The OCaml Platform v1.0

Anil Madhavapeddy (speaker), Amir Chaudhry, Jeremie Diminio,
Thomas Gazagnaire, Louis Gesbert, Thomas Leonard, David Sheets,
Mark Shinwell, Leo White and Jeremy Yallop

The OCaml Platform combines the OCaml compiler toolchain with a coherent set of tools for build, documentation, testing and IDE integration. The project is a collaborative effort across the OCaml community, tied together by the OCaml Labs group in Cambridge and with other major contributors listed above. The requirements of the Platform are being guided by the industrial OCaml Consortium (primarily Jane Street, Citrix and Lexifi).

This talk follows up the OCaml 2013 talk that introduced the Platform. Since then, many tools have been released in parallel via the OPAM package manager, and this year's talk will demonstrate the concrete workflow that ties them together (see Figure 2). We will first recap the Platform ethos briefly, update on the OPAM package manager v1.2, and conclude with the Platform workflow.

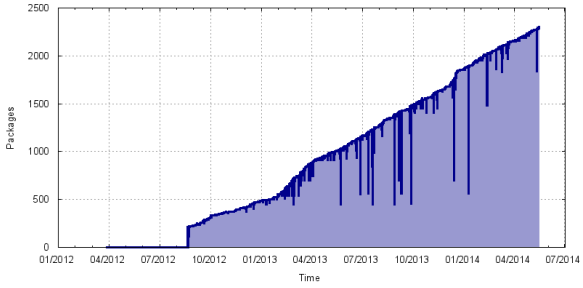

Figure 1: Growth in OPAM packages in the central repository. Note that this plots all unique versions, since OPAM supports installing older revisions of a given package.

1 Background

We have initially taken direction from major industrial users because these groups have a great deal of experience of using the language at scale. For the Platform to be considered successful, it has to be a viable product for those heavy users of OCaml. However, each of the users also have large codebases with distinct coding styles, and often have built their own extensive libraries to complement the OCaml standard library and open-source ecosystem.

A defining characteristic of OCaml programming is modularity, where one component can be swapped out for another component with the same interface. The Platform follows the same philosophy; it does not mandate any given set of libraries, but instead focusses on providing the tooling to assemble, build, test and document a set of packages suitable to a particular problem domain. Example domains include the Core standard library from Jane Street, the Ocsigen client and server web framework that compiles OCaml into JavaScript, and the Mirage unikernel operating system.

2 OPAM

OPAM plays a key role in the tooling for the Platform, by providing a frontend that can control a concurrently installed set of compiler versions and package sets. Since its public release, thousands of packages and revisions have been added to the central repository, and growth continues strongly in 2014 (see Figure 1). An important feature of OPAM is that it tracks multiple revisions of a single package, thereby letting packages rely on older interfaces if they need to. It also supports multiple package repositories, letting users blend the global stable package set with their internal revisions, or building completely isolated package universes.

The OPAM 1.2 release has been adapted to make it easier to package up on different operating systems, and features better support for external solvers to deal with the growth of the package database. Most importantly, much of its logic is available as an OCaml library, making it easy for other tools to interface directly with OPAM.

One major use of OPAM as a library is that we now have the capability to run tests over the complete package set to understand the ramifications of proposed compiler changes and any breaking effects they might have. This substantially improves the usage data available to anyone working on compiler modifications, and has been used extensively during the development of OCaml 4.02.

