QA FOR MISSION-CRITICAL SOFTWARE:

The Comprehensive Testing of Tungsten Clustering and Replicator

Patrik Michalak

October 2021

CONTACT

1957 Joseph Dr., Moraga, CA 94556

marketing@continuent.com

Table of Contents

Part 1: Introduction

Quality of software is a critical topic in the IT world today. Agile methodologies in software development brought new opportunities and customized development processes, which make releasing software easier and faster. But, for this reason, many experience that a lot of new, small IT companies provide "fast cooked" software, with none, or very poor quality assurance (QA).

> "Whatever can happen, will happen."
>
> Murphy's law

At Continuent, we develop business-critical and mission-critical MySQL database clustering and replication software that is used to protect billions of dollars of revenue each year. For this reason, and the fact that our software runs in many different environments and covers a variety of use cases, we have a bit of an extensive testing process.

This whitepaper will bring you closer to the world of QA at Continuent. We will talk about the most important and, of course, interesting parts of our QA.

Starting from the Bottom

Tungsten Clustering and Tungsten Replicator are written in the Java programming language. At the very beginning of the QA process, the source code is built, and individual components and small pieces of code are immediately tested using the *JUnit* framework. This way we test our software on code level - every function and every possible line.

But that's not all we can do to test on code level. Every time any developer makes any change in code, or even new features, the rest of the developers do a short code review. This is very effective for double-checking with another pair of eyes with minimal effort. The main goal is to find inconsistencies or possible bugs in the code of the other developer, but the other advantage is that developers are always up-to-date on code written by others, and thus they understand "how it works."

The World of a Continuent QA Engineer

At Continuent, there is a huge space of possibilities of how to test, and what to test; there are endless combinations of environments and different setups.

Work in QA is not isolated to component development. The QA engineer must have “a little bit” of knowledge in all possible IT professions in the company. It’s not about monkey-style testing. One must know networking, system administration, automation tools, follow new technologies and have a sense for architecture design.

Every day brings different problems to solve. We work day after day on something new.

Part 2: Simulating Realistic Conditions

After Tungsten components are built, we test their functionality in the most realistic conditions possible; this means simulating the dynamic environments running on customer servers in Production. To achieve this, we need tens of servers ready to deploy Tungsten software, perform many different tests and in all the different use cases. Thankfully we are living in the future, and we can use cloud technologies to reserve the computing power and quantity of virtual servers to fit our needs. For this purpose, we leverage *Amazon EC2* instances in AWS.

Always With a Clean Shield

Every time we want to build and test (or just test) our software, we create a fresh new group of instances. This proves to us, we always have a clean environment for testing and all packages are up to date and with latest versions. After testing is done, and we are no longer interested in those running instances, we terminate them. This way we are not wasting money for constantly running servers and, more importantly, we don't face environmental issues or inconsistencies.

Developers are not dependent on one shared group of servers! We create groups of servers on demand; each group is dedicated to one build/testing round. Also, one testing process could create 1 to 6 parallel child processes, each having its own group of servers and running tests from a shared pool of test suites.

When we say "we do this, or that" we mean - these processes are fully automated! Starting servers, dependencies preparation, build software and then test software in thousands of different ways and finally terminating servers and reports creation. As folk wisdom says: "Never spend 10 minutes doing something by hand, when you can spend 10 hours failing to automate it!"

Keep It Lightweight

In the past, we put a lot of libraries and packages that were not defined as *prerequisites* for Tungsten Clustering, just to make our testing process more comfortable for us. We found out it's not best practice because our customers don't have all packages we use for testing installed. That made our testing environment "not clean enough". Now, we install only the packages and third party software that we define in our Documentation as prerequisites for Tungsten.

And what about libraries that make test development easier? We basically removed all dependencies from tests and testing tools and did everything on our own. We have our own testing frameworks written from scratch, which fully fits and is targeted to testing only our products. Anyway, there is no public testing framework, fitting and able to test software like Tungsten Clustering and Replication. For DevOps we use tools like Ansible, but for testing we prefer to have it all under our own tight control.

