated + manual testing; may use synthetic / anonymized data.
- **SIT (System Integration Test):** validate cross-component integration end-to-end (common in stage-gated orgs).
- **Staging / pre-prod:** production-like environment for release validation (config parity, performance smoke tests).
- **Prod:** real users + real data; highest change-control.

Two example workflows:

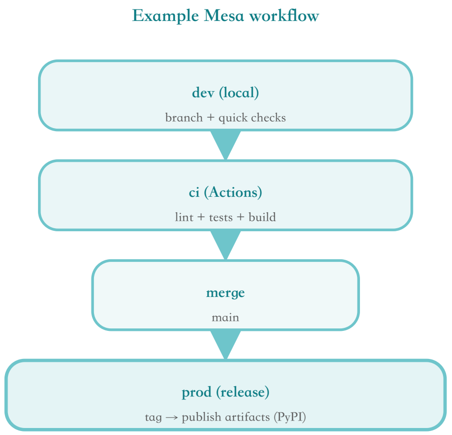
- **Traditional stage-gated (waterfall):** dev → QA/test → SIT → UAT → prod (clear gates, fewer releases).
- **Modern agile/DevOps:** dev → CI → preview/review apps → staging → prod (small PRs, frequent releases; feature flags/canary often used).

Learn more:

- ISTQB: Standard Glossary of Terms Used in Software Testing
- AWS: Continuous Integration and Continuous Delivery
- GitLab: Get started with GitLab CI/CD
- GitHub: Understanding GitHub Actions

Environments in Mesa

1. **dev** (local): an isolated Python env (venv/conda/uv/$\cdots$)
2. **ci** (GitHub Actions): clean machines ("runners") execute reproducible workflows triggered by PRs/pushes
3. **staging**-ish (optional): docs preview for PRs, release candidates (rc) or pre-releases, TestPyPI
4. **prod**: versioned release/tag, publish package artifacts (PyPI), publish docs (stable site)

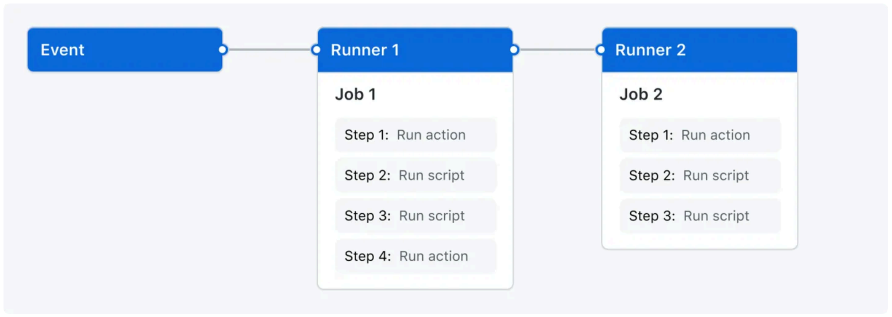


GitHub Actions

A CI/CD platform built into GitHub repositories: it runs **workflows** on GitHub-hosted or self-hosted **machines** when repository **events** happen.

Core concepts:

- **Workflow:** a YAML automation defined in `.github/workflows/` .
- **Event:** what triggers a workflow run (`push` , `pull_request` , `schedule` , ⋯).
- **Runner:** the machine that runs a job (GitHub-hosted or self-hosted).
- **Job:** a set of steps executed on the same runner; jobs can run in parallel or depend on other jobs.
- **Steps (inside a job):** each step is either a `run:` shell command or a `uses:` action.
- **Action:** a pre-defined, reusable unit that performs common tasks inside workflows.



Source: GitHub Docs - Understanding GitHub Actions

Mesa Workflows & Checks

Mesa's main workflows (from `.github/workflows/`)

- **`build_lint.yml`** : unit tests + code coverage + example runs
- **`release.yml`** : build & publish releases
- **`benchmarks.yml`** : performance benchmarks to detect slowdowns)
- **`vale.yml`** : docs prose/style checks (spelling, terminology, writing rules) using Vale

Each workflow file defines:

- **Workflow name**
- **Triggers**: when to run (PR / push / schedule / ⋯)
- **Jobs**: logical groups of work (test, lint, docs, release, ⋯)
- **Steps**: commands / actions (install deps, run pytest, upload coverage, ⋯)

"Supporting config" that *feeds into* checks (not all are Actions)

These aren't workflows themselves, but they control what checks do:

- `.pre-commit-config.yaml` : defines local/dev hooks; also used by **pre-commit.ci** (a 3rd-party CI service) to run the same hooks.
- `codecov.yaml` : config for Codecov.
- `.readthedocs.yml` : config for Read the Docs builds (docs check runs as an external service).
- `.coderabbit.yaml` : config for CodeRabbit automated review.

build: `build_lint.yml`

```
# simplified shape (not exact)
on: [pull_request, push, schedule]
jobs:
  build:      # matrix (os × python)
  coverage:   # coverage report → Codecov
  examples:   # run example models/tests
```

- **Purpose:** baseline PR gate - Mesa must install and tests must pass on common platforms.
- **Matrix:** runs across multiple OS + Python versions to catch platform/version issues.
- **Triggers:** pull requests, pushes to key branches (e.g., main / release*), and a weekly schedule.
- **Steps:**
  - build (install .[dev] + pytest)
  - coverage (pytest + coverage → Codecov)
  - examples (run example models).

