TUXERA

CONTINUOUS MONITORING OF PERFORMANCE BENCHMARKS IN KERNEL FILE SYSTEMS DRIVER DEVELOPMENT

Anoop Vijayan
DevOps Lead, Tuxera

Updated: 26 January 2018

Tuxera Inc., Westendintie 1, 02160 Espoo, Finland
Tel: +358 20 764 1720 • Fax: +358 98 565 7002 • info@tuxera.com • www.tuxera.com

Abstract

Continuous Integration[1] (CI) has become a very popular addition to software development cycles. With this process, companies are reaping great-quality software development. Typically, continuous integration comprises of building (compilation and packaging), followed by smoke testing. In some cases, this system is extended to continuously deliver software products to production systems—an approach called Continuous Delivery[2] (CD). Monitoring performance is a key aim, but it usually comes at a later stage, in the CD pipeline. However, this article describes the benefits of executing the performance benchmarking earlier in the smoke testing phase using Elastic Stack[3]. Elastic Stack comes with many helpful features including graphing and alerts.

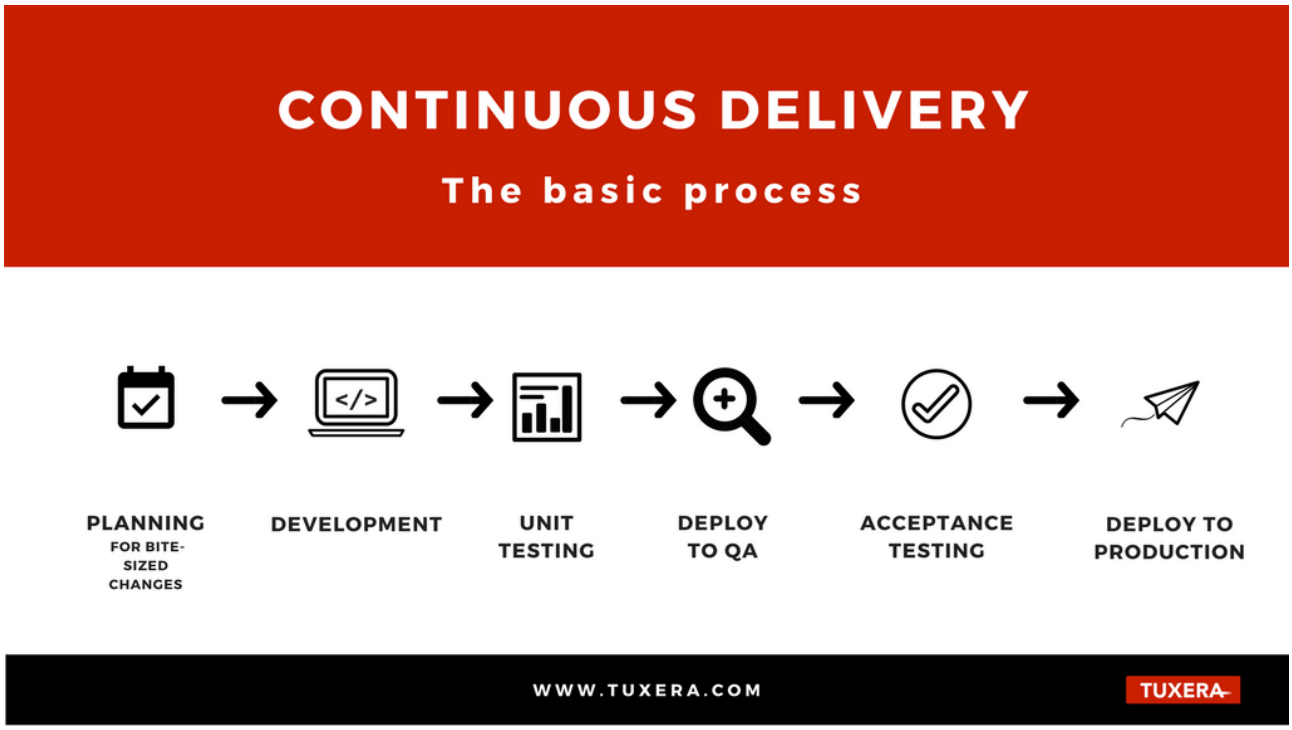
Introduction

What is CI? From Wikipedia, Continuous Integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day. This is the first step in most DevOps practices and primarily aims to solve integration problems. Usually the software is checked out from the version control system, compiled, and packaged. This is called the **build** phase. In normal cases, there are builds for every change to the software repository. There is also the **smoke testing**[4] phase, which comes immediately after a successful **build** phase. If both phases are successful, the committer or developer manages to successfully push the changes to the system. Then, the system gets ready to receive the next change.