Too Many Linux Distributions

At Continuent, we make sure that our Tungsten software is able to run on many different operating systems and environments. For this reason, we test on multiple different Linux distributions. In the article Automated MySQL Server Preparation Using Ansible you can find a description of the Ansible tool and source codes that we use for installation and configuration of prerequisites on our servers, including on different supported Linux distributions.

As every Linux distribution has a slightly different file system, we look for performance and security 'holes' by testing in different Linuxes. During '*multi-linux*' testing we cover:

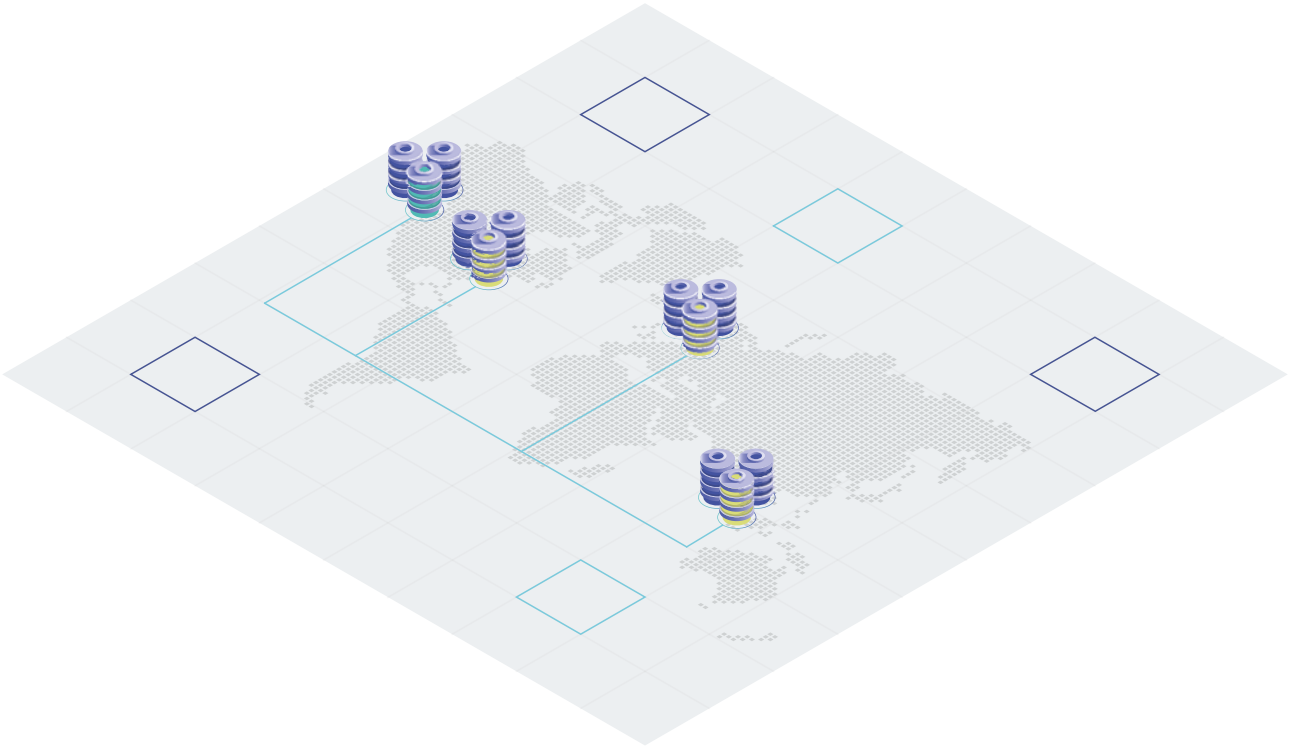
- Amazon Linux 2
- CentOS 7
- CentOS 8
- RHEL 7
- RHEL 8
- Debian 9
- Debian 10
- Ubuntu 18.04 LTS
- SUSE Linux Enterprise Server 15

A Little Big Playground

When Continuent discusses Composite Active/Passive and Active/Active topologies, one of the use cases is *Geo-scale multi-region* MySQL Clustering. Communication between servers located around the world brings new unexpected environment conditions like high latency and other performance issues. For this reason, we test our composite topologies across multiple regions.