Release: `release.yml`

```yaml
# simplified shape (not exact)
on: [push(tags), push(branches), pull_request, schedule]
# allows gh to authenticate and publish with PyPI
permissions: { id-token: write }
jobs:
  release:  # build dist + publish to PyPI (tag pushes only)
```

- **Purpose:** build Mesa distributions and **publish to PyPI** for tagged releases.
- **Triggers:** PR/push/schedule exist for visibility, but *publishing* only happens on **push + tag**.
- **Security:** uses OIDC (`id-token: write`) so PyPI publish can avoid long-lived API tokens.
- **Steps:** checkout → setup Python → install `hatch` → `hatch build` .
- **Artifacts:** on official repo + tag, uploads `dist/` as a GitHub Actions artifact ("package").
- **Publish:** runs `pypa/gh-action-pypi-publish` only when `startsWith(ref, refs/tags)` .

Performance benchmarks: `benchmarks.yml`

```yaml
# simplified shape (not exact)
on:
  pull_request_target:   # opened / ready_for_review / labeled
permissions:
  issues: write
  pull-requests: write
jobs:
  run-benchmarks:         # compare main vs PR + comment
```

- **Purpose:** run performance benchmarks and report a **main vs PR** comparison directly in the PR.
- **Trigger:** runs when PR is opened / marked `ready_for_review`, or when `trigger-benchmarks` label is added
- **Steps:** benchmark Mesa main → upload baseline timings results → checkout PR branch → benchmark again → compare → comment results.
- **Output:** posts a PR comment via `actions/github-script` (requires the workflow's write permissions).

Vale Prose Linter: `vale.yml`

```
# simplified shape (not exact)
on:
  pull_request:   # docs/text files only
  push:           # main, docs/text files only
concurrency:      # cancel older runs on same ref
jobs:
  vale:           # run Vale prose linter
```

What is Vale?

Vale is an open-source *prose linter* that enforces writing rules via config + rule sets ("styles").

- You can enable existing style packs (e.g., "Google", "Microsoft") or write your own project rules.
- Rules can check spelling, vocabulary, casing, headings, punctuation, and many other patterns in *documentation* and *text files*.

Learn more: Vale documentation

- **Purpose:** run **Vale** (prose/style linter) on documentation and text files to catch wording/style issues early.
- **Path-filtered trigger:** only runs when docs-related files change (`.md` , `.rst` , `.txt` , `docs/**` , `.vale.ini` , style rules).
- **Concurrency:** for the same PR/branch, new runs cancel in-progress older runs (keeps CI noise down).
- **Implementation:** checks out the repo, then runs `errata-ai/vale-action` with `files: all` and `--minAlertLevel=warning` .
- **Non-blocking by default:** `fail_on_error` : false means it's typically an advisory check (useful feedback without hard-failing PRs).

Dependabot: `.github/dependabot.yml`

```yaml
version: 2
updates:
  # Maintain dependencies for GitHub Actions
  - package-ecosystem: "github-actions"
    directory: "/"
    schedule:
      # Check for updates to GitHub Actions every week
      interval: "weekly"
```

Learn more

- GitHub Docs: Keeping your actions up to date with Dependabot

- **Dependabot** is not a GitHub Actions workflow. It's a separate GitHub feature that **opens automated PRs** to update dependencies.
- Once Dependabot opens a PR, your GitHub Actions workflows run on that PR (tests, lint, docs, etc.) like any other PR.
- **Steps:** Dependabot PR → CI passes → reviewer merges → dependency updates.

Other Automated Checks

These are **automations around PRs** that complement the GitHub Actions workflows, and some run locally too:

- **pre-commit** hooks & **pre-commit.ci:** hooks to enforce formatting/linting
  - **ruff**: fast lint + formatter
  - **pyupgrade:** auto-modernize Python syntax
  - **codespell:** catch common typos in code + docs
  - **Vale**: *prose linter* for docs (spelling, style, terminology consistency)
- **Codecov**: coverage reporting + PR status checks / diffs
- **Read the Docs**: docs builds + hosted docs, often with PR previews
- **CodeRabbit:** automated PR review comments (code reading + suggestions)

pre-commit
local + pre-commit.ci

ruff
lint + format

Vale
prose lint

Codecov
coverage

Read the Docs
docs build/preview

CodeRabbit
PR review bot

pre-commit: local hooks & pre-commit.ci

Same formatting/lint results locally and in PR checks:

