CI/CD

Continuous Integration

Aleix Llusà Serra

Master's degree in Machine Learning and Cybersecurity for Internet Connected Systems
Universitat Politècnica de Catalunya
https://epsem.upc.edu

9 de novembre de 2026



This work is licensed under a Creative Commons "Attribution-ShareAlike 4.0 International" license.

1 Continuous integration

2 References

Continuous integration

Definition

We keep our latest code ready to release. Is a practice, not a tool (CI server). (Shore i Warden 2008)

- Integration: merging all the team's code, testing, building, generating artifacts (such as docker images)...
- There is no integration post-phase at the end of the week nor when the full feature is done, it is automatically done at each change.
- The more often you integrate, the less painful it is. Less chance of merge conflicts.

Continuous integration

Local:

1. Get the latest code/Pull -r
2. Work
3. Run tests (must be a set of quick tests)
4. Check in/Push

Continuous integration

Automated build and testing in the CI server (triggered by commits):

1. Start from zero and get the latest code
2. Install dependencies (can be cached)
3. Build (if needed for testing)
4. Set up environment (microservices?) and testing data (hermetic testing)
5. Run tests: you need a suite of tests
6. (maybe) Multistage builds: e.g. one quick with basic testing and another slow with full testing
7. Build (if it was not needed for testing)
8. Test that the final build is ready for releasing/deploying (i.e. test installing the app or initialising the full service)
9. Reports: failing tests, coverage, style check...

Requirements from Agile

- Bugs are cheaper if you discover them earlier
- Never break the build: the main/master/trunk is the known-good
- Pull requests don't work well on teams using CI
- Integrate frequently every few hours (although partially done work, however needs to be stable and deployable)
- Integration machine
- Automate the build (such as Make): compile, run tests, configure settings, initialise DB schema, launch processes, run web servers

(Shore i Warden 2008)

Extra points

- When CI fails
  - Send an email to the developer with the report
  - Prevent going to CD
- Automatically apply tests for old tags/versions to discover where a new bug affects
- Reproducibility for older versions needed: virtualization?

Some relaxing reading

CARGO CULT

The Tool



"Continuous integration? What do you mean, continuous integration might help? We've already got a CI tool."

Your boss is on the phone with an Agile coach. It was supposed to be a phone screen, but you're not sure who's interviewing whom.

"No, they don't work together on stories. I run a tight ship here. I assign stories to resources and each story gets a feature branch. A few weeks later, the resource submits a pull request, I review it, and tell 'em what they got wrong."

You can't hear the other end of the conversation, but your boss's face is turning an interesting shade of red.

"Look, I asked you to help us with our design documentation, not spout some mumbo-jumbo about communist code ownership and continuous integration. I told you, we already have a CI tool. We've got real problems to solve. Merges have too many conflicts and nobody knows how to pick up Tiffani's work since she quit. I need my resources on the same page, and design documentation is how I'm gonna do it. Now, are you going to stop wasting my time and come help me with that, or…hello? Hello?"

Your boss flings their phone across the desk and turns to you. "I can't believe it! Another one hung up on me. What's wrong with these people?"

©Shore i Warden (2008)

More reading: branching is easy, merging is harder

`https://martinfowler.com/articles/branching-patterns.html`

CI automation

Laboratory exercises

`https://docs.gitlab.com/ee/topics/build_your_application.html`

Gitlab CI: stages and jobs

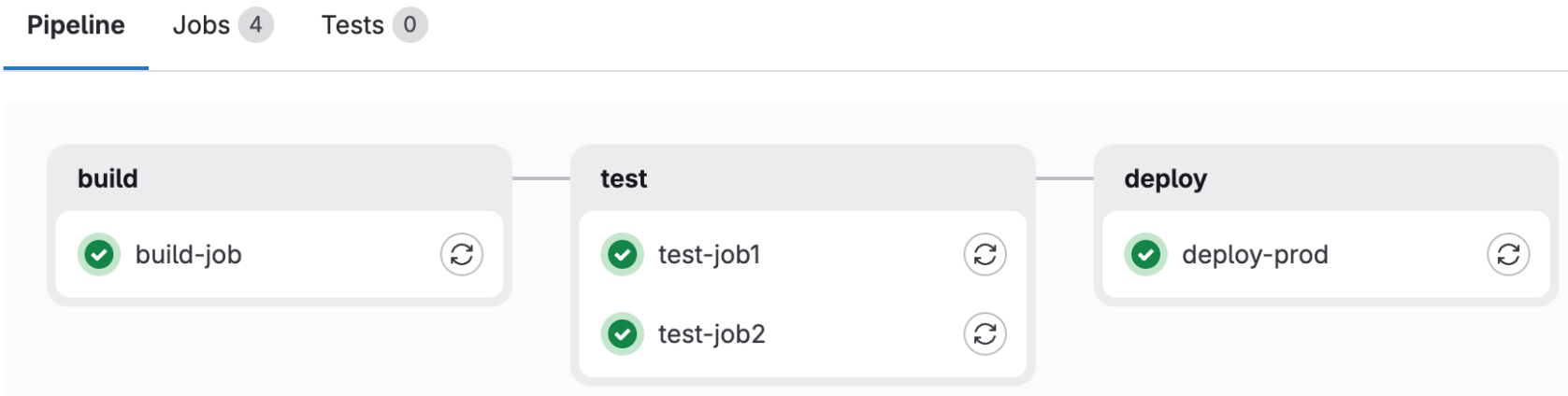
.gitlab-ci.yml

```
stages:
- test
- deploy

build:
  stage: test
  script:
    - echo "This job tests something"
    - sleep 20

production:
  stage: deploy
  script:
    - echo "This job deploys the $CI_COMMIT_BRANCH."
```