The solution is pretty 'easy;' we reserve a few servers in multiple regions, then we set up networking between them and run the tests. For example, when we are testing cross-region replication and clustering in a Composite Active/Passive topology, we prepare 3 instances per each of following regions in AWS:

- US East (N. Virginia)
- Europe (Ireland)
- Asia Pacific (Singapore)



MySQL Versions

Last but not least we test our software against all possible *versions of MySQL*. Tungsten Clustering version 5.4.0 (released in 2019) brought full support of MySQL 8, during 2020 we certified our software for Percona Server distributions, and Tungsten Clustering version 7.0.0 will bring full certification for distributions of MariaDB.

At the moment, it is possible to test on servers with the following distributions of MySQL server:

- MySQL versions 5.6, 5.7 and 8.0
- Percona Server versions 5.6, 5.7 and 8.0
- MariaDB versions 10.0, 10.1, 10.2, 10.3, 10.4 and 10.5

Environments Matrix

As described above, we are able to set up pretty various and dynamic environments for testing. 9 Linux distributions, multi-region, 12 MySQL versions in total... oh, and we are also interested in testing different Java versions! We are able to set servers on Java versions from 8 to 13, but most important are long term support Java versions 8 and 11.

Now, if we want to test all possible combinations of Linux, MySQL, Java and include multi-region testing, it will get us a massive multidimensional matrix of possible environments. Just for information, theoretically, with all our test suites, it takes weeks to finish tests on all possible environments we can prepare. Also we calculated with 6 parallel processes which could be busy working on this task. This is why we defined some combinations, they are necessary to be tested and also 'levels' of testing - from the easy smoke testing to advanced testing of release candidates.

Part 3: Bug Hunting

Functional, non-functional testing, smoke testing, integration testing, performance testing, regression testing... Yes, we have them all! Yet we don't divide our tests into categories by *theoretical software testing methodologies*. There is a difference between testing mobile applications and testing complex software like Tungsten Clustering and Replication. Of course we follow recommended testing practices, but our tests depend fully on our software architecture and its components.

We don't use any public testing frameworks even though that would ease the testing process. This means more work for us, but gives us absolute freedom and control over our QA environment and tests.

In the last section we described how we prepared a testing environment that fulfills our needs. Next, we want to deploy Tungsten software to those servers, install the topology, run tests, uninstall the topology and collect as many logs and as much information as possible.

Rubber Duck Method

Let's focus now on the most advanced testing approach in our company - we test our software mostly as a whole system, installed on virtual servers, simulating realistic conditions and scenarios. Our testing tools and Tungsten software are independent units on those servers. Testing tools see Tungsten software as a *black box*, but Tungsten software allows them to look inside the box through an interface, which uncovers a lot of information. Also, the testing tools know how to read the log files of Tungsten components, or where to look for them in the environment.

Rubber ducks busy in debugging our software - Amsterdam, October 2019

Meet our main testing tool which is named *Quackbook*! At Continuent, besides the elegant swans in our logo, we love rubber ducks as a symbol of our QA. Quackbook is an intelligent testing tool, which is able to install, uninstall and test any topology in any environment. It fully understands the topology, its states, inputs, operations, and outputs. The tool is easily modifiable (it's very easy to add new functionality) and offers a lot of possibilities and options. It's a software written in pure Perl programming language without any external dependencies (libraries or modules).

From our perspective, we divide the tests into two important categories - testing features and testing scenarios. We can say that feature testing is also a kind of scenario testing. The difference is in test implementation. Automated tests for features are integrated right inside the source code of Quackbook. They test overall functionality from the top to bottom, from bottom to top and also from side to side. ;-) They test all supported topologies (Standalone cluster, Composite Active/Active and Composite Active/Passive), all supported installation methods (staging and INI installation method) and all supported cluster operations.

