When the testing gets tough, the tough get ElasTest

Antonia Bertolino
CNR-ISTI
Pisa, Italy
antonia.bertolino@isti.cnr.it

Antonello Calabró
CNR-ISTI
Pisa, Italy
antonello.calabro@isti.cnr.it

Guglielmo De Angelis
CNR-IASI, and CNR-ISTI
Rome, Italy
guglielmo.deangelis@iasi.cnr.it

Micael Gallego
Universidad Rey Juan Carlos
Madrid, Spain
micael.gallego@urjc.es

Boni García
Universidad Rey Juan Carlos
Madrid, Spain
boni.garcia@urjc.es

Francisco Gortázar
Universidad Rey Juan Carlos
Madrid, Spain
francisco.gortazar@urjc.es

ABSTRACT

We present ElasTest, an open-source generic and extensible platform supporting end-to-end testing of large complex cloud systems, including web, mobile, network and WebRTC applications. ElasTest is developed following a fully transparent and open agile process around which a community of developers, contributors and users is collected. We demonstrate ElasTest in action by testing the FullTeaching application: the video is available from http://elastest.io/videos/icse2018-demo .

CCS CONCEPTS

• **Software and its engineering** → Software testing and debugging; Software notations and tools; • **Computer systems organization** → Cloud computing;

KEYWORDS

Cloud testing, End-to-end testing, Open-source test platform, TaaS, Test automation

ACM Reference Format:
Antonia Bertolino, Antonello Calabró, Guglielmo De Angelis, Micael Gallego, Boni García, and Francisco Gortázar. 2018. When the testing gets tough, the tough get ElasTest. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, Article ICSE-Demo-65, 4 pages. https://doi.org/10.1145/3183440.3183497

1 INTRODUCTION

We assist today to two trends whose convergence motivates the urgent need of new more powerful approaches and tools for software testing. On the one side, software-intensive systems become increasingly pervasive, interconnected and interactive, so that in all business and social activities we are now strongly dependent on their proper functioning. Hence we need effective means to assess that such systems expose adequate levels of dependability and security. On the other side the context in which such systems are immersed and executed may vary widely and dynamically, requiring novel and additional types of testing on top of the standard functional or structural testing techniques: the front end of mobile and web applications need to be tested over many versions of different systems, server-end sides might be running in the cloud and need to be tested for multi-tenancy, elasticity and performance; the application could embed IOT (Internet Of Things) facilities and need to be tested for proper accounting of environment interactions.

In brief, testing modern software systems is increasingly challenging and costly, and more than ever the research needs to develop appropriate tools to help test automation. In particular the end-to-end testing of large complex applications remains a manual and huge effort-intensive activity. In our experience, while many powerful commercial and academic tools have been proposed to address this or that specific testing task, which we call "testing-in-the-small" (TiS), the tester remains largely responsible of properly combining and orchestrating the composition and configuration of the System under Test (SUT) to address global system properties, which we call "testing-in-the-large" (TiL).

The reasons why many IT companies are migrating their applications to the cloud are widely acknowledged in the literature (see, e.g., [9]) and include infrastructure cost reduction with no up-front investments, high scalability and elasticity, fast and easy access to infrastructure. For the same reasons several authors (see, e.g., [7]) have suggested that testing resources themselves could be hosted in the cloud and leveraged according to arising needs.

A recent study [6] investigated the adoption and use of cloud-based testing in practice by conducting a survey over 20 organizations. The findings confirmed that there is growing interest into the opportunities offered by cloud testing tools. However, the authors also conclude that [6] "*many testing-related problems remain unsolved. For example, cloud-based testing widens both manual and automated testing offerings, but it does not offer a generic test automation environment that covers testing needs.*"

We present here ElasTest, a test platform that goes exactly towards the above direction: it is a generic environment for automating the testing process of modern cloud applications and also provides facilities to leverage specific test services as device emulators, browsers, security tools, and monitoring services.

ElasTest is developed with an H2020 European Project (http://elastest.eu/). While a preliminary overview of the project objectives has been given in [4], the project is still ongoing and this demonstration has two aims: we show here for the first time

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3183497

a complete release of ElasTest in action while testing a multimedia e-learning application, and we invite software engineering researchers and practitioners to join the ElasTest community of users and developers to contribute to its next releases.

