# Full Automated Continuous Integration and Testing Infrastructure for Maxscale and MariaDB

Mark Zaslavskiy[1,3], Alexander Kaluzhniy[1,2], Tatyana Berlenko[1,2], Ilfat Kinyaev[1,2], Kirill Krinkin[1,2]
[1]FRUCT LLC
[2]Saint-Petersburg State Electrotechnical University «LETI»
[3]ITMO University
Saint-Petersburg, Russia
mark.zaslavskiy@fruct.org, galiaf1995@gmail.com, tatyana.berlenko@fruct.org, kinyaevilfat@gmail.com, kirill.krinkin@fruct.org

Timofey Turenko
MariaDB Corporation Ab
Finland, Espoo
timofey.turenko@mariadb.com

***Abstract*—This article describes design and development of MariaDb Maxscale robust and easy extendable test automation tool - MariaDB Continuous Integration (MDBCI). For determining optimal architecture a comparison of analogs and a detailed review of existing solution is given. Developed instrument is described and discussed in details in order to determine its extendability level. For proving robustness the comparison on a specific use case between MDBCI and Vagrant as a base of an existing solution was performed.**

## I. Introduction

Nowadays databases technologies are developed with tremendous speed. NoSQL Market Forecast [1] shows that market of only NoSQL products will reach $3.4 Billion in 2020. In this case user expectations and requirements for traditional SQL solutions will grow with similar speed. However, each modern relational database is a very complicated solution that combines high availability features with sophisticated information security and consistency mechanism [2]. Also, there are a number of different use cases and different environment options for each product. This two facts make quality assurance procedure time and resource consuming. Virtualization technologies [3] and configuration management [4] instruments can overcome this problems but such solutions will lead to number of unavoidable adaptations to existing development process due to individual aspects of each product, team and use cases.

This paper aims to provide solution for automating quality assurance and environment management for particular product – Maxscale [5], database proxy for the MariaDb [6]. This solution wraps popular virtualization and configuration management instruments and reorganizes their behavior and configuration languages in order to simplify typical testing use cases of the Maxscale team.

## II. Existing infrastructure

### A. Testing tasks

For understanding necessity and possible improvements of Maxscale test automation infrastructure the first version of it should be described in details. The instrument was created for solving three basic tasks: Build, Run and Test, Upgrade Test. Before describing action sequences it should be clarified that all tasks are performed inside virtual machines (VM) and therefore include VMs start and stop procedures. Due to simplicity the description of these procedures will be omitted.

Build task is aimed to perform quality assurance of the Maxscale packaging and publishing procedures. The task contains of build dependency setup inside of VM, building RPM or DEB packages depending on used OS, collection of packages into debug repositories.

“Run and Test” task main goal is an integration testing of Maxscale packages created during “Build”. The task includes installation and setting up of the given product version and DB cluster with ability to chose particular cluster type: MariaDb/MySql Master/Slave or Galera cluster [7]. Testing is performed by Maxscale-System-Test [8] package which contains integration tests written using CTest [9] framework. After testing is completed test results are gathered for further analysis.

“Upgrade Test” task is actions sequence for integration testing of existing Maxscale installation upgrade. This task differs from the “Run and Test” only by adding additional repo and running package upgrade procedure.

For performing typical tasks described above there were two solutions implemented: libvirt/QEMU-based [10, 11] and AWS-based [12]. Two separate sets of scripts were created.

### B. libvirt/QEMU automation

Test setup is created by set of scripts which executes 'qemu' command line tool directly. Linux distribution have to be installed manually to QEMU machine image. This machine is configured to use hard-coded IP address (e.g. 192.168.121.2). In order to create multi-machine setup following process was implemented:

1) Script copies images file 9 times (one machine for Maxscale, 4 for Master/Slave replication setup, 4 for Galera cluster setup).
2) Script brings up machines one by one and replaces network configuration files to re-configure machines for different IP addresses and the reboot each machine.
3) Script brings up all machines again and configure Master/Slave and Galera.

