A Gentle Introduction to Container-based CI for Coq projects

Érik Martin-Dorel

ACADIE team / IRIT lab. / Univ. Toulouse III - Paul Sabatier / France

erik.martin-dorel@irit.fr

Coq Workshop 2020

2020-07-05

Goals of this talk

- CI tutorial: review the advantages of using docker-coq in your project
- Self-contained talk: recap the main notions involved in this infrastructure
- For advanced users: present the recently developed tool docker-keeper to help maintain your own images
- Gather feedback: post questions/suggestions in the Zulip thread

Continuous Integration (CI)

or how to integrate code frequently to reduce regressions and the cost of software bugs

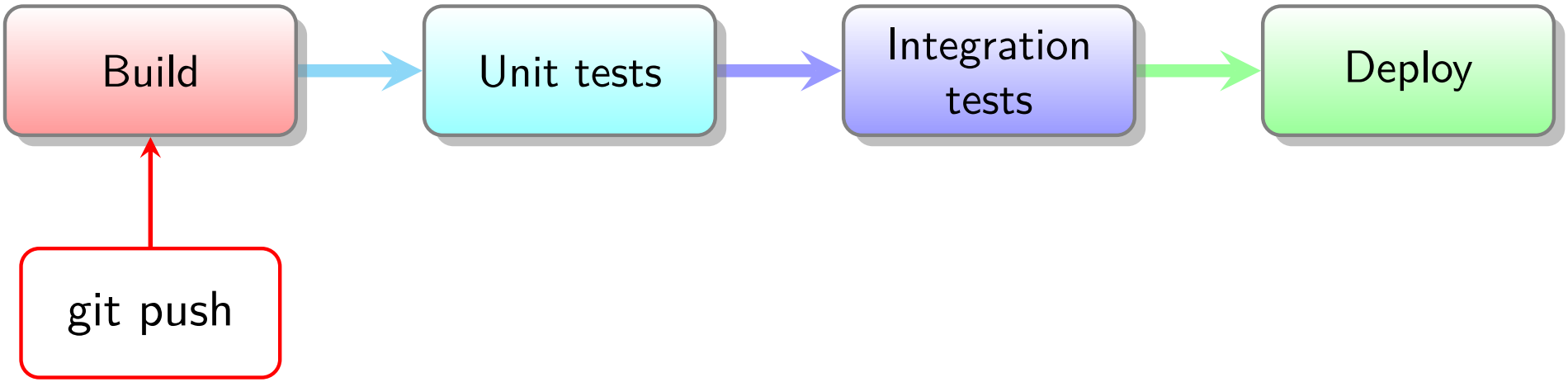
1. "maintain a single source repository" using version control;
2. "automate the build" using dedicated tools;
3. include automated tests in the build process;
4. integrate developed code in the main branch frequently;
5. ensure the main branch always builds, using a dedicated machine;
6. "fix broken builds immediately";
7. "keep the build fast" (if need be, using so-called pipelines);
8. "test in a clone of the production environment";
9. facilitate the access to the (latest) build artifacts;
10. ensure the build status is clearly visible;
11. "automate deployment" into production.

(cf. https://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration)

Continuous integration and continuous deployment (CI/CD)

Notion of pipeline:

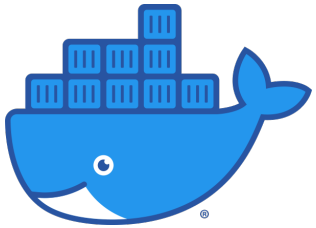
- triggered for each "push"
- each stage starts only if previous stages succeeded



Container-based CI

- CI infrastructure $\rightsquigarrow$ relies on a continuous integration platform, e.g. GitHub Actions, GitLab CI, CircleCI, Travis CI...
- Perform build/tests in an isolated way $\rightsquigarrow$ virtualization techniques: virtual machines or containers
- Docker containers: more lightweight (disk, memory, CPU) than a VM: directly uses Linux kernel capabilities; need not emulate hardware/OS

Why using docker-coq?

