HAL
open science

OPAM-builder: Continuous Monitoring of OPAM Repositories

Fabrice Le Fessant

To cite this version:

Fabrice Le Fessant. OPAM-builder: Continuous Monitoring of OPAM Repositories. OCaml Users and Developers Workshop 2016, Sep 2016, Nara, Japan. hal-01352008

HAL Id: hal-01352008

https://inria.hal.science/hal-01352008v1

Submitted on 5 Aug 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

OPAM-builder: Continuous Monitoring of OPAM Repositories

Fabrice Le Fessant
INRIA & OCamlPro
fabrice.le_fessant@inria.fr

Abstract

In this talk, we will present the opam-builder system. opam-builder is an online service that monitors the official OPAM repository, continuously builds all versions of all packages for 6 different versions of OCaml, and display html reports with all the information about failed installations. It can be used both by repository maintainers and package developers as a useful tool to improve the quality of the OPAM repository.

1 Introduction

The OPAM package manager has become the official package manager of the OCaml community. As a consequence, ensuring that as many packages as possible can be installed at any given time has become a critical task for repository maintainers, but also a very complex task, as the installability of a package depends both on the installability of all its transitive dependencies, and on the ability to build the package itself.

Most developers submitting packages only test their package in a limited environment, usually with one or two versions of OCaml, on a given state of the repository. They then submit their package for insertion in the official repository, through a pull-request on Github. The pull-request is then tested by two continuous integration tools:

- On Travis, installation of the modified packages are tested on 5 different versions of OCaml, 2 versions of OPAM and 2 different systems (Linux and OSX);
- The Camelus bot checks that modified package descriptions comply with a set of linting rules;

Although these tests are very useful, they only provide limited information on the impact of a pull-request: in particular, they will not detect that a package cannot be installed anymore, if its own description was not modified in the pull-request. As a consequence, the quality of the repository is good for new packages, but gets worse for existing packages.

In 2015, we presented a website, the OPAM Weather Service [1], that monitors the OPAM repository, computes dependency uninstallability and displays its results online (http://ows.irill.org/). This service is quite useful to detect packages with broken dependencies (dependencies that are missing or conflicting), usually resulting from modifications of the ranges of accepted versions in OPAM files.

However, the OPAM Weather Service does not provide any hint of the build installability of a package, i.e. if a package can be compiled with a given version of OCaml (that might not have been available at the time it was submitted) and with the new versions of its dependencies (whose interfaces might have changed in incompatible ways).

To solve this issue, we designed and deployed a new service, opam-builder, that tracks modifications of the OPAM repository, and tries to rebuild all impacted package versions for all the major versions of OCaml. Results are displayed on http://opam.ocamlpro.com/builder/.

2 Overview

After a few weeks of development, opam-builder was officially announced on March 23, 2016, and its code released under AGPLv3 on https://github.com/OCamlPro/opam-builder

Since then, it has been monitoring the OPAM repository and displays accurate information for 6 major versions of OCaml, within hours after a commit in the OPAM repository. It is running on a 4-core i5-3570S CPU at 3.10GHz with 8 GB of memory and 2TB of SATA disks.

On May 3, 2016, the last report was showing:

- 1197 packages and 4985 different versions
- 15950 successful builds
- 3350 broken packages (broken dependencies, either missing or conflicting)
- 3228 failed builds

Space usage, including the GIT opam-repository, .opam folder, cached binary archives, and build logs, after 6 weeks of activity without any cleaning, was:

- Version 3.12.1: 14 GB
- Version 4.00.1: 25 GB
- Version 4.01.0: 63 GB
- Version 4.02.1: 74 GB
- Version 4.03.0: 10 GB (released after 5 weeks of running)

As a side effect of opam-builder, we also released an opam-file website (http://opam.ocamlpro.com/builder/html/api.4.02.1/) that displays type information extracted from installed .cmi files. This website can be used to lookup fully-qualified identifiers and see in which packages and which versions of these packages they are available.

3 Architecture and Scalability

opam-builder is running on single computer, but with multiple processes:

- For each version of OCaml, a directory contains a GIT clone of the opam-repository, and OPAM installation (.opam) and a cached of previous builds. An opam-builder process monitors the GIT repository, detects new commits, computes the impacted packages and triggers OPAM installations for all impacted packages and versions, in empty OPAM switches. At the end, the results for the new commit are exported as one single file in an export directory.

- A single opam-builder process monitors all the export directories of other opam-builder processes, and, when a new file appears, recomputes the HTML pages to be displayed.

Given the number of packages and versions in the opam-repository, building all versions of all packages for 6 versions of OCaml to display complete information after every commit cannot work without some optimisations, that we have applied for opam-builder:

- **Package Filtering:** To avoid rebuilding all packages after a commit, we compute the set of package versions that have been impacted by the modifications, and only rebuild those packages. For that, we compute a hash of meta-data associated with the transitive dependencies of a package, that we can easily compare with the previous hash. Only packages whoses hashes have changed are further examined.

- **Direct Solver Access:** Calling opam to compute the *install set* of a package versions is particularly slow, as opam has to load the complete universe, translate it to CUDF, call the solver, and then translate back the results. In opam-builder, we only use opam for the first package version, we then re-use the computes CUDF[**?**] universe to directly query the aspcud solver for all other impacted packages and versions.

- **Version Filtering:** Once we have the install set of every impacted version, we filter again by computing a hash for every install set. Then, we only try to install versions whose install set has changed, showing that either the build instructions have changed, the sources, or the dependencies.

- **Build Caching:** Installing an OPAM package in an empty switch requires to install first all its install set. Doing so for many packages would result in installing the same package tens of times (for example, ocamlfind is a required by hundreds of packages). To avoid this issue, we modified opam to cache the binary result of a package installation, so that we can just reuse previously compiled packages. Since build and install instructions are not always clearly separated, we actually generate a binary installation archive for both compilation steps. Also, opam is configured to work sequentially, to avoid parallel building instructions.

- **Fast cleaning:** For every package that we need to build, we have to start from an empty switch. Initially, this was done by completely removing the switch, and then extracting a copy of the empty switch (with just the OCaml compiler) on it. Given the size of the OCaml distribution, this step was actually quite slow, so we decided to use a different method: after testing a package installation, we snapshot the switch and compare that snapshot with the snapshot of the empty switch. We then remove all the files that were added, unless a file was actually modified, in which case we backtrack to the initial cleaning method.

Thanks to all these optimisations, opam-builder is now able to respond in almost real-time to commits in the opam-repository. More than 6-hour delays were actually observed only because the same server was used for other experiments, that would eat up all the server memory, leading to out-of-memory crashes of opam-builder.

4 Extensions

This work could be extended in many directions:

- Stable OPAM distributions: solvers such as aspcud are known to run very slowly on repositories with many versions per package, which is the direction in which the current opam-repository is going. A solution would be to extract smaller snapshots from the repository, containing only a few versions per package, and that are known to build and install correctly on at least one system. Such a task could be completely automated using opam-builder.

- Binary distributions, documentation and opam-files: binary archives generated by opam-builder could be used to provide many services to the community: they could be used to provide a binary cache, speeding up installation of basic packages, to generate documentation of all installed packages, and finally, the opam-file service could be turned into a dynamic querying service, instead of the current static site seldomly updated.

References

[1] P. Abate, R. Di Cosmo, L. Gesbert, F. Le Fessant, R. Treinen, and S. Zacchiroli. Mining component repositories for installability issues. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 24–33, Piscataway, NJ, USA, 2015. IEEE Press.