The process described above has following disadvantages:

- all IP are hardcoded, script is able to create a limited number of VM sets (e.g one uses IP from 192.168.121.110 until 192.168.121.118, another from 192.168.121.120 until 192.168.121.128);
- a number of VM is hardcoded;
- the number of simultaneously running sets of VMs is limited due to limited number of sets of IPs;
- scripts do very limited 'qemu' command error processing (due to complexity of such error processing);
- VMs bringing up process is slow due to need of replacing network configuration files and lack of parallel execution of the process for different VMs;
- solution is not secure: all VM uses one single embedded ssh key.

### *C. AWS automation*

The second set of scripts uses AWS command line interface to bring up 9 VMs in Amazon cloud. Scripts use AWS command line tool to control VMs in the cloud. Scripts are specific for AWS and can not be used for other clouds. Script sends 'start-instances' commands to AWS and then gets information about IPs, SSH keys using 'describe-instances'. This process has disadvantages similar to QEMU case:

- Amazon machine image (AMI), region, VM parameters are hardcoded in the scripts;
- a number of VM is hardcoded;
- scripts do very limited AWS cli error processing and can not automatically re-create failed VMs;
- created VMs have to be managed manually after the test.

The fact that scripts are VM-provider specific makes hard to create sophisticated control tool for VMs: e.g. implement a good error processing, define VM parameters in the convenient and flexible way, keep all information about VMs in centralized way and make such information easily available for clients, etc

## III. Problem statement

According to description of the first version of Maxscale testing system three fundamental requirements can be formulated in order to improve stability and decrease support cost.

The first and the most important requirement is a full automation of a variable VM configuration support. The configuration term stands for the set of machines where each machine can have different parameters: OS type, product type, product version, network configuration. Because of big support cost of several different scripts which control VM work such solution should be replaced by special tool with single entry point.

The second requirement is a usage of a single configuration file format for VM configurations with simple syntax. Existing solution fully relies on embedded constants in Bash scripts and Vagrant files which provide unclear and ambiguous way to configure VMs. Replacement of this mechanism with one domain-specific format will lead to increasing stability of test tool and will decrease its support cost because of more clear and obvious interface for configuration.

The third requirement is a support of automated VM errors processing algorithms. Current solution does not provide any safe way to re-run failed configurations in case of errors which causes need of human interaction with test automation. However, most of errors which disappear after VM restart are related to AWS and therefore following solution with automatic re-run can improve stability of test tasks execution.

## V. Existing instruments comparison

For building replacement for Maxscale test automation existing instruments for configuration management and virtualization should be compared using requirements stated above. This should be done in order to determine architecture of the solution and instruments that can be used inside the solution.

### *A. Criterions*

Before comparison requirements from “Problem statement” subchapter should be decomposed because they are very general and does not allow to compare several instruments without implementing high level solution.

Variable VM configuration support requirement stands for variable number and type of VMs that can be used simultaneously. This can be decomposed to following criterion:

- simple replacement of VM images;
- support of different virtualization providers with ability to change it;
- arbitrary number of VMs;
- support of various products deployment.

Single configuration file with simple syntax requirement can be not related to particular instruments because each instrument use its own configuration format, usually applicable for general tasks. Due to necessity of Maxscale-related use cases support in the solution it is unlikely that there are any instruments already designed for this domain. Therefore, single configuration file requirement can be only satisfied by creation of a custom wrapper for different instruments configuration files formats.

Automated error processing algorithms requirement is related not only to approaches for automatic error handling. For enabling ability to research new and rarely appearing errors in tested products there also should be interfaces for transparent control and debug of launched configurations:

- transparent access to VMs;
- configuration cloning interfaces;
- several test round can be executed in parallel;
- locally tool deploy.

*B. Comparison*

For the comparison the following instruments for VM configurations and provisioning were selected: OpenStack [13], Vagrant [14], Ansible [15], Chef [16], Puppet [17].

OpenStack is a collection of open source software project that developers and cloud computing technologist can use to setup and run their cloud compute and storage infrastructure. Its services are available through Amazon EC2/S3 compatible APIs and hence the client tools written for AWS can also be used with OpenStack. It consist of three core software projects:

- OpenStack Compute Infrastructure also called Nova;
- OpenStack Object Storage Infrastructure also called Swift;
- OpenStack Image Service Infrastructure also called Glance.