2 HOW DOES ELASTEST ADVANCE THE STATE-OF-ART?

Due to lack of space we cannot include here an extensive review of the literature on cloud testing, and refer to some existing surveys to explain what is the novelty of ElasTest. In particular, in their overview of cloud testing literature between 2009 and 2012, Incki et al. [5] categorize existing tools along four dimensions according to: test level (namely: Unit, Integration, System and Acceptance) and test type (i.e., functional, performance, security, interoperability), the offered contribution (whether a solution for test execution automation, or test case generation, or a general framework, or an evaluation) and the delivery model (service, platform or infrastructure). They also orthogonally map the body of work onto different problem domains (e.g., Mobile, Cloud, Desktop, Real-time, among others), obtaining a matrix from which existing gaps can be visualized. In particular the study evidences quite few work at integration level and addressing interoperability testing. Thus one first novelty of the ElasTest solution is the focus on *end-to-end testing of large complex cloud applications*, which can include web applications, mobile, WebRTC [1], and so on. Such focus poses many challenges, as end-to-end tests aim at exposing functional and non-functional behaviours at system level to ensure compatibility among:

- all interacting components, and
- all involved infrastructures and tools (possibly including different variants and versions)

With respect to contribution dimension, the core of ElasTest is a solution for test automation all along the test process cycle, including SUT deployment, test execution, SUT monitoring during test run, and test results reporting to testers. Moreover, the intent is to make ElasTest flexible and extensible to also include more tools to cover other testing tasks, e.g. test generation, or test reliability assessment.

Another survey by Bai et al. [1] classifies cloud testing tools according to the main goal they address. Specifically, they distinguish four main tool classes: simulation, service mocking, test job parallelization and environment virtualization. As their survey evidences, there exist several tools that address one or another of the above goals, and the list of available tools has grown further after that survey. A second novelty of ElasTest is the ambition of developing *one solution that can cover all-in-one those four testing goals*: service mocking (including database usage, IoT devices, and also user impersonation) and environment virtualization are already available. Simulation (concerning, e.g., network failures and stress load for scalability testing) and test job parallelization are under way at the time of writing. In addition, it is also planned that ElasTest provides support for *test orchestration*, i.e., for properly combining smaller tests addressing specific components or properties into end-to-end test sequences, and for *test recommendations*, i.e., for advising the testers on what could be the next test cases based on machine learning technology.

Several authors have proposed Testing as a Service (TaaS) solutions in which, essentially, "traditional" testing approaches and tools are migrated to the cloud. A couple of examples, among others, include: Ciortea et al. [3] developed an on-demand software testing service that performs symbolic execution leveraging the cloud to allow for massive parallelization over machine clusters. Tsai et al. [8] propose a framework for combinatorial testing provided as a service on the cloud that enhances scalability by exploiting composition and multitenancy. The main purpose of such TaaS solutions is to reduce the costs and effort from testers in setting and maintaining the needed test infrastructure. One further novelty is the very motivation behind ElasTest: *improving the software testing process as a whole, rather than improving the cost-effectiveness of one specific testing approach.*

In conclusion, ElasTest is a comprehensive framework for deploying and testing a cloud system that offers some own core test services and provides the capability to leverage cloud resources for facilitating functional and non functional testing.

While such aim may seem ambitious, we can rely on a large consortium of partners that are collaborating in developing the basic solution. In addition, as described in Section 5, we are strongly motivated to establish a wide community of users and developers for further enlarging ElasTest components and capabilities.

The project follows an agile process and at time of writing a beta release is available. The platform is being validated on four industrial demonstrators through comparative case studies and quasi-experiments [2].

3 ELASTEST ARCHITECTURE

The ElasTest platform is conceived as a distributed architecture over the cloud that exposes its functionalities through REST APIs (both HTTP, and WebSocket) accessible by developers and testers (e.g. by means of web or command-line interfaces), or by Continuous-Integration frameworks (e.g. by means of dedicated plugins). The design solution that led to the definition of the architecture combined principles for: composing testing units (i.e. TJobs[2]) in order to cover more comprehensive testing objectives; experimenting the SUT in operational conditions close to the actual real-world context; customizing the SUT so to gather relevant information during testing; supporting the recommendation of testing actions enabling interactive decision-taking during the testing phase.