A CI system is meaningless if any of these phases takes hours to finish. The biggest challenge with this approach is to have a faster **build** phase, because smoke testing is typically a very minimal set of tests to verify the software.

After establishing a reliably working CI pipeline, and if there is need to have a faster release process, then the Continuous Deployment or Delivery (CD) pipeline should be implemented. In most systems, CD implementation comes following CI. Also, CD is far more difficult to achieve due to the duration and bureaucracies around it, as it involves multiple phases of testing: integration testing, system testing, end-to-end testing, performance testing, and acceptance testing, to name a few. A software being tested does not have to undergo all these testing phases, but at least some are based on the product or customer needs. Put simply, the CI's scope is until "deploy to QA" whereas CD's scope extends until it is deployed to production (see figure on page 3).

In kernel file systems development, where read/write speed is a crucial factor, performance testing and short-long term performance degradation is important. Performance testing indicates how fast we execute operating system kernel operations. Poor performance indicates slower kernel operations that cause a poor user experience. Anything less than **10%** than the previous performance values are not accepted. Also, it is important to visualize how we have been performing over time, as short-term degradation should not accumulate to a higher value.



Problem

It is apparent that faster performance testing is important to achieve faster CD, which is challenging. This is due to factors such as performance testing duration, requirement of real hardware devices, a huge variety of kernel-architecture-file system combinations, amongst others. Taking only the duration factor for slow CD, there are many options which are becoming industry standards. Some advocate that a software can be slightly tested and delivered to a customer in a **Canary release**[6] approach so that it is easier and faster to rollback. There are also approaches which perform regression release testing after the software is in production. This method will always roll-in with fixes and never rollback to previous versions.

Solution

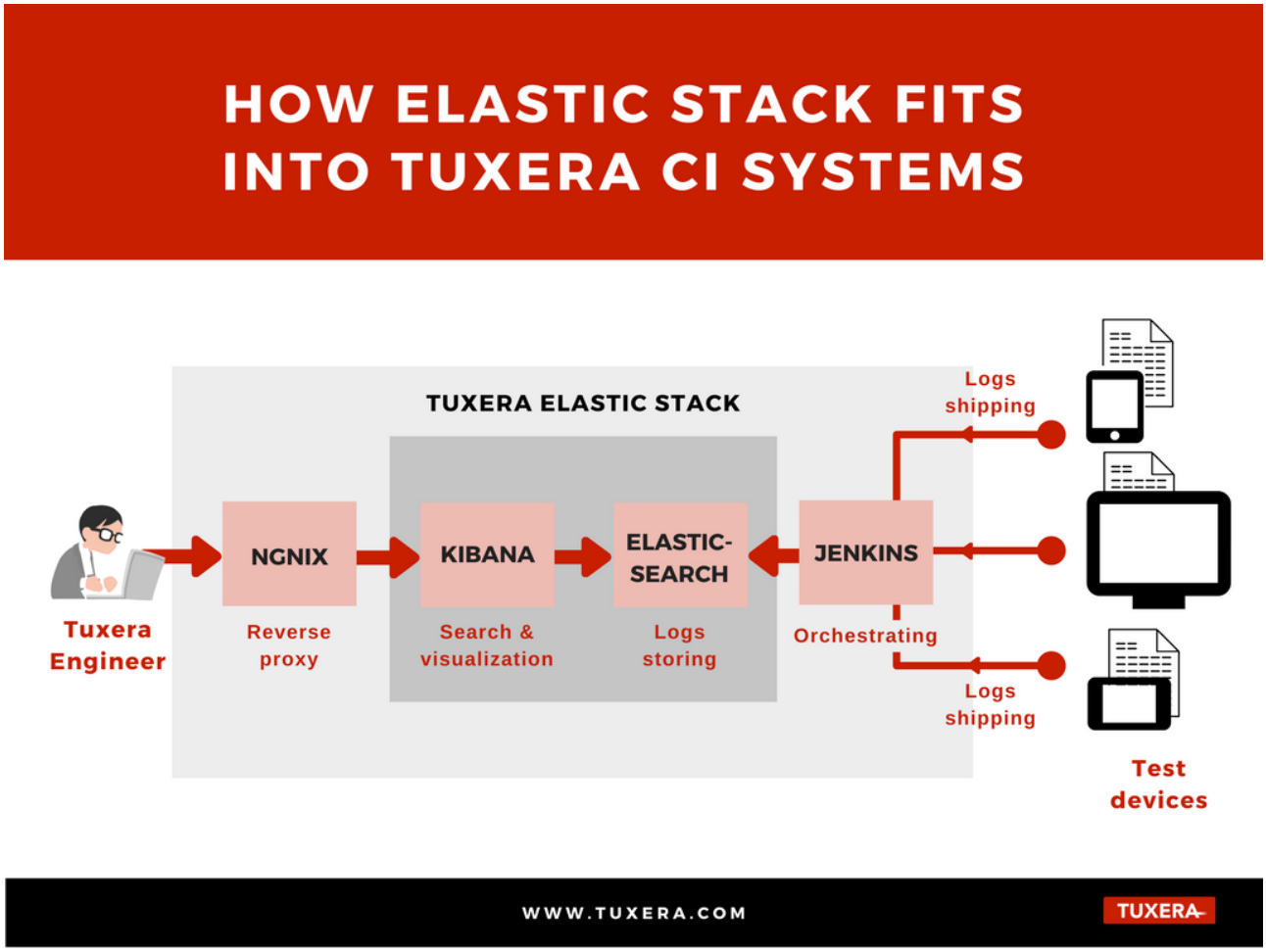
At Tuxera, we have performance bench tests for our kernel file system driver products. We run them as a part of the CI pipeline in the smoke testing phase in parallel with other smoke tests. They are called quick performance testing and are run on every commit. The tests are run on a specific kernel-architecture-file system combination that is fast and already indicates any performance improvement or degradation. After a successful smoke testing phase, this is promoted to a full performance testing phase, where we test for more kernel-architecture-file system combinations on various real hardware devices which are already configured.