Nova is the main part of Infrastructure as a service and it also is the computing Fabric controller for the OpenStack cloud. Enterprises/Organization can use Nova to host and manage their cloud computing systems. Nova manages all the activities that are needed to support life cycle of instances within the Open Stack. Swift offers a distributed,consistent virtual object containers in which lots of data can be store and from which data can be retrieve. It is capable of storing large number of object distributed across nodes. Glance is a lookup and retrieval system for VM images.

Vagrant is used to manage the creation and configuration of VMs for testing environments. Vagrant is a software tool that provides a simple and consistent configuration and command line interface for developers to manage VMs. For each project, a “Vagrantfile” is created within the root directory of the software project and can be included within the source control system. After this, developers can simply run “vagrant up” to have all of the VMs required for local testing made available, with networking and local file synchronization set up automatically.

Ansible is an IT automation tool that provides provision, orchestration, and configuration management features. Unlike with Puppet and Chef, Ansible doesn't require any software to be preinstalled on the server, other than an SSH service, as the heavy lifting is done by own computer that connects to our Ansible-managed servers and instructs the server on how it needs to change.

Chef is a provisioning tool that can be used to set up a server for use for a project. The configuration, which determines how the server needs to be set up, can be stored within our Vagrant project and can be shared with teammates through version control, ensuring that everyone gets an up-to-date copy of the required development environment. Information about how a server should be configured, that is, its software, files, users, and groups, is written into files known as Chef recipes. These recipes are written as Ruby files. Chef takes this information and matches it to providers that are used to execute the configuration on the machine in a compatible way.

Puppet is a provisioning tool that can be used to set up a server for use for a project. The configuration that determines how the server needs to be set up can be stored within our Vagrant project and can be shared with teammates through a version control, ensuring everyone gets an up-to-date copy of the required development environment.

TABLE I. COMPARISON OF EXISTING INSTRUMENTS

| Criteria | Openstack | Vagrant | Ansible | Chef | Puppet |
|---|---|---|---|---|---|
| Different VM providers | - | + | - | + | - |
| Transparent access to VM | + | + | + | + | + |
| Simple replacement of VM images | + | + | - | - | - |
| Cloning | - | + | + | - | - |
| Local tool deploy | - | + | - | + | - |
| Arbitrary number of VMs | + | + | + | + | + |

The results of the comparison by criterion at Table I show that two the most applicable instruments are Vagrant and Chef because they satisfy. Therefore the solution which should replace previous Maxscale test automation must be a single script virtualization provider-agnostic wrapper around these tools.

## VI. PROPOSED SOLUTION

For solving task of MariaDb stack test automation with requirements stated above the MariaDB Continuous Interface (MDBCI) [18] middleware scripts were developed.

*A. Use case*

General MDBCI use case is a test run with different versions of MariaDb stack products and various VMs containing following steps:

1) Developer creates configuration file with configuration parameters: provider, OS type and version, machines counts, RAM, product type and version.
2) Developer downloads needed Vagrant boxes.

3) Developer generates Vagrant config by MDBCI configuration file.
4) Developer launches Vagrant file using MDBCI.
5) After the configuration have successfully launched developer can aquire connection information using MDBCI and execute tests from Maxscale-System-Test repository.
6) For further configuration control MDBCI and Vagrant can be combined.

MDBCI also supports use cases of cloning and creating a snapshot for already launched configurations. Configuration clone is an independent and persistent copy of an existing running configuration. Configuration snapshot is a temporary backup of current state for given configuration which can be restored in any time. Cloning and snapshot use cases have similar action sequences with following components:

1) Developer launches a configuration as in test run use case.
2) Developer starts cloning or create snapshot operation for a running configuration using MDBCI.
3) In case of cloning developer start to use configuration clone.
4) In case of snapshot developer may continue to use configuration with ability to restore its state in any time to snapshot created previously.