The overall architecture (see Figure 1) distinguishes between a set of core services (i.e., the ElasTest core-platform), and a set of other pluggable services (i.e., test support services) which are either optional, or provide some domain-specific/legacy feature. Among the others, the ElasTest core-platform includes the *ElasTest Platform Manager* (EPM) that abstracts to the ElasTest services the underlying cloud infrastructure where they are actually deployed. In other words, the EPM is the interface that makes the core-platform fully technologically agnostic and enables ElasTest to be deployed and executed seamlessly in several target cloud infrastructures (e.g. Docker, OpenStack, Kubernetes, AWS, etc.).

[1] https://webrtc.org/

[2] TJobs are technologically neutral: the ElasTest platform supports TJobs expressed in any language and using any testing framework.

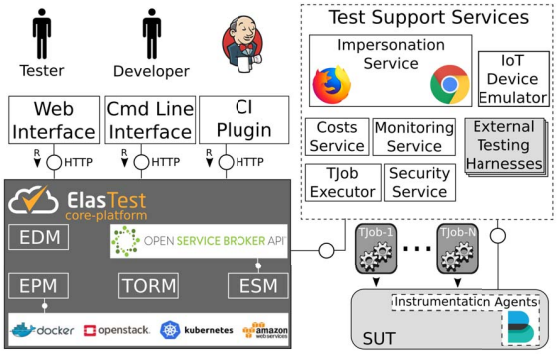

Figure 1: Architecture of the ElasTest Platform

The *ElasTest Data Manager* (EDM) is responsible for installing, managing, and uninstalling the different persistent services available for the rest of the architecture. Specifically the EDM includes features supporting both SQL and NoSQL DBMS, as well as several kinds of elastic distributed file systems; in addition it provides features for query parsing, search indexing, and trend visualization. The EDM gathers various kinds of data collected by the platform: metrics; events at platform level; logs from TJobs executions; files issued by other services, TJobs, or the SUT; and platform management data and metadata.

The *ElasTest Service Manager* (ESM) is responsible for creating instances of supporting services that can be requested by the SUT, or by the platform. Among others, such test support services may concern the proper execution of sets of TJobs, monitoring activities, accounting, or impersonation of actors external to the SUT. In addition, the ESM is also responsible for the engagement of test support services specifically conceived to reuse testing harnesses external to the ElasTest services suite. In the reference implementation the ESM manages the life-cycle of service instances by relying on the Open Service Broker API (OSBA) technology[3].

Both EDM and ESM integrate existing applications leveraging on the containers technology offered by Docker solutions. For example, the ElasTest platform relies on Docker Compose in order to package, and to manage the life-cycle of the supporting services.

Last but not least, the *ElasTest Orchestration and Recommendation Manager* (TORM) is the entry point to all the functionalities offered by the ElasTest platform. It is responsible for guiding the selection and the execution of the TJobs during a testing session, and orchestrating all the above mentioned support services.

In broad terms, the ElasTest architecture does not impose any specific constraint on the SUT: the platform has been conceived in order to target any kind of SUT exporting the only interfaces needed for the execution of the tests. In this sense, the SUT is a system external to ElasTest deployed on some premises. Nevertheless, the TORM also foresees the possibility to directly handle the deployment of a SUT instance over the cloud facilitating effective test deployment and execution. Like in the case of the supporting services, currently the deployment of a SUT leverage on the Docker technology.

[3]see http://www.openservicebrokerapi.org

In the case of an external deployment, typically the SUT has to be injected with specific bundles referred to as the *Instrumentation Agents*; if not the ElasTest platform can only observe/act on the supporting services the SUT is interacting with. However, when the SUT is directly deployed by the TORM, a pre-defined set of Instrumentation Agents are automatically available for the testers. In this case the SUT can be internally monitored, and even controlled from the ElasTest platform unveiling the full potentials of the proposed solution.