On the other hand, every testing scenario is a standalone script (which usually covers one use case) and uses available Quackbook functions to understand and operate a topology. These scenarios could be written in Perl, calling any function in Quackbook, or in a JSON notation understandable by Quackbook. Writing test scenarios is a very easy and fast way for developers to cover every developed functionality with a test. Not only are new features tested with these scenarios, but also fixed bugs, support cases, edge scenarios and non-standard configurations can be tested this way.

Running Tests Under Load

Once we have Tungsten Clustering installed on our servers, and before we start testing, one additional step is required - run a continuous load of queries into the database. For this purpose we use the tool Bristlecone, which generates mixed transactions. Continuent has released the Bristlecone Testing Tool under a GPL v2 license and the source code is available to the public.

Targeted Tungsten Components

In our QA, we test Tungsten Clustering as a system, but also separate components, parts and related tools:

- Tungsten Clustering as a whole system - all possible topologies, installation methods, SSL, servers configuration and roles, servers reporting chains, replication between servers and services, connections through connectors, and a lot of more...
- Cluster operations - switch, failover, recovery, backup and restore; including switch, failover and recovery on service level in Composite Active/Passive topologies.
- Tungsten Replicator - homogenous and heterogenous replication topologies.
- Tungsten Connector - establishing and testing connections using different programming languages and frameworks.
- TPM (Tungsten Package Manager) - installations, updates, upgrades, various TPM commands.
- tungsten_* tools - helpful scripts

Testing the version *7.0.0* of the Tungsten product suite constituted a major part of our QA work in 2021. This major release is complete with new features, functionality and improved security. For this reason we expanded our QA with:

- API v2 (RestAPI)
  - Enables better performance, lower overhead
  - Can be used to replace the older monitoring scripts with the lighter-weight API calls
  - Includes a new convenience tool called tapi with functions to help with daily admin tasks
  - Updates the vast majority of Tungsten CLI tools so users may optionally use the API v2 interface when desired
  - Lays the foundation for both Kubernetes development and Tungsten Cloud
- Security Enhancements
  - SSL within all cluster layers is now enabled by default
  - Added support for TLSv1.3
  - On-disk THL Encryption now available
- Performance Improvements

  - At-rest THL compression
  - In-flight THL compression
  - Tungsten JDBC driver
- All New Tungsten Dashboard
  - New optional REACT front-end with impressive design
  - Significant performance improvements
  - Integration with API v2
  - Many new features, including notes per node
- Enhanced Monitoring
  - Prometheus integration - full metrics now available for Connector, Manager and Replicator
  - New `tmonitor` command-line tool
  - Dashboard integration with Prometheus and Grafana
  - The Nagios and Zabbix checks are also available via API v2 using the `tapi` tool.
- Improved Management
  - Audit Logging now available within the Connector
  - Improved Backup/Restore and Reprovision support, including support for `mariabackup`
  - the `tpm diag` command has been polished in many areas
  - `tungsten_provision_slave` has been renamed and improved to `tprovision`
- Replication Updates
  - MariaDB 10.3+ now fully supported
  - The replicator will now be able to handle new SQL_MODES available in later releases of MySQL and MariaDB, these are as follows:
    - MySQL: TIME_TRUNCATE_FRACTIONAL
    - MariaDB: TIME_ROUND_FRACTIONAL, SIMULTANEOUS_ASSIGNMENT
- IPv6 Support, RedHat/CentOS 8 Certification, and more!

So...How Many Tests?

It's hard to say what the "number of tests" really means. We may distinguish 3 groups based on the 'level':

1. *Test suite* - group of scenarios, which are somehow related together (by topology, by component, etc.)
2. *Test scenario / tests for feature* - group of commands and 'checks', to reproduce and fully exercise some feature or real life scenario
3. *'Checks'* - small (base) tests anytime during scenario lifetime, determining if current state is correct

In these terms, we may say that we have:

- **48** unique test suites, but including (reduced) matrix of 9 possible Linux distributions and 12 RDBMS versions, we run during release build **139** test suites in total
- During release testing we also have:
  - **3,434** tested scenarios
  - **156,503** checks (base tests)

Not counted in the above statistics, we also have JUnit tests during Tungsten product build time, which are testing on code level (specific code lines or functions).