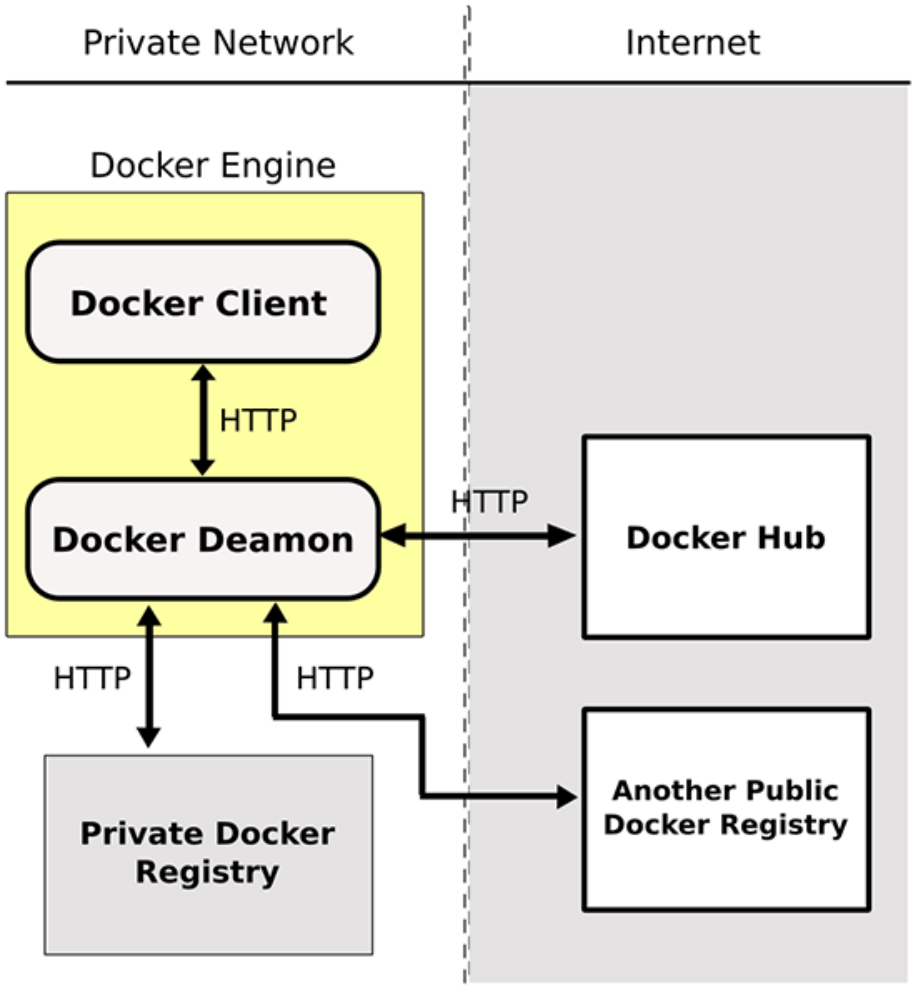

&

?

Two possible use cases:

1. Setup CI & Use a prebuilt Coq (+ selected libs) to speed-up the build
   cf. https://github.com/coq-community/docker-coq/wiki/CI-setup
2. Locally perform tests and builds with a specific Coq version

Overview of the Docker architecture



Source: https://theodorosploumis.github.io/docker-java/ (CC BY 4.0)

(see also https://docs.docker.com/get-started/overview/#docker-architecture)

Overview of Docker commands (useful for "use case #2")

- `docker build` : (Dockerfile, context) $\mapsto$ image
- `docker run` : image $\mapsto$ running container

- `docker push` : image $\rightarrow$ Docker registry
- `docker pull` : image $\leftarrow$ Docker registry

For details: https://github.com/coq-community/docker-coq/wiki/CLI-usage

Some Docker conventions and requirements

Images naming convention: *repository* : *tag*, where:

- *repository* =
  - `some-registry.com/user/repo` — specify a custom registry
  - `user/repo` — use the Docker Hub registry
  - `repo` (no slash) — use an official Docker Hub distribution
- *tag* = `latest` if omitted; denotes a stable version

Some Docker conventions and requirements

Images naming convention: *repository* : *tag*, where:

- *repository* =
  - `some-registry.com/user/repo` — specify a custom registry
  - `user/repo` — use the Docker Hub registry
  - `repo` (no slash) — use an official Docker Hub distribution
- *tag* = `latest` if omitted; denotes a stable version

Requirements:

- Installing Docker requires a 64-bit architecture
  cf. https://docs.docker.com/engine/install/#supported-platforms
- Given that the Docker daemon socket is owned by `root`, under Linux one needs to prepend all docker commands by `sudo`
  (or add in your ∼/`.bashrc`: `alias docker="sudo docker"`)

Brief history of docker-coq

- Needs: build on Travis CI (within the 50' time limit) the validsdp library (depending on Coq + nine libraries)
- 1st August 2018: start experiments with Docker Hub and document it
- 13 September 2018: contact the Coq team (issue 8474) to suggest distributing images of Coq for both stable and `dev` versions

  [...]