Instrumentation Agents enable the ElasTest platform to observe relevant information for testing purposes that are grubbed from within the SUT (e.g. current local configuration, internal status, resources utilization, etc.) during a testing session. In addition, they can also bring controllability for the local environment hosting the SUT. In this sense, the agents can force custom operational conditions, for example by modifying the network parameters, the CPU utilizations, the memories consumptions, or even shutting-down some hosts. From a technological perspective, the reference implementation of the Instrumentation Agents relies on the Elastic Beats[4] data shippers.

A detailed description of all the services in the ElasTest ecosystem is available from: https://github.com/elastest/.

4 HOW CAN TESTERS USE ELASTEST?

This section describes some main use cases that testers can benefit from by working with the ElasTest platform. The context of the presentation is given by the testing of the FullTeaching application [5]. FullTeaching is an open-source educational web-based platform supporting teachers in structuring courses, classes and contents; also it provides means (e.g. calendars, dashboards, forums, etc.) for engaging and interacting with students.

From the testing perspective, FullTeaching is a complex software application involving several components; among the others the core of the web application (i.e. the FullTeaching App Server), the OpenVidu Server [6], the Kurento Media Server [7], and the MySQL DBMS. Each component can be deployed in a distributed manner, and configured according to various independent settings: different versions of each component can be selected and combined, possibly with different results to be validated. In this setting, the adoption of several datasets (e.g., the data retrieved from the persistence layer of FullTeaching by querying a specific MySQL instance) can contribute to spot possible unexpected behaviours that are data-dependent. In addition, as FullTeaching includes features enabling real-time video conferencing, testers should be able to systematically master the conditions of the underlying infrastructure hosting the application but external to the specific SUT instance (e.g., assumed level of latency or packet loss ratio in the network, configuration of firewalls, number of concurrent clients), as they can hardly impact on functional and non-functional aspects.

Testing a structured application such as FullTeaching challenges testers to validate the behaviour exposed by the SUT when interacting with different 3rd party clients (e.g., web browsers) or services (e.g., calendars). Hence a test plan should identify the sets of 3rd

[4]see https://www.elastic.co/products/beats
[5]see https://github.com/elastest/full-teaching
[6]see http://openvidu.io/
[7]see http://www.kurento.org/

party software to be targeted. The ElasTest platform facilitates such task by providing: an abstraction that supports the specification of a variety of 3rd party software to be bound during the tests; and a seamless orchestration of testing instances for such external software aiming at reproducing some actual execution context for the SUT (e.g. users impersonation through web browsers).

Also, testers adopting the ElasTest platform can monitor and analyse the results from TiL sessions. The ElasTest platform observes and logs events emitted during the executions; it collects elastic log data from many distributed sources enabling the aggregation of large volumes of events. The analysis and the visualization of these data aim to support testers to ultimately detect whether the observed sequence, or some complex sequence of interest contains any fault. The inspection approach advocated by the ElasTest platform is agnostic to both the architecture, and the technology of the SUT. Specifically it exploits canonical techniques for TiS in order to dynamically build (i.e. create and instantiate) monitors producing simple or aggregated event sequences to be inspected. The dynamic orchestration of monitors enables to narrow or widen the focus of the analysis according to the current needs/goals.

The rest of the section details the demonstration of the use cases described above within the context of the FullTeaching application.

4.1 Configuration and Deployment

The ElasTest platform may expolit some set-up over the SUT (see Section 3). For this demo, the FullTeaching application has been split in four parts (namely FullTeaching App Server, OpenVidu Server, Kurento Media Server, and MySQL); each part has been virtualised in independent Docker containers. The demo shows the configuration for the multi-containers instantiation of FullTeaching by means of Docker-Compose technology. The setup includes the specification of a configuration parameter enabling the selection of the versions for the different parts in the SUT.

In this setting, the demo shows how the ElasTest platform will be able to deploy the FullTeaching application under different environments/configurations. As a result the testers can exercise the SUT in several configurations, and are thus enabled to retrieve and compare execution traces from different setups (see Section 4.3).