*A. Architecture*

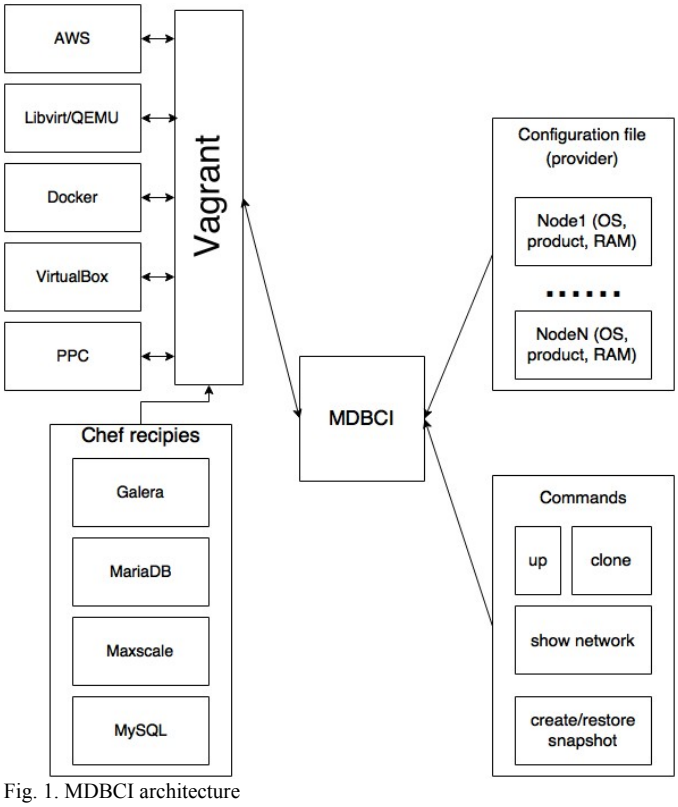

Fig. 1. MDBCI architecture

MDBCI was developed as a set of Ruby command-line non-interactive scripts strictly dependent from Linux environment. Such limitation was approved because target platforms of Maxscale project are only Linux-based OS. Ruby was chosen because it is a base programming language for Vagrant and its plugins and this allows to achieve deep integration with MDBCI sources.

Due to requirements of task and existing instruments comparison MDBCI architecture (Fig. 1) is a wrapper around Vagrant and Chef utilities. The application uses single configuration file as a base for a Vagrantfile generation. MDBCI configuration file format is based on a JSON with simple rules for the cluster nodes declaration:

Nodes declaration

```
0: {
1:   "galera3" :
2:   {
3:     "hostname" : "galera3",
4:     "box" : "ubuntu_trusty_libvirt",
5:     "memory_size" : "1024",
6:     "product" : {
7:       "name": "galera",
8:       "version": "5.5",
9:       "cnf_template" : "galera_server4.cnf",
10:      "cnf_template_path": "~/build-scripts/test-
setup-scripts/cnf"
11:    }
12:  },
13:  "maxscale" :
14:  {
15:    "hostname" : "maxscale",
16:    "box" : "ubuntu_trusty_libvirt",
17:    "product" : {
18:      "name" : "maxscale"
19:    },
20:    "memory_size" : "1024"
21:  }
22: }
```

Generation process also includes substitution of product-dependent Chef recipes into Vagrantfile in order to install and configure each node properly. When Vagrantfile is generated MDBCI can launch it just by wrapping actual “vagrant up” command inside Ruby script. However this wrapping allows to perform continuous Vagrant and Chef error monitoring with ability to non-interactive repair by node relaunch or reprovision [19].

Launched configuration can be managed using MDBCI in following ways:

- Collect network information for establishing ssh connection to nodes.
- Install products.
- Clone configurations.
- Create and restore snapshots.
- Execute commands by SSH.

All actions described above except cloning and snapshots are available for all virtualization providers. For the current moment MDBCI supports VirtualBox [20], Docker, AWS, Libvirt, PPC [21] and any remote machines with SSH access. Cloning and snapshots are unavailable for AWS, VirtualBox

and PPC due to rare usage of this platforms in Maxscale development use cases.

MDBCI was designed to be deeply integrated with Maxscale testing. For simplification of routine actions the solution contains a set of Bash scripts which performs collecting Maxscale-System-Test results. Scripts perform parsing of tests output and store results together with particular test run parameters to special DB in order to do further analysis of regression testing statistics.

