Continuous Integration for Snabb Switch

Max Rottenkolber <max@mr.gy>

Monday, 30 November 2015

For over year now, I have been working on *Snabb Switch* (`https://github.com/SnabbCo/snabbswitch`), an open source software Ethernet networking stack. One of my ongoing responsibilities has been to devise, develop and maintain a *continuous integration* (CI) tool for the project. Wait, why would anyone spend time "developing" a new continuous integration system with *Jenkins* and *Travis CI* being around?

Snabb Switch is Different

Snabb Switch is being developed as an open source project by a diverse community of interested parties. The "upstream" repository, through which the consensus of development is mediated, is maintained by *Snabb GmbH* (`http://snabb.co`). Other interested parties, be it individuals or companies, maintain their own "forks" of Snabb Switch in which they develop features specific to their interest. Every month, upstream releases a new version of Snabb Switch, with which the forks are recommended to merge with. The release itself consists of commonly beneficial contributions pulled in from individual forks during the course of a month. This work flow enables Snabb Switch development to take place in a loosely decentralized fashion: contributors can do *their thing*,[1] and propose bug fixes and new features that they feel benefit a wider audience to be merged into upstream while receiving everyone else's contributions too. Snabb Switch development is coordinated on *GitHub*.

Our testing infrastructure is built with this decentralized organization in mind. Often, a fork will develop a Snabb Switch application that itself has no place in the upstream repository because its too specific to a certain problem or product.[2] These applications usually have their own test suites and benchmarks in addition to the upstream test suite as well, and the testing infrastructure provided by Snabb GmbH supports that use case too.[3]

Testing a high performance Ethernet networking stack requires high performance networking hardware. Further than that, a big part

of our test suite are *performance regression tests*. Performance regression tests are benchmarks in which Snabb Switch may not score below a certain threshold. In order to run a representative benchmark you need to run it in a deterministic environment. Specifically, we need a dedicated "Network Interface Card" (NIC) and an idle CPU core on the same *NUMA node* (`https://en.wikipedia.org/wiki/Non-uniform_memory_access`). These requirements already exclude us from using most hosted solutions.

Even further, we support multiple NIC drivers that have to be tested with the right hardware as well as a broad range of different hardware and software platforms. Some parts of the test suite even spin up virtual machines using *QEMU* to test virtualized networking. Different CPU models, different operating systems and different user land software versions: all these are factors that can impact the performance of Snabb Switch and thus need to be covered by the test infrastructure.

Another important factor to effectively cooperating on a peer developed software system is *reproducibility*. Whenever somebody produces a test result for review it needs to be reproducible so that others can confirm the result. Traditionally, truthfully reproducing a given test result, which may depend on the specific hardware and/or software in use, can be a tedious and time consuming task. Ideally, that process should be simple and fast. Especially for Snabb Switch, which supports a broad range of different platforms, there is demand for reducing the overhead inherent to reproducing different test environments.

- 1. The different parties working on Snabb Switch indeed have different goals, needs and specialties.
- 2. There might even be active closed source forks tracking upstream development. Hard to tell.
- 3. Snabb GmbH sponsors the development of our custom continuous integration tools as well as the "Snabb Lab": a flock of development machines that sport the right networking hardware, accessible to anyone who wants to tinker with Snabb Switch.

Rewritable Software

The Snabb Switch community encourages its members to produce *rewritable software* (`https://github.com/lukego/blog/issues/12`). The term *rewritable software* describes the ideal of a simple and easy to understand program distilled from prolonged experimentation and iterative

design. Distilling simple programs requires us to try out many possible solutions just to throw most of them away afterward.

Big dependencies get in the way of that process: first you need to learn how to use a complex system, and then it will be hard to adapt to new requirements. We prefer a 200 line *Bash* script that does exactly what we *think* we want over a complex Java application that probably only does a subset of what we *think* we want. Small, orthogonal programs on the other hand fuel the process. In fact, we reuse *tons* of Unix command line tools in our test suite: be it *iperf* to benchmark throughput or *awk* to compute the median and standard deviation of a benchmark's results.