- **pre-commit:** a framework to run checks **locally** before commits (see pre-commit.com)
- **pre-commit.ci:** a CI service that runs the **same config** on PRs (see pre-commit.ci)

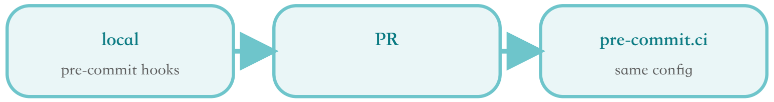
Steps

- **ruff**: lint + auto-fix; **ruff-format**: formatter. (Ruff Docs)
- **pyupgrade:** modernize syntax (Mesa config targets `--py312-plus`). (pyupgrade)
- **codespell:** catch common typos in code + docs. (codespell)
- **Vale**: run locally, skipped in ci (handled in a workflow `vale.yml`)

`.pre-commit-config.yaml` (simplified shape):

```
ci:
  autoupdate_schedule: monthly
  skip: [vale]    # Vale runs via vale.yml workflow instead

repos:
  - ruff:          lint (--fix)
  - ruff-format:   formatter
  - pyupgrade:     --py312-plus
  - pre-commit-hooks:
      trailing-whitespace
      check-toml
      check-yaml
  - codespell:     typos
  - vale:          prose lint (md/rst/txt)
```



Codecov

- Codecov is a **3rd-party coverage service**, but it plugs into GitHub Actions: Action runs tests & uploads coverage.
- Turns binary whether "tests passed" into coverage signals: project/patch coverage, diffs/annotations.
- The `coverage` job uploads via `codecov/codecov-action` and fails the workflow if upload breaks (`fail_ci_if_error: true`).
- Config: target 80% project coverage, small threshold window of 1%.

part of the `build_lint.yml` workflow:

```yaml
# simplified shape (not exact)
jobs:
  coverage:
    steps:
      # generate coverage report
      - run: pytest --cov
      # upload to codecov
      - uses: codecov/codecov-action@v5
        with:
          fail_ci_if_error: true
          token: ${{ secrets.CODECOV_TOKEN }}
```

`codecov.yaml` config:

```yaml
coverage:
  status:
    project:
      default:
        target: 80%
        threshold: 1%

ignore:
  - "benchmarks/**"
  - "mesa/visualization/**"

comment: off
```

Read the Docs

- Read the Docs (RTD) is a **3rd-party docs hosting/build service**, but it integrates with GitHub PRs.
- On PRs, RTD can build the docs and provide a **preview link** (a “review app” for documentation).
- The build is driven by repo config (`.readthedocs.yml`) plus docs toolchain (Sphinx/MkDocs).
- **Steps:** PR opened → RTD builds → preview available → merge → default branch rebuild → stable docs updated.

`.readthedocs.yml` config:

```yaml
version: 2

# Build documentation in the docs/ directory with Sphinx
sphinx:
  configuration: docs/conf.py

# Optionally build your docs in additional formats such as PDF
formats:
  - pdf

build:
  os: ubuntu-lts-latest
  tools:
    python: latest

# Optionally set the version of Python and requirements required
        to build your docs
python:
  install:
    - method: pip
      path: .
      extra_requirements:
        - docs
```

CodeRabbit

- **CodeRabbit** is an automated PR review bot: posts high-level summaries, file-by-file walkthroughs, and code suggestions directly on the PR. (coderabbit.ai)
- Repo config aims for "lightweight but helpful": `profile: chill`, summary + changed-file summary + optional sequence diagrams, plus suggested reviewers/labels.
- Auto-review is disabled (no unsolicited full reviews on every PR by default).
- It can run a toolbox of linters/checkers (e.g., ruff, shellcheck, markdownlint, GitHub checks) and provide results in review feedback.
- Optional "chat" features are enabled (auto-reply, can create issues), and it may use web search in its knowledge base.

`.coderabbit.yaml` (selected settings):

```yaml
reviews:
  profile: chill
  high_level_summary: true
  commit_status: true
  fail_commit_status: false

  auto_review:
    enabled: false

  tools:
    ruff:
      enabled: true
    shellcheck:
      enabled: true
    markdownlint:
      enabled: true
    github-checks:
      enabled: true
      timeout_ms: 180000

chat:
  auto_reply: true
  create_issues: true

knowledge_base:
  web_search:
    enabled: true
```

Debugging Workflows

Start from the workflow run page:

- Read logs, expand failed steps, download logs if needed: Using workflow run logs
- Re-run a failed job/workflow after fixes: Re-running workflows and jobs

Get more debug information:

- Enable step + runner debug logs (`ACTIONS_STEP_DEBUG`, `ACTIONS_RUNNER_DEBUG`): Enable debug logging

References:

- GitHub Docs: Troubleshooting workflows
- Lost Pixel: How to debug GitHub Actions
- GitHub Community discussion: Actions debugging thread