Part 4: Testing Time and Test Suites Management

If we could divide our QA into layers, at the bottom layer there are tests of specific features and scenarios. These consist of steps, how to reproduce a certain feature, and how to validate that feature. Individual tests of features and scenarios can be grouped together according to similar characteristics, for example: tests for features/scenarios grouped by topology, tests for TPM, connector tests, API tests...let's name these groups of tests: "*test suites.*"

CI/CD Tools

At the very beginning of our CI/CD pipeline, source code and test suites are handed over to our Bamboo server. Bamboo is a continuous integration tool from Atlassian that we use to externally build Tungsten products and run tests on them (and do other automated processes as well). Bamboo is comparable to other CI/CD tools: "Give me instructions, and I will process them." It's a pretty straight-forward tool, where one '*Plan*' (CI/CD pipeline process) consists of *serial 'Stages'* (CI/CD steps in pipeline) and each Stage could have any number of *parallel 'Jobs'*.

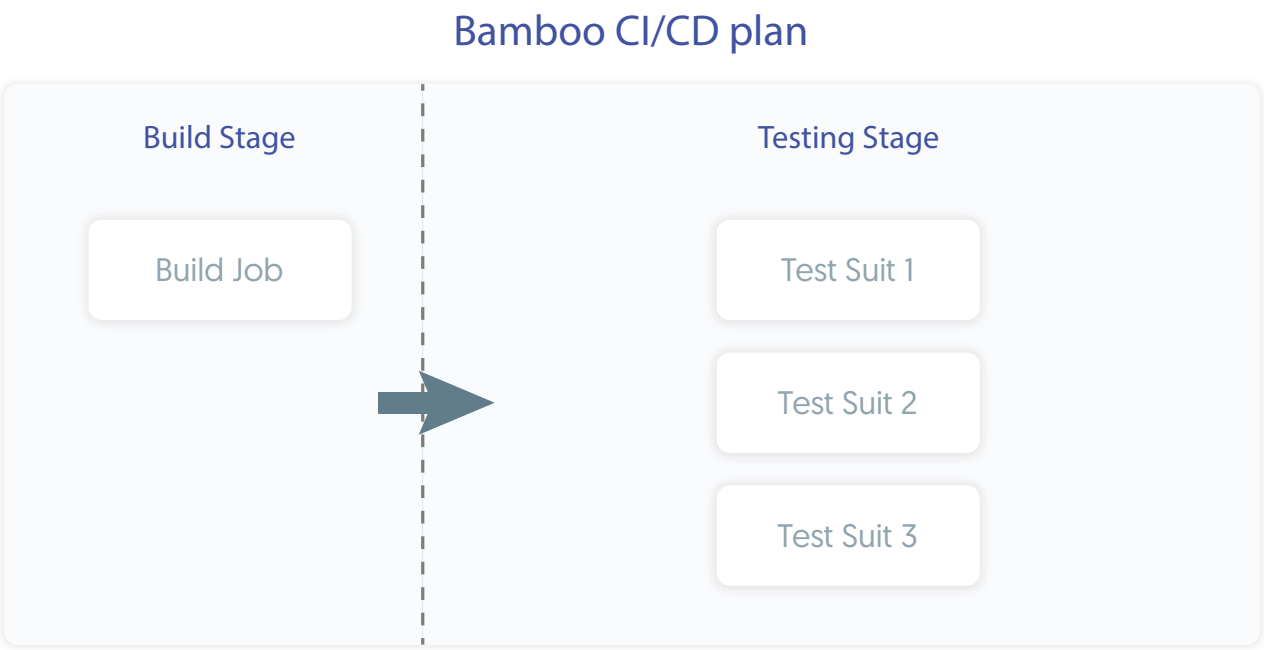
Bamboo was designed to run Plans (CI/CD pipelines) always in the same way:

- "Here are Stages, for source code build and testing"
- "Here are Jobs for running test suites in parallel"
- "Want to skip some Stage or Job? No, dynamic builds are not allowed. Sorry."
- "You can create more Plans, trigger them when needed and provide artifacts between them."