This solution is based on an early testing approach, facilitated by shifting the phases which usually exist after CI, but before CD. Earlier testing gives us insight into driver performance already at the smoke testing phase.

Analyzing performance test results

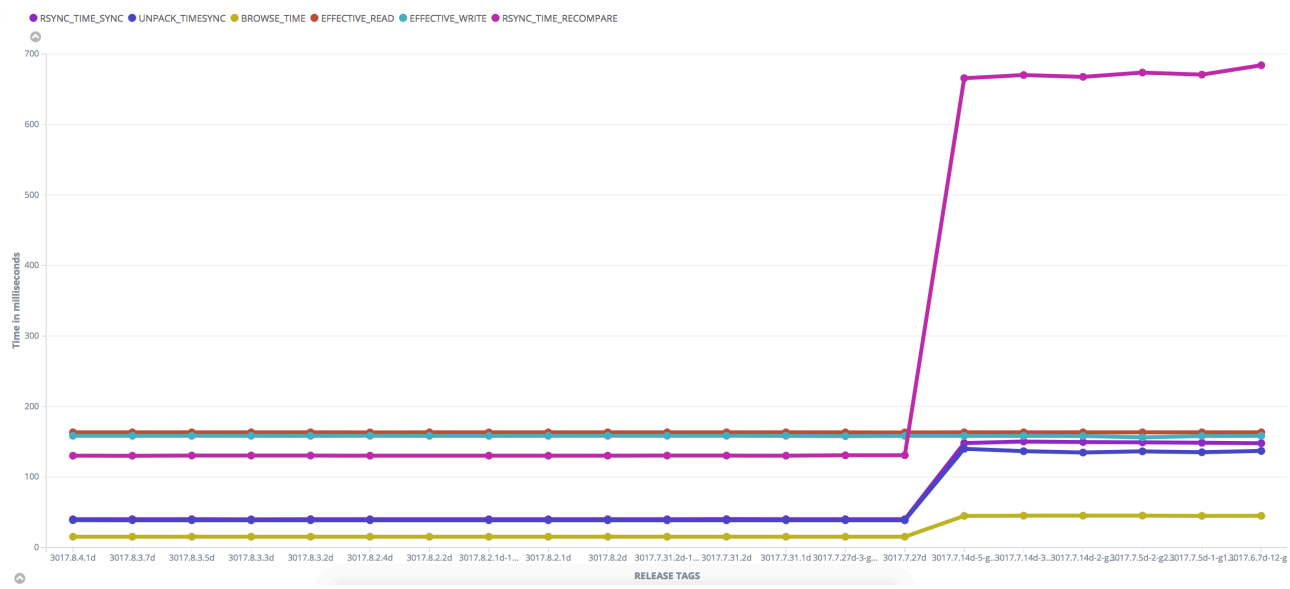
Performance tests usually produce a lot of results, especially due to the fact that they are run on a per-commit basis. We use Elastic Stack, which indexes and graphs the performance test result of every commit.