Continuous Integration as Part of the System

At the core of the Snabb Switch test suite is a *Makefile*. It has a target *test* that runs "all the tests":

- Every Lua module can optionally provide a function aptly named *selftest* which is invoked to run a module's unit tests. (This is a convention imposed by Snabb Switch.)
- Additionally, any file named *selftest.sh* is considered a test and run as such. This mechanism allows for "free form" testing that entails things such as spinning up virtual machines.
- Some tests can or must be configured via *environment variables* and the result of a test (success, failure or "skipped") is indicated by its return value.

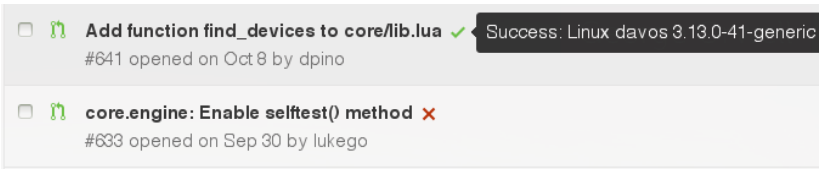
Some of the individual tests are small and independent, some require a specific piece of hardware, some even require a couple gigabytes of assets such as virtual machine images to run. A good example of a complex test script is the *Snabb NFV* test suite: it comes with its own abstraction layer (150 lines, *Bash*) to facilitate spinning up virtual machines connected to each other via *Snabb NFV* and running commands on the individual guests such as *ping*, *iperf* and *nc*. This abstraction is used by some benchmarks as well.

A Snabb Switch benchmark is just a shell script that prints a floating point number indicating a score. These scripts can again be arbitrarily complex. The target *make benchmarks* (20 lines, *Bash*) runs all benchmarks and prints their names followed by their score. The sample size of *make benchmarks* can be configured through an environment variable in order to compute median and standard deviation values.

```
$ SNABB_PERF_SAMPLESIZE=5 make benchmarks
basic1-100e6 17.9 0.352136
[...]
```

Possible output of *make benchmarks*: The *basic1-100e6* benchmark ran five times with a median score of 17.9 and a standard deviation of 0.352136.

As for the reproducibility of test results, we maintain *Docker* images publicly hosted on *DockerHub* that contain all the external, variable assets required to run the full Snabb Switch test suite. A script (20 lines, *Bash*) is provided to seamlessly run the test suite of your Snabb Switch working tree within such a container. That way, test results depending on a specific software environment can be quickly reproduced on any machine that runs *Docker*. The creation of our test environment and the containing *Docker* image is another complex hassle that I only want to touch briefly. For instance, we need to repeatably build virtual machine images containing specific versions of some software. For that, we *make use of Docker* (`build-vm-image-with-docker.html`) too.



Finally, there is *SnabbBot* (200 lines, *Bash*): Snabb Switch's own tailor made continuous integration service. *SnabbBot* is a *cron job* that launches automated test runs for incoming contributions:

- It uses the *GitHub API* (`https://developer.github.com/`) to fetch new *pull requests* and post a status with a log attached, which is again posted as a *gist*.
- It merges the *pull request* branch with the *master* branch.
- It uses *Docker* to build and run the test suite within a container.
- It uses the *Makefile* target *test* to run the test suite and, on failure, attaches the test output.
- It uses the *Makefile* target *benchmarks* to compare performance between the *pull request* branch and the *master* branch and detects performance regressions.

- Multiple instances of *SnabbBot* can provide results for a single Snabb Switch repository using different test environments and it is *trivial* (`https://github.com/SnabbCo/snabbswitch/blob/master/src/doc/testing.md#running-a-snabbbot-ci-instance`) to run *SnabbBot* on a forked repository.

A lesson learned along the way: continuous integration should not be an external tool, but part of the program it tests in the same way a program's test suite is part of that program. Every piece of the continuous integration puzzle is part of the mainline Snabb Switch source tree (with exception of the variable test assets and more importantly, the scripts that build them). Continuous integration as part of the system. Thanks for reading!

Interested in Snabb Switch? I offer Snabb Switch consulting! Feel free to contact me at service@mr.gy.