GitLab CI/CD Pipelines in YAML

START HERE

`stages: [build, test, deploy]`
Order the pipeline

`rules: - if: $CI_COMMIT_BRANCH`
Decide when a job runs

`<job>: script: <cmd>`
Define a job

`artifacts: paths: [<dir>]`
Hand files to later jobs

`image: <image>`
Run in a container

`cache: paths: [<dir>]`
Reuse dependencies

`variables:`
Set CI/CD variables

`needs: [<job>]`
Skip the stage wait

CONCEPTS

Everything lives in .gitlab-ci.yml at the repo root; a push or merge request creates a pipeline of stages, each holding parallel jobs.

Pipeline one per push, MR or schedule
- **Stage** runs in order; next waits for all jobs
  - **Job** script run by a runner in a clean checkout

Jobs start clean: pass files with artifacts, speed up installs with cache (best effort only).

Validate edits with the pipeline editor's Lint tab before pushing.

JOBS

`<job>: script: - <cmd>`
Minimal job

`stage: <name>`
Put the job in a stage

`image: <name>:<tag>`
Container image for the job

`before_script: [<cmd>]`
Run before script

`after_script: [<cmd>]`
Run after, even on failure
Runs in a separate shell; changes to the environment from script are not visible.

`services: [postgres:16]`
Sidecar containers (databases)

`tags: [<runner-tag>]`
Choose runners by tag

`timeout: 30m`
Fail the job after this long

`retry: 2`
Retry failed jobs

`allow_failure: true`
Let the job fail without failing the pipeline

`when: manual`
Wait for a click to run

`interruptible: true`
Cancel when a newer pipeline starts

`resource_group: <name>`
Run one job at a time per group

`parallel: 5`
Run N copies

JOBS (cont.)

`coverage: '/Total: (\d+)%/'`
Extract coverage from the log

STAGES AND ORDER

`stages: [build, test, deploy]`
Set stage order

`needs: [<job>]`
Start once these jobs finish
needs: [] starts the job immediately, ignoring stages.

`needs: [{job: <job>, artifacts: false}]`
Depend without artifacts

`dependencies: [<job>]`
Fetch artifacts only from these jobs

`stage: .pre / .post`
Always first / last stage

`trigger: {include: <file>}`
Start a child pipeline

`trigger: {project: <group/proj>}`
Start a downstream project pipeline

RULES

`rules: - if: $CI_COMMIT_BRANCH == "main"`
Run when the expression is true

`rules: - if: $CI_PIPELINE_SOURCE == "merge_request_event"`
Run in merge request pipelines

`rules: - changes: ["src/**"]`
Run when these files changed

`rules: - exists: [Dockerfile]`
Run when a file exists

`rules: - when: never`
Fallback: skip the job
The first matching rule wins; with no match the job is not added.

`rules: - if: <expr> when: manual`
Manual only on a condition

`rules: - if: <expr> variables: {<K>: <v>}`
Set variables per rule

`workflow: rules: [...]`
Decide whether a pipeline exists

`workflow: auto_cancel: on_new_commit: interruptible`
Cancel superseded pipelines

ARTIFACTS AND CACHE

`artifacts: paths: [<path>]`
Keep files after the job

`artifacts: expire_in: 1 week`
Expire artifacts

`artifacts: when: always`
Upload even on failure

`artifacts: reports: junit: <file>`
Show test results in the MR

`cache: paths: [<dir>]`
Reuse files between runs

`cache: key: $CI_COMMIT_REF_SLUG`
One cache per branch

`cache: key: {files: [<lockfile>]}`
Key on lockfile contents

`cache: policy: pull`
Read-only cache

REUSE AND INCLUDE

`include: - local: <path>`
Include a file from this repo

`include: - project: <group/proj> file: <path>`
Include from another project

`include: - template: <name>.gitlab-ci.yml`
Use a GitLab template

`include: - remote: <url>`
Include by URL

`include: - component: <fqdn>/<path>@<ver>`
Use a CI/CD component

`<job>: extends: .<template>`
Inherit from a hidden job

`.<name>:`
Hidden job: a template, never runs

`script: - !reference [.<job>, script]`
Reuse part of another job

`default: {image: <img>}`
Defaults for all jobs

`parallel: matrix: - <VAR>: [a, b]`
One job per combination

`spec: inputs: <name>: {default: <v>}`
Typed inputs for included files

VARIABLES AND ENVIRONMENTS

`variables: <NAME>: <value>`
Pipeline or job variable

`$CI_COMMIT_BRANCH`
Branch name

`$CI_COMMIT_SHA`
Commit hash

`$CI_COMMIT_REF_SLUG`
URL-safe ref name

`$CI_PIPELINE_SOURCE`
Why the pipeline ran (push, merge_request_event, ...)

`$CI_DEFAULT_BRANCH`
The project's default branch

`$CI_PROJECT_DIR`
Checkout directory

`environment: {name: <env>, url: <url>}`
Track a deployment

Secrets go in Settings > CI/CD > Variables (masked, protected), not in the YAML.

RECIPES

Test and build

```
image: node:22
stages: [test, build]
test:
  stage: test
  script: [npm ci, npm test]
  cache:
    key: {files: [package-lock.json]}
    paths: [node_modules/]
build:
  stage: build
  script: npm run build
  artifacts: {paths: [dist/], expire_in: 1 week}
```

Merge request or main only

```
workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

Deploy on main with approval

```
deploy:
  stage: deploy
  environment: production
  script: ./deploy.sh
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
      when: manual
```

Matrix over versions

```
test:
  image: python:$PY
  parallel:
    matrix:
      - PY: ["3.11", "3.12", "3.13"]
  script: pytest
```