Elastic Stack consists of components: Elasticsearch, Logstash and Kibana—which store, process and index, and visualize logs, respectively.

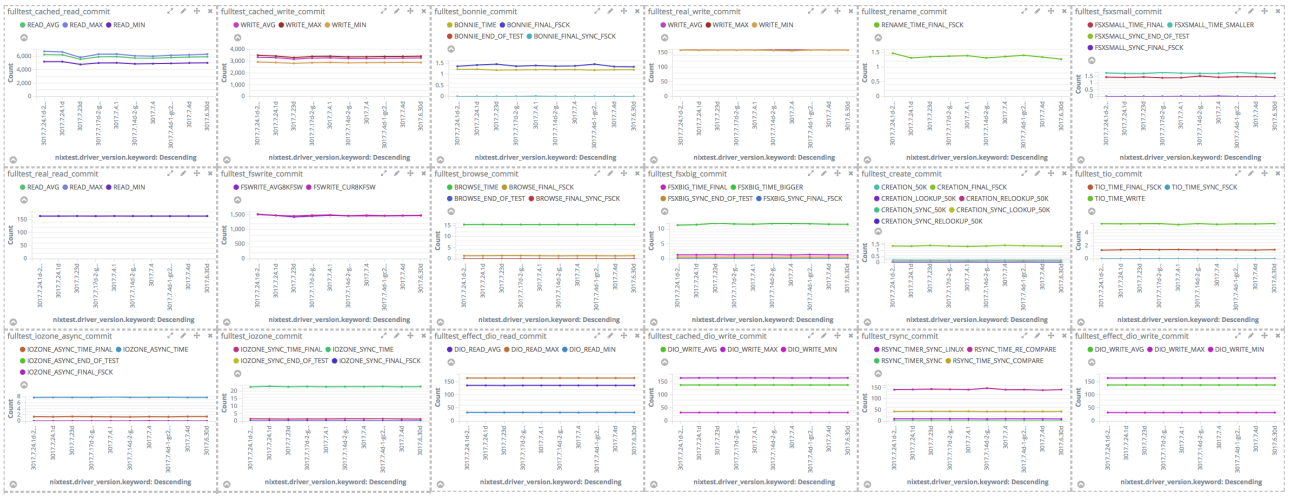


From the figure above, **Jenkins**[8] acts as an orchestrator executing tests on devices, fetching and shipping the logs to the Elastic Stack server. For simplicity, the logs are converted to json format so they can be fed straight to Elasticsearch, where they are seen from Kibana instantly. An Nginx reverse proxy acts as a frontend serving the user interface.

The graph on the following page shows the behavior of our unpack, browse, read, write, and rsync tests over Git commits tags on full performance tests. The y-axis indicates the delay in milliseconds for an operation. The x-axis indicates Git commit tags. As one can see, there has been a clear improvement of 75% on rsync speeds as per the 27 July commit.



This is just a sample example. The overall picture of critical performance tests looks like this:



*Please get in touch with us (info@tuxera.com) if you would like to know more specific details about our tests.

Discussion

The CI approach, which is a byproduct of agile practices, defines integration of software components well ahead of the development cycle in comparison with the older waterfall model. However, not many advancements have been made concerning the practices over the years. Our approach, which takes the performance benchmarking phase one step ahead into CI to find problems well in advance, is clearly an improvement for us. It is also evident that huge log processing tools like Elastic Stack have helped us easily store, process, and plot large amounts of logs for our purposes.

Future enhancements

Currently, we are analyzing the graphs manually, which on rare instances can be laborious and slow before the committer is notified. It is also possible to define alerts in Elastic Stack. This means that the committer can be notified whenever the performance drops below a certain range. This is a work in progress.

Conclusion

It is always better to verify software products well ahead in the pipeline to ensure a very low cycle time. To make the CD faster, it is also a good practice to push some phases in CD ahead into CI if that is feasible. Apparently, all that counts is to reduce the cycle time for a faster product release to production, which this approach tries to solve.

References

1. https://en.wikipedia.org/wiki/Continuous_integration
2. https://en.wikipedia.org/wiki/Continuous_delivery
3. https://www.elastic.co/products
4. https://en.wikipedia.org/wiki/Smoke_testing_(software)
5. https://medium.com/@raddougall/moving-to-continuous-delivery-is-hard-but-well-worth-the-effort-a0f4b492b12b
6. https://martinfowler.com/bliki/CanaryRelease.html
7. https://www.elastic.co/webinars/introduction-elk-stack
8. https://jenkins.io