Brief history of docker-coq

- Needs: build on Travis CI (within the 50' time limit) the validsdp library (depending on Coq + nine libraries)
- 1st August 2018: start experiments with Docker Hub and document it
- 13 September 2018: contact the Coq team (issue 8474) to suggest distributing images of Coq for both stable and `dev` versions

  [...]
- 7 November 2018: announce (coq-club) the availability of the images. Infrastructure at that time:
  - two-switches images coqorg/coq, built by Docker Hub (1 worker)
  - Dockerfiles in github.com/coq-community/docker-coq (multi-branches)

Brief history of docker-coq

- Needs: build on Travis CI (within the 50' time limit) the validsdp library (depending on Coq + nine libraries)
- 1st August 2018: start experiments with Docker Hub and document it
- 13 September 2018: contact the Coq team (issue 8474) to suggest distributing images of Coq for both stable and `dev` versions

  [...]
- 7 November 2018: announce (coq-club) the availability of the images. Infrastructure at that time:
  - two-switches images coqorg/coq, built by Docker Hub (1 worker)
  - Dockerfiles in github.com/coq-community/docker-coq (multi-branches)
- 1st July 2020: new infrastructure:
  - Images built using a GitLab CI pipeline driven by a YAML specification (single-branch repository)
  - On-going migration to single-switch images (with a transition time)

Contents of the docker-coq images

- based on the Debian 10 Slim distribution
- contain 16 essential APT packages + opam
- coq as default user with UID = GID = 1000, with sudo permissions.
- list of provided tags: https://hub.docker.com/r/coqorg/coq#supported-tags

Contents of the docker-coq images

- based on the Debian 10 Slim distribution
- contain 16 essential APT packages + opam
- coq as default user with UID = GID = 1000, with sudo permissions.
- list of provided tags: https://hub.docker.com/r/coqorg/coq#supported-tags

- images for $8.4 \leq$ coq ≤ 8.6 only available with ocaml 4.02.3
- images for coq ≥ 8.7 available with ocaml 4.05.0 and 4.07.1+flambda
- images for coq ≥ 8.8 *also* available with ocaml 4.09.1+flambda
- all images contain `opam-depext`, `ocamlfind`, `dune`, `num`, and `coq`
- images for coq ≥ 8.6 *also* contain `coq-bignums`

Contents of the docker-coq images

- based on the Debian 10 Slim distribution
- contain 16 essential APT packages + opam
- coq as default user with UID = GID = 1000, with sudo permissions.
- list of provided tags: https://hub.docker.com/r/coqorg/coq#supported-tags

- images for $8.4 \leq$ coq ≤ 8.6 only available with ocaml 4.02.3
- images for coq ≥ 8.7 available with ocaml 4.05.0 and 4.07.1+flambda
- images for coq ≥ 8.8 *also* available with ocaml 4.09.1+flambda
- all images contain `opam-depext`, `ocamlfind`, `dune`, `num`, and `coq`
- images for coq ≥ 8.6 *also* contain `coq-bignums`

- for math-comp users: see also https://hub.docker.com/r/mathcomp/mathcomp

About the coq-community templates

- Project URL: github.com/coq-community/templates
- Gathers ready-to-use templates for Travis CI, CircleCI, GitHub Actions (using both docker-coq and a Nix setup for `coq.dev`, cf. github.com/coq-community/manifesto/wiki for a comparison)
- These templates are currently provided with a script (`generate.sh`), reading a dedicated metadata file (`meta.yml`)

Some interesting features of GitHub Actions

- naturally well-integrated in GitHub
- provides a so-called Problem Matchers feature, useful in Pull Requests (see `PR erikmd/docker-coq-github-action-demo#5, src/demo.v, line 14`)
- it offers a large number of concurrent jobs
- the configuration file (see next slide) is much more concise than, e.g., a dockerized Travis CI configuration

Minimal working example of docker-coq-action config

```yaml
name: CI
on:
  push:
    branches: ['master'] # forall push/merge in master
  pull_request:
    branches: ['**']  # forall submitted Pull Requests
jobs:
  coq:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        coq_version:
          - '8.11'
          - 'dev'
        ocaml_version: ['4.07-flambda']
      fail-fast: false  # don't stop jobs if one fails
    steps:
      - uses: actions/checkout@v2
      - uses: coq-community/docker-coq-action@v1
        with:
          opam_file: 'folder/coq-proj.opam'
          coq_version: ${{ matrix.coq_version }}
          ocaml_version: ${{ matrix.ocaml_version }}
```

```yaml
name: CI
on:
  push:
    branches: ['master'] # forall push/merge in master
  pull_request:
    branches: ['**']  # forall submitted Pull Requests
jobs:
  mathcomp:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        image:
          - 'mathcomp/mathcomp:1.10.0-coq-8.10'
          - 'mathcomp/mathcomp:1.10.0-coq-8.11'
          - 'mathcomp/mathcomp:1.11.0-coq-dev'
          - 'mathcomp/mathcomp-dev:coq-dev'
      fail-fast: false  # don't stop jobs if one fails
    steps:
      - uses: actions/checkout@v2
      - uses: coq-community/docker-coq-action@v1
        with:
          opam_file: 'folder/coq-proj.opam'
          custom_image: ${{ matrix.image }}
```