MDBCI also contains scripts for performing and analyzing results of Maxscale Sysbench [22] testing. This data contains benchmarking measurements for different performance checks on a various set of Maxscale configuration. Scripts allows to generate special performance test configurations, execute Sysbench on a launched MDBCI VMs cluster, extract results and store it in the same manner as in case of regression testing.

Due to proposed architecture MDBCI has several limitations:

- MDBCI do not have support of plugins.
- Configuration file do not have support for all of machines specifications.
- Chef recipes can be changed or added only in source code which complicates new products support.

These limitations makes MDBCI extension a difficult task which is caused by MDBCI orientation for very narrow set of use-cases and products. However in order to keep required level of integration between MDBCI and Maxscale development process this issue will exist.

## VIII. Evaluation

Robustness and safe error handling are key requirements for MDBCI. In order to prove their existence in developed solution the check on the typical use-case was performed. According to existing Maxscale experience Vagrant may face several not running machines during launch of the big AWS configuration in case of high load on the server. This problem can be simply repaired by performing relaunch of dead machines but Vagrant instead of MDBCI does not contain any tools for such fix. The goal of this experiment was to measure percentage of fails during use-case described below and to measure time needed to fully launch the given VM configuration for both instruments.

The experiment included following steps:

- Launch of test configuration using MDBCI command generate.
- N sequential launches of the generated configuration using “vagrant up” command.
- N sequential launches of the generated configuration using “mdbci up” command.
- Calculation of failed launches percentage for each instrument usage.
- Calculation of launch time in minutes for each instrument usage.

Configuration which was used for measurements contained seven MariaDb nodes, one Galera node and one Maxscale node. The experiment was performed on the Maxscale continuous integration server in order to maximize realism of network load. Steps of experiment were done by several execution of script [25] which also collected results for further analysis.

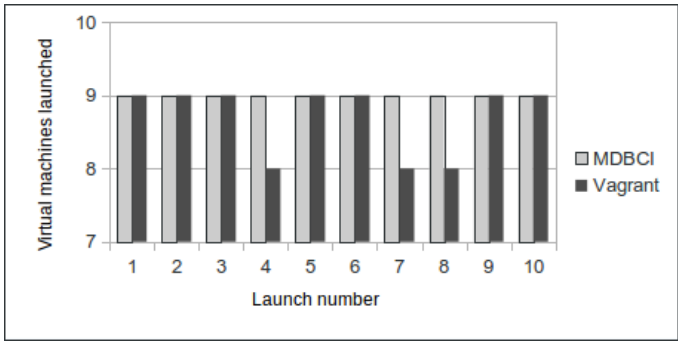


Fig 2. Number of successfully running machines after per test configuration launch

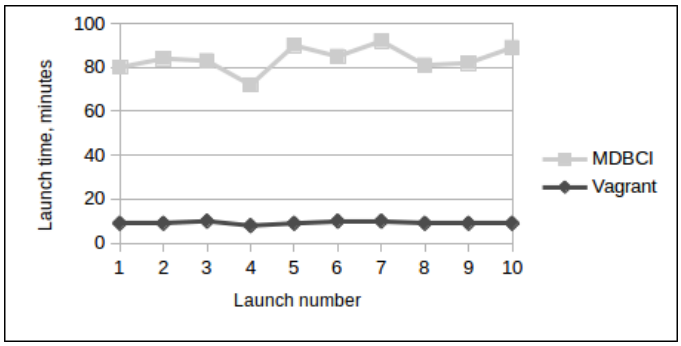


Fig 3. Time of test configuration launches

Fig. 2 shows that MDBCI had successfully launched all machines from test configuration in all cases instead. For the same configuration Vagrant launches contain 3 failed machines which means that MDBCI is a more robust instrument. At the same time MDBCI experienced deceleration of configuration launch process during the experiment. Fig. 3 shows that average MDBCI time equals to 84 minutes which is more than ninefold bigger than Vagrant average launch time for the same configuration. However such speed degradation can be acceptable for most of Maxscale development use-cases due to big duration of test sets - particular tests from Maxscale-System-Test can take several hours to finish.