Gitlab CI: stages and jobs



©Gitlab.com

Gitlab CI: build and test

.gitlab-ci.yml

```
stages:
- all

build-test:
  stage: all
  image: python:3.15  # docker pull python:3.15
  script:  # workingdir on the project
    - pip install cookiecutter
    - "echo \"default_context:\n  app_name: myapp\" > config.yaml"
    - cookiecutter https://github.com/cookiecutter-flask/
cookiecutter-flask.git --no-input --config-file config.yaml
    - pip install -r myapp/requirements/dev.txt
    - pytest myapp
    - flake8 myapp
```

Gitlab CI: build and test stages sharing artifacts

.gitlab-ci.yml

```
stages:
- build
- test
cache:
  paths:
    - venv/
default:
  image: python:3.11
build-python:
  stage: build
  script:
    - python -m venv venv
    - source venv/bin/activate
    - pip install cookiecutter
    - "echo \"default_context:\n  app_name: myapp\" > config.yaml"
    - cookiecutter https://github.com/cookiecutter-flask/cookiecutter-flask.git --no-input --config-file c
    - pip install -r myapp/requirements/dev.txt
  artifacts:
    paths:
      - "myapp/"
test-pytest:
  stage: test
  script:
    - source venv/bin/activate
    - pytest myapp
test-style:
  stage: test
  script:
    - source venv/bin/activate
    - flake8 myapp
```

Gitlab CI: reports

.gitlab-ci.yml

```
test-style:
  stage: test
  script:
    - source venv/bin/activate
    - pip install flake8-gl-codeclimate
    - flake8 --format gl-codeclimate
--output-file quality.json myapp
  artifacts:
    reports:
      codequality: quality.json
```

https://docs.gitlab.com/ee/ci/yaml/artifacts_reports.html

Gitlab CI: Docker

compose.yml

```
services:
  web:
    image: web:${TAG:-latest}
    build:
      context: web
```

.gitlab-ci.yml

```
docker-build:
  image: docker:cli
  stage: build
  services:
    - docker:dind
  variables:
     TAG: $CI_COMMIT_REF_SLUG
  script:
    - docker compose build
```

Gitlab CI: Docker image push

.gitlab-ci.yml

```
docker-build:
  image: docker:cli
  stage: build
  services:
    - docker:dind
  variables:
     TAG: $CI_PIPELINE_IID
  script:
    - docker compose build
    - echo "$CI_REGISTRY_PASSWORD" | docker login
-u $CI_REGISTRY_USER --password-stdin $CI_REGISTRY
    - docker tag "web:${TAG}" "$CI_REGISTRY_IMAGE/web:${TAG}"
    - docker tag "web:${TAG}" "$CI_REGISTRY_IMAGE/web:latest"
    - docker push $CI_REGISTRY_IMAGE/web:${TAG}
    - docker push $CI_REGISTRY_IMAGE/web:latest
```

Gitlab CI: Docker testing

.gitlab-ci.yml

```
docker-build:
  image: docker:cli
  stage: build
  services:
    - docker:dind
  variables:
     TAG: $CI_PIPELINE_IID
  script:
    - docker compose up --detach
    - docker compose exec -T web curl http://localhost:80
  after_script:
    - docker compose down || true
```

Building and Publishing

https://packaging.python.org/en/latest/tutorials/packaging-projects/

https://packaging.python.org/en/latest/guides/section-build-and-publish/

References



Shore, James i Shane Warden (2008). The Art of Agile Development. O'Reilly. ISBN: 9780596527679. URL: https://discovery.upc.edu/permalink/34CSUC_UPC/8e3cvp/alma991005248477306711.