OBuilder - Homogeneous builds with OCaml

David Allsopp, Kate Deplaix, Patrick Ferris, Thomas Leonard, Anil Madhavapeddy
Tarides
Cambridge, UK

Antonin Decimo
Tarides
Paris, France

Tim McGilchrist
Tarides
Sydney, Australia

1 Abstract

This talk will present a lightweight sandboxing solution, OBuilder [3], that works beyond the usual Linux containerisation solutions, providing support for macOS, Windows and the BSDs without requiring full (expensive) virtualisation. We will cover the implementation for macOS and Windows, the challenges encountered with providing sandboxes on such different platforms, and how this work is being used to provide cross-platform builds to the OCaml community.

We previously introduced OCaml-CI [4] providing an opinionated, fast-feedback CI system for OCaml projects. Since the end of 2020, opam-repo-ci[1] has provided a similar service for testing pull requests to opam-repository. Both of these systems use OBuilder[2] to provide support for multiple operating systems and hardware architectures.

2 Introduction

Providing a suitable, manageable collection of machines across platforms and hardware architectures is a challenging problem. For our purposes we needed to provide support for building and running OCaml on many operating systems and architectures, both efficiently and reliably. Almost all existing lightweight sandboxing solutions exclude macOS and the BSDs, so we had to build a bespoke solution that worked beyond Linux containerisation but without requiring full (expensive) virtualisation.

For Linux the solution was clear, using runc, existing container orchestration tools and a snapshotting filesystem (ZFS or Btrfs). However for macOS and Windows a number of different approaches were investigated before settling on our current solutions. Originally all builds were running inside Docker, relying on it for sandboxing and caching, but a bug in BuildKit[3] running parallel builds motivated the switch to OBuilder.

To fully reach their goals, ocaml-ci and opam-repo-ci, should support as many platforms as OCaml is available on.

3 OBuilder Architecture

OBuilder is designed to take a build script and perform the steps in it, inside a sandboxed environment. After each step, OBuilder uses the snapshot feature of the filesystem to store the state of the build. Repeating a build will reuse the cached results where possible, presenting the cached logs in place of actually performing the build. For a CI system we are primarily interested in the logs rather than a build artefact.

A system using OBuilder, like ocaml-ci, provides a build script as either a Dockerfile or an S-expression equivalent, describing the steps to perform. This build script is then submitted to a local worker to execute or to a build cluster. For ocaml-ci and opam-repo-ci we have a custom cluster orchestration solution that is also written in OCaml, that provides a way to submit OBuilder specs that are then scheduled across different pools of workers who then execute the build script using OBuilder. OCluster provides different pools of Linux, Windows and macOS workers running on various hardware architectures (e.g. x86, arm64, s390x).

Workers are responsible for providing a sandboxed execution using the implementation available on that plaform, and providing a store for caching build steps to avoid repeating the execution. For each platform we want to support, we need to provide a solution for the sandbox and store. On Linux the sandbox is provided via runc and native containerisation, while the store is provided by Btrfs or ZFS and its support for efficiently snapshotting the filesystem.

4 Implementation on macOS

Supporting OBuilder on macOS required overcoming the major shortcoming that macOS does not provide a native containerisation solution. You cannot carve up macOS using namespaces like you would Linux. However you can have multiple users on a single machine, each with their own home directory and ability to execute commands. This approach forms the basis of the sandboxing on macOS, user isolation.

[1] https://opam.ci.ocaml.org
[2] https://github.com/ocurrent/obuilder
[3] https://github.com/moby/buildkit/issues/1456

OCaml Workshop '22, Sept 11–16, 2022, Ljubljana, Slovenia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Opam supports having multiple opam roots on a system typically in the home directory of the user `~/.opam`, which complements the user isolation approach taken. You can run something like `sudo -u macos-builder-705 -i /bin/bash -c 'opam install irmin'` and it will use macos-builder-705's home directory to build irmin. This provides a level of isolation similar to runc on Linux.

Homebrew, a macOS system package manager, used by opam depext for installing native libraries, is not so flexible with the official documentation saying you could install it elsewhere but *"Pick another prefix at your peril!"*[4]. We needed to containerise homebrew using FUSE (file system in user space) to setup a faked system installation of packages per user, which then gets used by Opam.

The native macOS file system APFS doesn't provide the snapshotting control required for OBuilder. There is a port of ZFS that provided some promise but after many hours were lost debugging many small bugs in ZFS, we opted for a portable and reliable solution using rsync. The obvious trade-off being it was slower and memory inefficient using rsync as the store backend.

Equipped with both rsync, user isolation and a sense that we needed to perform some file system indirection, a macOS implementation of OBuilder emerged.

5 Implementation on Windows

We considered using a similar model for Windows. However, Windows does have native containers and it does have the tools for assembling a snapshotting filesystem. Direct access to these technologies is considerably harder and, more importantly, less stable than their Linux counterparts, so we turned to using Docker on Windows. This has allowed us to address a shortcoming of Docker for Windows which prevents building Docker images using more than 2 CPUs on the host. While using Docker directly might seem straight-forward, implementing it required fixing a number of OCaml's Unix library Windows port and LWT bugs and some creative use of Docker. Two modes of sandboxing are available: either full virtualisation through the Hyper-V hypervisor, or process-level isolation using Windows Native Containers. We use VMs by default, with the cost of longer boot and execution time, but hopefully better security and stability.

6 Future Work

The OBuilder support described for macOS/x86 workers is already available in OCluster, while for Windows workers we have tested deployments and succesfully run builds on them. The next stage is scaling up the usage of both worker types in ocaml-ci and opam-repo-ci, and to scale out the underlying hardware for macOS/x86 and macOS/arm64. As a consequence, when packages are released on opam, their support for Windows and macOS is demonstrated as they are built.

On Windows there is an experimental container orchestration tool hcsrun that we would like to try, as well as snapshotting filesystem support Btrfs for Windows [1]. These native solutions promise better resource usage of the underlying hardware, translating into more capacity for running builds.

Supporting the BSDs is another focus, with the preliminary support for Docker on FreeBSD we have experimented with using runj [2], (a new experimental, proof-of-concept OCI-compatible runtime for FreeBSD jails) to add FreeBSD support and reusing ZFS from Linux support for snapshotting the file system.

[4] https://docs.brew.sh/Installation#untar-anywhere

References

[1] Mark Harmstone. 2016. *WinBtrfs - an open-source btrfs driver for Windows.* Retrieved June 10, 2021 from https://github.com/maharmstone/btrfs

[2] Samuel Karp. 2021. *runj: a new OCI Runtime for FreeBSD Jails.* Retrieved June 10, 2021 from https://samuel.karp.dev/blog/2021/03/runj-a-new-oci-runtime-for-freebsd-jails/

[3] Thomas Leonard, David Allsopp, Antonin Decimo, Kate Deplaix, Patrick Ferris, and Tim McGilchrist and Anil Madhavapeddy. 2022. *OBuilder - Experimental "docker build" alternative using btrfs/zfs snapshots.* Tarides. https://github.com/ocurrent/obuilder

[4] Thomas Leonard, Craig Ferguson, Kate Deplaix, Magnus Skjegstad, and Anil Madhavapeddy. 2020. OCaml-CI - A Zero-Configuration CI. In *OCaml Workshop 2020 (OCaml Workshop 2020).*