In short, we are not using this tool as it was originally designed to be used...

When we are developing, we need some kind of freedom to run test suites, or even specific tests, and provide for custom options in our testing environment. This means we need different levels of testing and different behaviours of the environment. For this reason we developed some intelligent software which is running as a wrapper inside of the Bamboo

server. This software nicely changes the behaviour of the Bamboo CI/CD pipeline to fit our needs.



Schema of basic Bamboo CI/CD pipeline

The main characteristics of this tool are:

- Lots of options to customize the environment (Linux distributions, RDBMS versions, Java versions, testing options....).
- Ability to re-run different tests on one product tarball.
- Ability to switch between development branches and versions (Git version control).
- Test suites management.
- Nicely formatted HTML reports generated at the end.

Test Suites Management

So, when testing options are defined, our Bamboo wrapper/tool creates a shared pool of test suites that should be executed, and also defining their corresponding environment. In the next Stage in the sequence, which is actually created from a number of parallel jobs, each running copy of the Bamboo tool 'takes' the test suite from a shared pool in an intelligent way:

- The first job, which takes a test with some custom Linux distribution and/or RDBMS version and marks itself as a *dedicated job for this environment setup* - this saves time in preparing the same environment for multiple parallel jobs.
- Test suites that fit the current job's setup are taken in order from longest to shortest (according to duration) - this ensures all parallel jobs will have almost the same execution time.

Schema of testing using parallel Bamboo wrapper agents

Levels of Testing

Let's say we don't need to run all our hundreds of test suites after every application code change. It's useless and wastes computing power and time (i.e. when developers make a change in Tungsten Connector API v2, but run all available test suites). We call a limited run "branch testing" (one branch in our Git version control tree represents one new feature, or one fix). For the change made in Tungsten Connector API v2, we need to run only the test suite which tests Tungsten Connector API v2 via branch testing.

In fact, there are options for testing which developers can use wisely to save testing time while ensuring enough testing for coverage of their fix, such as:

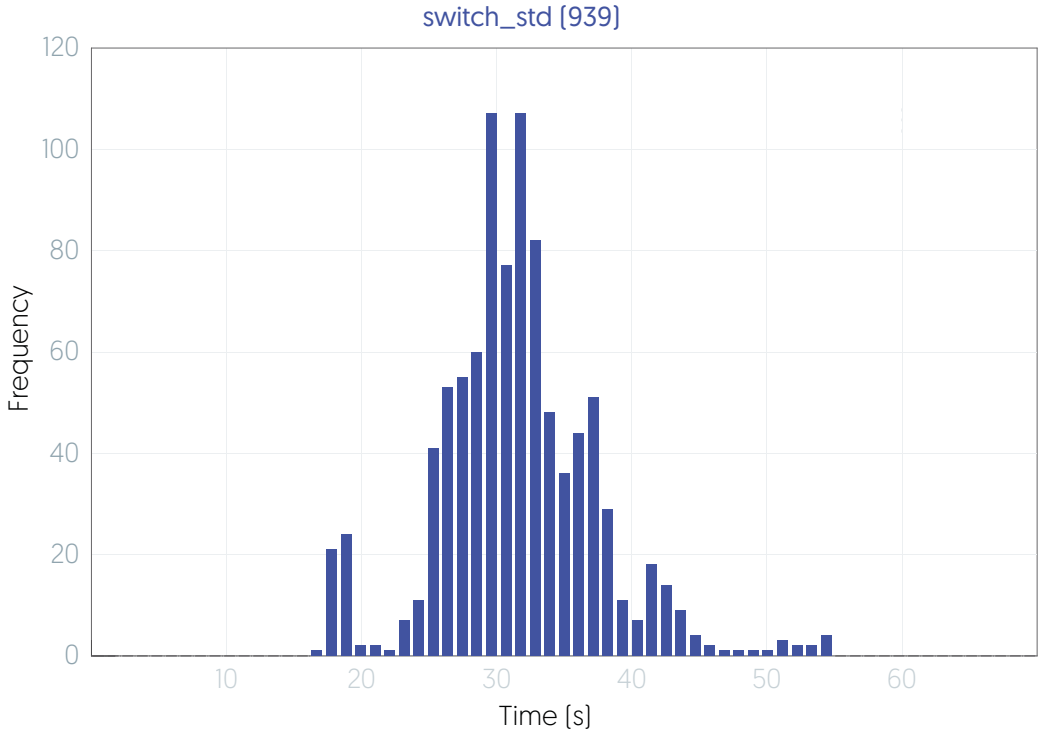
1. Running only a specific test file (test scenario) - there is no need to run the whole test suite with all test scenarios on the same environment setup.
2. Running only one specific test suite - eg. developer wants to run only the test suite for Tungsten Connector API v2.
3. Running a 'family' of test suites - eg. use an option like 'TEST-API' to run all test suites, which is focused on the API v2 (Tungsten Connector, Manager, Replicator, Tungsten Cloud APIs); this way we can also test only a group of test suites focused on, for example, Tungsten Replicator, Tungsten Connector or TPM.