## VII. Conclusion

Development of the test automation instrument can be a very difficult task in order to support of a specific software product. One of the possible solutions is an integration of the existing low-level tools using a domain dictionary of a particular development process. For the MariaDB product stack the problem was solving by creating MDBCI - a Ruby middle-ware which wraps Vagrant and Chef. The implemented solution allowed to overcome limitations of the existing test

automation: nontransparent debug interfaces, limited support of virtualization providers and high cost of support due to plenty of different configuration file formats.

For proving quality of MDBCI the comparison between imitation of previous test automation (Vagrant) and a current solution was done. Evaluation showed that despite higher level of robustness MDBCI is almost ten times slower that Vagrant based solution, however such slowdown is acceptable because typical Maxscale testing use-cases can take incomparably much time.

Future development of MDBCI includes embedding new virtualization providers and performance benchmarks support in order to provide interfaces to more sophisticated testing of MariaDB stack products. Also, simplification of new provider addition and interfaces for multi provider configurations are planned.

## ACKNOWLEDGMENT

This project was done with financial support of MariaDB Corporation.

## REFERENCES

[1] Market Research Media site, NoSQL Market Forecast 2015-2020, Web: https://www.marketresearchmedia.com/?p=568

[2] C Coronel and S. Morris, *Database systems: design, implementation, & management.* Cengage Learning, 2016.

[3] T. Anderson et al, *Overcoming the Internet impasse through virtualization.* Computer 38.4, 2005, pp. 34-41.

[4] S.S.M. Fauzi, P.L. Bannerman, and M. Staples, *Software Configuration Management in Global Software Development: A Systematic Map.* Asia Pacific Software Engineering Conference. IEEE, 2010.

[5] MariaDB Corporation Ab site, MariaDB MaxScale, web: https://mariadb.com/products/mariadb-maxscale

[6] D. Bartholomew, *MariaDB Cookbook.* Packt Publishing Ltd, 2014.

[7] Galera Cluster for MySQL site, web: http://galeracluster.com/products/

[8] Github repository, Maxscale-System-Test: System level tests for MaxScale, web: https://github.com/mariadb-corporation/maxscale-system-test/

[9] CMake Documentation, ctest(1), web: https://cmake.org/cmake/help/v3.0/manual/ctest.1.html.

[10] M. Bolte et al, *Non-intrusive virtualization management using libvirt. In Proceedings of the Conference on Design, Automation and Test in Europe.* European Design and Automation Association, 2010, pp. 574-579.

[11] C. Guillon, *Program instrumentation with qemu*. 1st International QEMU Users' Forum, 2011, Vol. 1, pp. 15-18.

[12] K.R. Jackson et al, *Performance analysis of high performance computing applications on the amazon web services cloud.* Cloud Computing Technology and Science (CloudCom), 2010, IEEE Second International Conference on 2010, pp. 159-168.

[13] S. Omar, M. Aissaoui, and M. Eleuldj, *OpenStack: toward an open-source solution for cloud computing.* International Journal of Computer Applications 55.3, 2012.

[14] M. Hashimoto, *Vagrant: Up and Running*. O'Reilly Media, Inc., 2013.

[15] J. Keating, *Mastering Ansible*. Packt Publishing Ltd, 2015.

[16] M. Taylor, S. Vargo, *Learning Chef: A Guide to Configuration Management and Automation*, O'Reilly Media, Inc., 2014.

[17] J. Rhett, *Instant Puppet 3 Starter,* Packt Publishing Ltd, 2013.

[18] Github repository, OSLL mdbci: MariaDBCI, web: https://github.com/OSLL/mdbci

[19] A. V. Romero, *VirtualBox 3.1: Beginner's Guide.* Packt Publishing Ltd, 2010.

[20] Biallas S. et al, *PearPC-About*. 2007.

[21] A. Kopytov, *SysBench manual.* MySQL AB, 2012, pp. 2-3

[22] Github repository, OSLL mdbci: MariaDBCI, source code, web: https://github.com/OSLL/mdbci/blob/integration/scripts/paper/check_vagrant_on_heavy_aws_config.sh