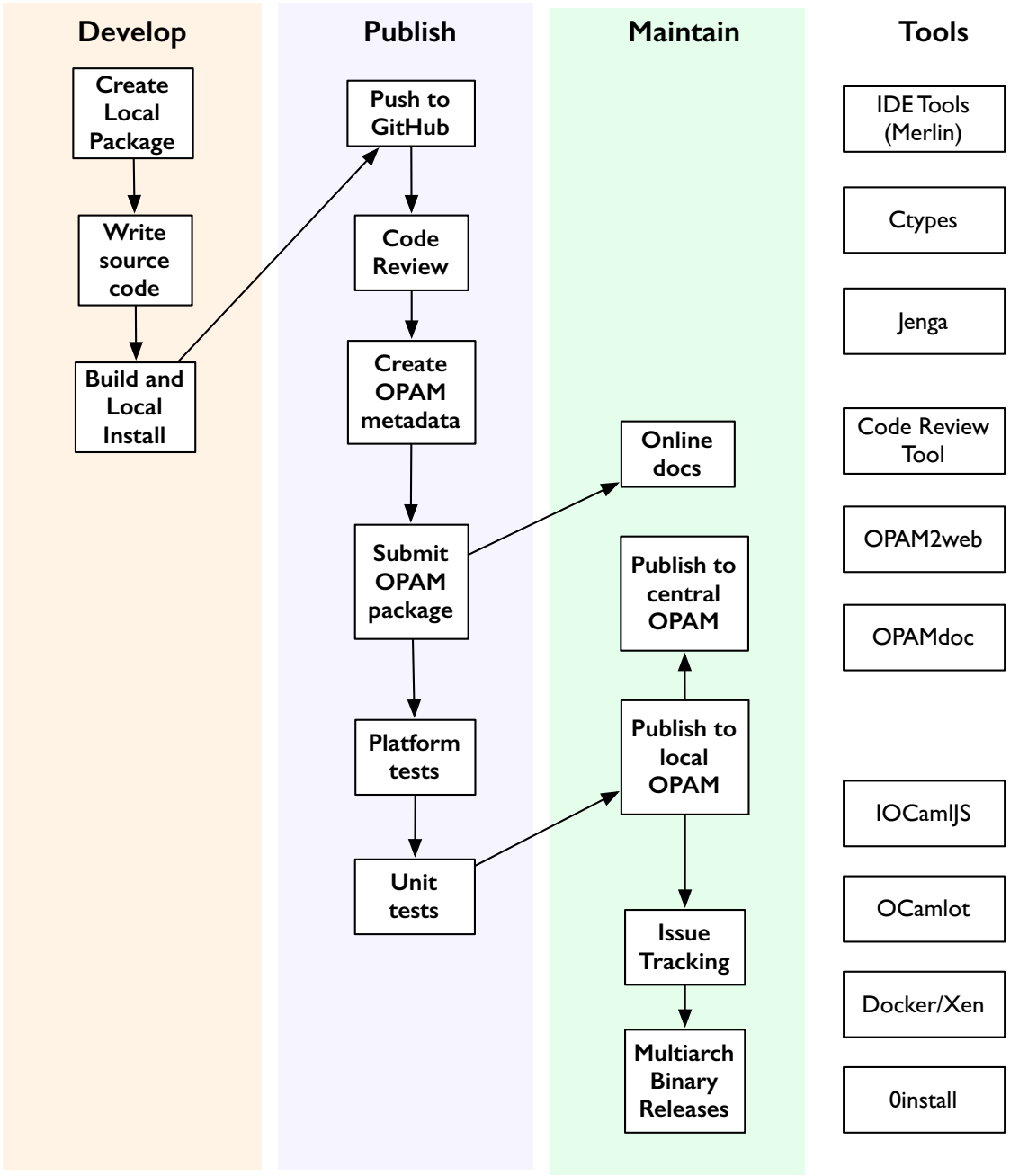

Figure 2: Workflow of building, publishing and maintaining a package using the OCaml Platform tools. On the right are the tools that are used in various stages of the workflow (described in the text).

3 Workflow

The OCaml Platform tools that we will demo during the talk will follow this workflow depicted in Figure 2.

3.1 Development

This covers the initial development of a new library from scratch, and is intended to highlight editor assistance and the Platform build tools.

Create Local Package: We will initialize a new package intended to bind to a C library installed on the system.

Write Source Code: While writing the source code (using the Ctypes[1] library), the editor will give feedback to the user in the form of type annotations and code completion using the Merlin IDE assistant.

Build and Local Install: Building takes place using the Jenga build tool, designed to operate on large codebases far more efficiently than alternatives such as `ocamlbuild`. Jenga also has built-in rules for all the Platform tools, and is easily extensible via a library.

3.2 Publish

Push to Git: Once the package has been written, it is time to publish it online. We use GitHub through our demo as the default online site, but the Platform is intended to work with any similar Git-style hosting platform in the future (to avoid lockin to one provider). There is a pure OCaml Git implementation now available in OPAM to make manipulating such repositories easier from within Platform tools.

Code Review: An emergent trend in new OPAM packages has been for users to request code reviews of their newly uploaded package. This is possible, although awkward, in the GitHub interface, and we are evaluating several alternatives (including a home-grown one developed at Jane Street).

Create OPAM Metadata: Publishing a package on OPAM requires specifying some metadata, such as the archive URL and dependency constraints. OPAM 1.2 improves support for this by letting a new package be created by adding an `opam` file directly into the package source code, and prompting interactively for the data.

Submit OPAM Package: This is automated via a command-line plugin to OPAM that stores the user's GitHub login token and creates the pull request to the central Git repository without having to use the web interface.

Platform and Unit Tests: When a new package is submitted to OPAM, it is automatically run through the OCamlot continuous integration system that initially tests it on Ubuntu Linux. The tests are run inside a Linux container or a Xen VM, with extra system libraries specified in the package metadata. Every new package must pass this test before being merged into the package repository. Unit tests can also be run inside the sandbox environment.

To ensure the ongoing health of the repository, daily bulk build regression tests are also run (using a combination of Docker[2] and Xen). The results are logged to a central Git repository[3] and failures triaged by a combination of automated tools and the OPAM developers.

The final set of bulk runs test the package database on "unusual" platforms that casual developers are unlikely to have access to, such as ARM, PowerPC and Sparc devices, and a combination of MacOS X, FreeBSD, OpenBSD and Linux distributions.

3.3 Maintain

Online documentation: When a package is published, a central set of online documentation is regenerated using `opamdoc`. This is built using a patch to the OCaml 4.02 compiler that stores documentation strings in the `.cmt` files. These are gathered across all the packages and combined into a cross-referenced HTML site, with an interactive JavaScript console using the IOCamlJS notebook.

Publish to Local and Central OPAM: When a package passes its unit tests, it can be sent either to the global OPAM repository, or to a custom one designed for a particular set of packages, with the `opam2web` tool building the site. For instance, Citrix maintain their own OPAM remote just for Xen packages, as do the Ocsigen team. The testing tools described here work fully against custom repositories too, permitting different (and potentially conflicting) standard libraries to each maintain their own universe of packages. This is of importance to industrial users who may need to commercially maintain a particular package set long beyond the usual open-source lifecycle.

Multi-arch Binary Releases: While OPAM is a source-based package manager, it is important to allow users to quickly download a binary snapshot for their operating system to get started quickly. The 0install[4] binary distribution system is written in OCaml and supports the redistribution of signed binaries. The bulk build systems can upload binaries to 0install repositories, thus providing binary releases for slower platforms such as the rPi.

4 Conclusions

This talk will demonstrate the workflow and a wide variety of tools that comprise the first version of the OCaml Platform. *Note: This talk would benefit from a slightly longer timeslot due to the nature of the demo.*

[1] `https://github.com/ocamllabs/ocaml-ctypes`

[2] `http://docker.io`

[3] `http://github.com/ocaml/opam-bulk-logs`

[4] `http://0install.net`