A Gentle Introduction to Container-based CI for Coq projects (Coq-2020 extended abstract)

Érik Martin-Dorel

IRIT, Univ. Toulouse III – Paul Sabatier, Toulouse, France
erik.martin-dorel@irit.fr

Abstract

Continuous integration is a key notion in software development, as well as for developing formal proofs. In this talk, we briefly review the architecture of the components we developed or used to this aim, while focusing on the Coq ecosystem, with the ultimate goal to provide a tutorial for Coq users that may want to use one such approach for their own projects, and gather questions or suggestions regarding the proposed infrastructure.

1 CI/CD for V&V: Introduction and Motivation

Continuous Integration (CI) denotes a set of practices that aims at developing and integrating software artifacts in an “incremental” way. The “CI” term has first been proposed by Grady Booch [Boo94, p. 256] in the scope of object-oriented software development. This notion also integrates well in the scope of Verification and Validation (V & V), e.g., relying on unit tests and integration tests that are systematically run after building the artifact.[1] Since then, CI has been widely promoted and used in software development teams, as it significantly mitigates typical integration issues, reducing the risk of regressions and the cost of software bugs that can thus be found and fixed earlier. In particular, Extreme programming [Bec99] and subsequent agile methodologies strongly rely on the notion of CI, with the overall idea to “harness change” and “deliver working software frequently”. As mentioned by Martin Fowler [Fow06], CI itself builds upon a dozen of practices. The last one (“automate deployment”) actually refers to the so-called “CD” acronym, which denotes two slightly different approaches: (*i*) Continuous Delivery: if the build and test stages succeed, the artifact can be deployed on-demand (typically after a manual review and approval); (*ii*) Continuous Deployment: if the build and test stages succeed, the artifact is automatically deployed into production.

In the scope of Coq proofs development, CI pipelines can just consist in compiling the project (build stage), testing benchmarks or reverse dependencies[2] (test stage), and automatically uploading artifacts such as online documentation (continuous deployment stage). Beyond the main development branch that is intended to be tested *and deployed*, CI pipelines can just as well be triggered for feature branches and “pull requests”, in order to provide further insight when reviewing a contribution. For instance, the continuous integration platform can be configured to warn that a given change breaks the compatibility with a supported version of Coq.

2 From OS-level Virtualization to Container-based CI

An effective CI infrastructure relies on a dedicated server (the Continuous Integration platform, that can be on-premise or hosted by a cloud provider). When the pipeline is triggered, the server is in charge of fetching the code at stake and building it, ideally in an isolated way. This isolation can be done by virtualization techniques, using hypervisors (virtual machines) or OS-level virtualization (Docker[3] containers). Beyond the fact that a container is much more lightweight than a virtual machine in terms of disk, memory, and CPU resources, its use in a CI/CD context is particularly interesting because it enhances reproducibility (the container is itself an artifact can be built, tested (CI) and deployed (CD) as is) as well as portability (e.g., a given container based on a Debian distribution, could be built on an Ubuntu-based CI server, downloaded then run under Windows 10).

3 A panorama of the CI Tooling Available for Coq Projects

The development of the `docker-coq` tooling started from the need to build on Travis CI (within the 50’ time limit) the `validsdp` library, which depends besides Coq, on nine OCaml or Coq libraries. It resulted in the creation of three Docker Hub repositories (`coqorg/coq`, `mathcomp/mathcomp` and `mathcomp/mathcomp-dev`) that gather pre-built Docker images[4] of

[1] “The principle of continuous integration applies as well to testing, which should also be a continuous activity during the development process” [Boo94, p. 273].
[2] Similarly to the strategy that has been applied for the CI infrastructure of Coq itself [Zim19, Chap. 3.6].
[3] Released as free software since March 2013.
[4] Roughly speaking, a Docker *image* is a snapshot of a (running) container.

Coq and of the Mathematical Components library, using the `opam` package manager. Also, a dedicated tool `docker-hub-helper` was written to facilitate the maintenance of multi-branches, "automated build repositories" on Docker Hub, even if this architecture appears not to scale for large repositories, unlike some CD pipeline on, say, GitLab CI that deploys to Docker Hub.

Independently of `docker-coq`, Théo Zimmermann developed a CI infrastructure based on the `Nix` package manager [Zim19, Chap. 7]. Both approaches are being used for the CI of the `coq-community` GitHub projects, and the respective advantages of the two approaches are outlined in the `coq-community/manifesto` wiki. Some `Nix` support for the `math-comp` library is also developed by Cyril Cohen.

The `coq-community` organization gathers `templates` for CI configuration files that resulted from experiments conducted with several platforms: Travis CI, CircleCI, and GitHub Actions—which has a couple of advantages over the other solutions: it is naturally well-integrated in GitHub, it provides a so-called Problem Matchers feature, useful in Pull Requests[5], it offers a large number of concurrent jobs, and the configuration file is very concise,[6] cf. Figure 1 which shows two self-contained configurations for a Coq, `opam`-based project. Regarding GitLab CI (which natively supports Docker), templates are also available[7] and the `coqbot` developed by Théo Zimmermann can then be used to complement GitLab's mirroring feature of GitHub repositories.

As possible extensions of this work, it may be interesting to enhance the aforementioned templates (e.g., by providing Nix support to the GitHub Actions template), simplify the generation and use of their `meta.yml` "pivot" file (regarding specification and propagation facilities), to ultimately have a smooth update process for projects that rely on these templates. Also, the required infrastructure for building `mathcomp/mathcomp` stable images could be adapted and generalized, to help Coq users with maintaining their own registry of pre-built Docker images.

References

[Bec99] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, October 1999. `doi:10.1109/2.796139`.

[Boo94] Grady Booch. *Object-oriented Analysis and Design with Applications*. Benjamin/Cummings Publishing Company, 1994.

[Fow06] Martin Fowler. Continuous Integration. URL: `https://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration`, May 2006.

[Zim19] Théo Zimmermann. *Challenges in the collaborative evolution of a proof language and its ecosystem.* PhD thesis, Paris Diderot University, France, 2019. URL: `https://tel.archives-ouvertes.fr/tel-02451322`.

Figure 1: Minimal working example of `docker-coq-action` (file `.github/workflows/coq-action.yml`) [v1 syntax, as of 2020-06-18]. Left: use default `docker-coq` images. Right: use custom (larger) images.

```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 }}
```

[5] See `https://github.com/erikmd/docker-coq-github-action-demo/pull/5/files`, file `src/demo.v`, line 14.
[6] To be compared with, e.g., the Travis CI Docker configuration file which is much more cluttered.
[7] See `https://github.com/coq-community/templates/issues/25`