And finally, the true 'levels' of testing (used for regular testing of main version control branches, to final release builds):

1. Minimal acceptance tests - a group of test suites that cover main functionality, but in wide spectrum. This level takes up to one hour of testing time.
2. Running all tests on reference Linux distribution (Amazon Linux 2) and reference MySQL version (MySQL 5.7). As a bonus, there is a *fortune wheel* combination of Linux distribution and MySQL version which is executed (every build has a different combination, in order, not a random pick). Fortune wheel provides smoke testing, to verify that the other Linux distribution and MySQL version works as expected. Testing can also be fun, huh? This level takes up to 8 hours of testing time.
3. Finally, the *release* level of testing. This level takes almost 24 hours to complete, but it tests all available test suites in all possible combinations of environment setup - using the whole environment matrix, as we described in Part 2: Simulating Realistic Conditions. The output of this testing level is a *release candidate* - a fully and deeply tested product tarball, which is provided to our customers.

Watching Performance

For every cluster operation like switch, failover, recovery, etc., and also some "cctrl" commands and queries through the Connector (aka Proxy), we collect the time to execute such actions. From the collected samples of data, we can generate a nice graph as shown below.



Distribution of data samples (duration) for cluster operation (switch)

The graph represents (in an ideal case) a normal distribution of data samples, where we can find an average value (major frequency of samples) and a variance of data samples in both directions. From those data we can estimate what the standard time should be for the operation (i.e. a cluster switch operation) and what time variance is acceptable.

In reports generated at the end of a testing phase, we can easily watch performance statistics, see a possible regression of an action's duration or, on the other hand, any improvement.

Conclusion

We are proud of our comprehensive QA. The fact that Tungsten runs in many different environments and covers a variety of use cases, makes for an extensive and unique testing process alone. Besides this, Tungsten has been used to protect billions of dollars of revenue each year for over a decade, with a 100% rate of customer satisfaction - for MySQL, MariaDB and Percona MySQL applications that are business-critical and mission-critical.

That means it must be durable, strong, and hardened; hence why we call it Tungsten.

Hope you enjoyed learning about the world of QA at Continuent. If you have any questions or feedback, reach out to sales@continuent.com.

About Author

Patrik Michalák

DevOps and QA Engineer

Patrik has been with Continuent for a few years, having previously worked as a full-stack web and mobile application developer. He's a technology enthusiast, and has been awarded at country-level for a photography processing project, and has been involved as an IoT architect in a scooter sharing project in his country in 2019. Patrik is skilled in Perl, Python, JavaScript (ES6), C, Java, SQL, PHP, including technical skills in Linux administration and automation tools.

"Battle-tested" is the Continuent Tungsten QA (Quality Assurance) guarantee. Continuent Tungsten is a clustering and replication solution for MySQL and MariaDB used by some of the largest MySQL estates to achieve continuous MySQL operations, locally and globally (HA, DR and Geo Distribution). Besides the stellar support team and fully-integrated components, customers say: "Stability," and, "Tungsten just works."