4.2 Impersonation

The demo presents how the ElasTest platform starts several browser instances that are testing the video-conferencing feature of FullTeaching. Specifically, the tests will cover one-to-one video and audio real-time streaming using the Chrome web browser.

After the deployment of the SUT according to a desired configuration (see Section 4.1), and while the ElasTest platform waits that the FullTeaching App server is available on the port 8080, the demo shows the structure and the configurations for an impersonation test to be run. In this specific case, the tests are codified in JUnit and their implementations refer to the Selenium technology.

Back to the ElasTest platform, the demo presents how the TJob Executor loads and runs the specific test instances (i.e., TJob-i); samples of the interactions between the SUT and the impersonated users are shown during the tests executions. When the TJob Executor has run all the prescribed TJobs, the ESM stops the test support services involved in the scenario.

4.3 Test Results and Analytics

Both during and after the execution of one or more TJobs, the testers consult the ElasTest management dashboard in order to access the logs from the tests executions and the metrics values revealed by the Instrumentation Agents activated over all the several components of the SUT. Specifically, the demo focuses on test cases codified with JUnit, thus the output of the Maven executions have been stored in the platform and are shown by the demo. Also the ElasTest platform handles the log events proper of the whole FullTeaching application, thus testers can query and analyse the information they contain. About metrics (e.g., `cpu_totalUsage`, `mem_maxUsage`, or `net_txErrors`, etc.), the trends of measured values over the whole execution of the TJobs are stashed away and shown. Finally, as the demo concerns emulation of users acting via web browsers, the specific test support service (i.e., Impersonation Service) stores in the core-platform a video of the interactions referred in Section 4.2; this way testers can validate the actual behaviour of FullTeaching experienced by realistic clients during the tests.

5 CONTRIBUTING TO ELASTEST

The ElasTest platform is being developed following a completely open and transparent agile approach. The community web site (see http://elastest.io), reports the project roadmap, and also up-to-date installation instructions, user documentation, developer procedures, and histories from users.

It is our ambition to create a large and active community of both users and contributors around ElasTest. Not only we hope to attract developers of the core platform, but we also welcome very much the opportunity of enlarging the test services offered by ElasTest by embedding ("containering") other tools.

ACKNOWLEDGMENTS

This paper describes work undertaken in the context of the European Project H2020 731535: ElasTest.

REFERENCES

[1] Xiaoying Bai, Muyang Li, Bin Chen, Wei-Tek Tsai, and Jerry Gao. 2011. Cloud testing tools. In *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*. IEEE, 1–12.

[2] Antonia Bertolino and Eda Marchetti (Ed.s). 2018. ElasTest validation methodology and its results, Version 1. http://elastest.eu/deliverables.html. (2018). (Public Deliverable, in preparation).

[3] Liviu Ciortea, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, and George Candea. 2010. Cloud9: A software testing service. *ACM SIGOPS Operating Systems Review* 43, 4 (2010), 5–10.

[4] Francisco Gortazár, Micael Gallego, Boni García, Giuseppe Antonio Carella, Michael Pauls, and Ilie-Daniel Gheorghe-Pop. 2017. ElasTest - An Open Source Project for Testing Distributed Applications with Failure Injection. In *Conference on Network Function Virtualization and Software Defined Networks*. IEEE.

[5] Koray Incki, Ismail Ari, and Hasan Sözer. 2012. A survey of software testing in the cloud. In *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*. IEEE, 18–23.

[6] Leah Riungu-Kalliosaari, Ossi Taipale, Kari Smolander, and Ita Richardson. 2016. Adoption and use of cloud-based testing in practice. *Software Quality Journal* 24, 2 (2016), 337–364.

[7] Scott Tilley and Tauhida Parveen. 2013. *Software Testing in the Cloud: Perspectives on an Emerging Discipline*. IGI Global.

[8] Wei Tek Tsai, Guanqiu Qi, Lian Yu, and Jerry Gao. 2014. Taas (testing-as-a-service) design for combinatorial testing. In *Software Security and Reliability, 2014 Eighth International Conference on*. IEEE, 127–136.

[9] Qi Zhang, Lu Cheng, and Raouf Boutaba. 2010. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications* 1, 1 (2010), 7–18.