Under the hood: initial infra with docker-hub-helper

To help maintain multi-branches, automated-build repos on Docker Hub:

- first developed a dedicated Python3 tool: github.com/erikmd/docker-hub-helper (~ 500 loc)

Under the hood: initial infra with docker-hub-helper

To help maintain multi-branches, automated-build repos on Docker Hub:

- first developed a dedicated Python3 tool:
  github.com/erikmd/docker-hub-helper (~ 500 loc)

However, the architecture did not scale:

- for the Dockerfile repos: many branches to handle and force-push,
  incompatible with pull requests worfklow;
- for the images repos: manual configuration of the automated build,
  needed many clicks in the Docker Hub UI;
  only 1 worker $\rightsquigarrow$ took more than 7 hours to build 6 images for
  `mathcomp/mathcomp:1.11.0-coq-*`

Under the hood: new infra using docker-keeper

6 objectives:

1. scalability of the build (cf. previous slide)
2. maintainability of the spec (single YAML file in `master` branch)
3. genericity (being applicable for your own Coq-related (or not) project)
4. consistency of the deployed images (detect duplicates in specified tags, check tags to remove) and of the underlying Dockerfiles (linter)
5. better transparency (public GitLab CI logs)
6. better documentation (add more metadata & generate readme)

About docker-keeper-template

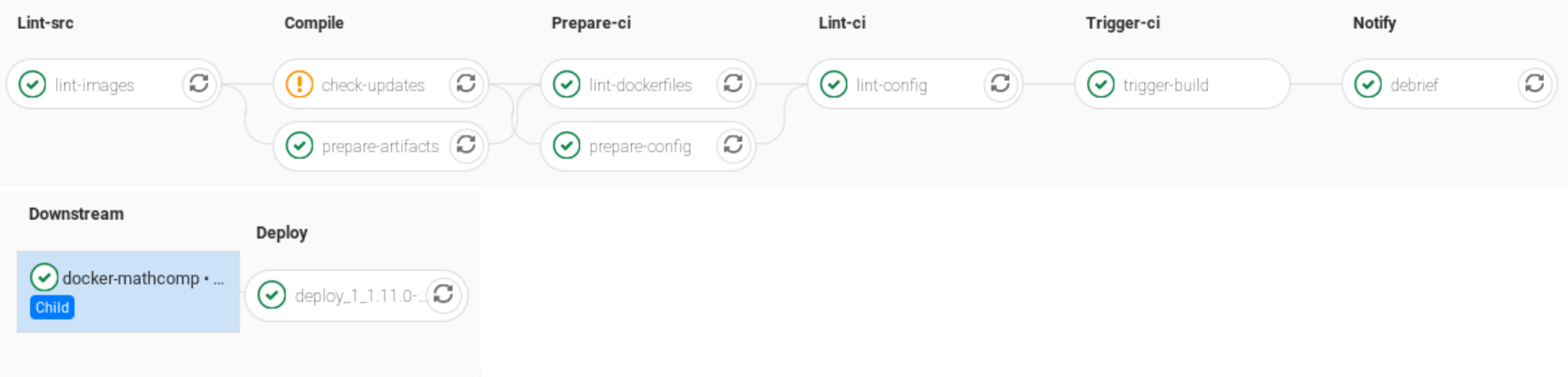
To configure your own repository of Dockerfiles and easily build/deploy them to the Docker Hub registry:

- Fork gitlab.com/erikmd/docker-keeper-template
- Follow the README.md instructions in the repo
  – *in particular, not exhaustively:*
  - Add protected/masked variables `HUB_USER` and `HUB_TOKEN`
  - Add a scheduled pipeline with variable `CRON_MODE := nightly`
  - Edit the Dockerfile(s)
  - Edit the YAML specification (e.g., putting `active:  true`)
  - Edit the README.md (containing the placeholder `<!-- tags -->`)
  - If need be, update the docker-keeper subtree:

```
git subtree pull --squash -P external/docker-keeper \
  https://gitlab.com/erikmd/docker-keeper.git master
```

  - Git push!

Example of generated GitLab CI pipeline (source)



Perspectives

- Automate docker-keeper's last 2 steps:
  see feature request docker/roadmap#115 !
- Enhance/extend the coq-community templates
  e.g., complement the GitHub Action template with a Nix config?
  (cf. Théo Zimmermann's and Cyril Cohen's works)
- Distribute images with ProofGeneral inside?
  (docker-coq being used in PG integration tests for now)
- Other ideas/questions?