The ElasTest Platform: Supporting Automation of End-to-End Testing of Large Complex Applications

November 2018

A white paper by the ElasTest team

Authors: Francisco Gortázar, Micael Gallego, Malena Donato, Enric Pages, Andy Edmonds, Guiomar Tuñón, Antonia Bertolino, Guglielmo De Angelis, Anton Cervantes, Thomas Michael Bohnert, Alexander Willner, Varun Gowtham.

1. ElasTest End-to-End Testing

Quality and high speed is the new mantra: everyone wants the best products delivered as fast as possible. On the one side, managers aim at "*the fastest time to market*"; on the opposite side, the cliché recurs that "*a good user experience is the key to a successful product*". Compounding both sides, developers know well that before they can release their software, *testing is a must*. Notwithstanding, it is often ignored or given low priority, why? Are the integrated systems tested enough at their ends to meet the user-demanded functionalities? Software code is written, transformed, and updated, then it is checked-in and verified before a new product is finally launched. But not always this translates into the best software solution or the best experience for users. Is it all about continuous integration process or are there more reasons? With the increasing need of distributed and more interconnected software systems, are developers ready to satisfy this demand?

To answer these and even more questions (continue reading for a longer, and yet incomplete, list of difficult questions faced by testers and developers of distributed systems), a diverse group of researchers and industry practitioners, with a common vision, have been participating for the past year and a half in the ElasTest project. This white paper provides an overview of the project and its goals, while providing answers to industrial and research challenges.

2. The Challenge of Large Scale Distributed System Testing

Many current software engineering practices are pushing towards building systems in a distributed manner thanks to new easily available capabilities such as microservices, cloud-native applications, big data applications, edge computing. The demand for larger and more interconnected software systems is constantly increasing, but the ability of developers to satisfy it is not evolving accordingly. ElasTest postulates that *the most limiting factor is software validation*, which typically requires very costly and complex testing processes.

When testing distributed systems, many challenges and questions are faced by testers and developers, such as:

- How can we deploy the system to be tested, otherwise known as the System Under Test (SuT)? Where is it to be deployed? How do we know if the deployment was successful and, if so, whether the SUT is ready to be exercised by the tests?
- How can we get information from the SuT? How can we get a trace of logs from the different tested services? How can we get metrics (*e.g.*, CPU usage, memory, Input/Output on-disk operations) of the different services to know how are they performing?
- How do we search logs in the presence of test failures? How do we make a connection from log lines in a service to log lines in a different one? And, how can we relate them?
- How can we know if a change in one service impacts the overall performance of the application or of other services?
- How do we emulate user interactions? Do we need a browser? If so, which version(s)? How can we use several browsers at one time to assess the concurrent behaviour of multiple sessions? How do we distribute those browsers among different machines to avoid bottlenecks unrelated to the SUT?
- How we do emulate interactions in system-to-system communications or Internet-of-Things (IoT) applications? In particular, how can we emulate sensors or actuators? How can we assess an action over an actuator?
- How can we evaluate the limits of the current application architecture in terms of concurrent sessions supported, response times, and so on? How can we simulate the load for the SUT?
- How can we know if the application will still work in the presence of failures? How do we make some parts of it fail, as for instance tearing down nodes, or blocking network interfaces?
- Does the SUT expose known vulnerabilities and how can we test for them?
- How can we perform some analytics over the terabytes of logs of the different services and look for patterns that indicate bottlenecks or any other kind of problem?
- Can we make a long running test to fail, if based on evidence from logs and/or metrics it is clear that it won't succeed? If so, how can we do that?
- When performing manual testing, how can we automate certain clerical steps? How do we save the context when a failure is detected? How do we record the browser

session and interactions? And, how do we get my browser logs or those of any of the services?

The above is a list of questions for testers and developers that are particularly difficult to answer when testing large complex distributed systems, and it could be continued.

Hence, more than ever new powerful tools and approaches are needed to match new capabilities and, importantly, help in testing and validation. In particular, *end-to-end testing* of large complex applications remains a manual and very effort-intensive activity. For example, a recent survey investigating the testing practice in cloud systems over 20 organizations concluded that "*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*" [8]. Indeed, many commercial and academic tools have been proposed to address specific testing tasks [6] [9] - what we call "testing-in-the-small" (TiS). However, the tester remains largely responsible of properly combining and orchestrating the composition and the configurations of the TiS test cases to address global system properties - what we call "testing-in-the-large" (TiL).

Cloud testing tools have been classified according to the main goal they address, specifically, simulation, service mocking, test job parallelization and environment virtualization. While there exist several frameworks and tools that address one or another of the above goals, the ambition of ElasTest is developing a comprehensive all-in-one solution that can cover those four testing goals [7]. In fact, concerning the above goals:

- Service mocking (including database usage, IoT devices, and also user impersonation) and environment virtualization are already available in ElasTest.
- Simulation (concerning, e.g., network failures and stress load for scalability testing) and test job parallelization are under development (planned for the beginning of 2019).
- Test orchestration, i.e., for properly combining smaller tests addressing specific components or properties into end-to-end test sequences, is in the roadmap.
- Test recommendation service, i.e., for advising the testers on what could be the next test cases based on machine learning technology, is also in the roadmap.

3. Can We Help You?

The activity of testing distributed systems lacks tools for the following tasks:

- System Under Test (SuT) deployment. In monolithic applications, usually the test framework is able to start the system and perform tests against it. Distributed applications cannot be deployed easily and likely require several services to be started in a specific order. Usually these systems, following infrastructure-as-code good practices, come with deployment descriptors and/or scripts. If your system is docker-based, a docker-compose file could be such a descriptor. If using AWS, a cloud-formation template might be available. Therefore, many things teams would like to test or many properties they would like to validate become really hard to assess.

- Testing services. 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 facilities and need to be tested for proper accounting of environment interactions. Therefore, test support services providing scalable testing capabilities to the tester are mandatory.
- Root cause analysis. Given the vast amount of information that could be captured from a testing session of a distributed system, novel visualization approaches are needed. Specifically, in the presence of failures, the relevant information should be presented to the user, who should be able to browse the information gathered from the system in an easy way until a root cause for the failure is identified.

ElasTest platform is conceived as an open source cloud-based testing platform aimed to simplify the end-to-end testing processes for different types of applications, including web, mobile, real-time video communications and Internet-of-Things. Such a platform co-exists within the software lifecycle process integrated with other tools, and taking care of the three tasks depicted above. ElasTest provides facilities to leverage specific test services as device emulators, browsers, security tools, and monitoring probes, and operates in different environments. ElasTest provides advance testing capabilities aimed to increase the scalability, robustness, security and quality of experience of large distributed systems, and new capabilities are on under development. All in one, ElasTest will make any software development team capable of delivering software faster and with fewer defects.

The core of ElasTest platform 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, such as test generation, or test reliability assessment.

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) that either are 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.).

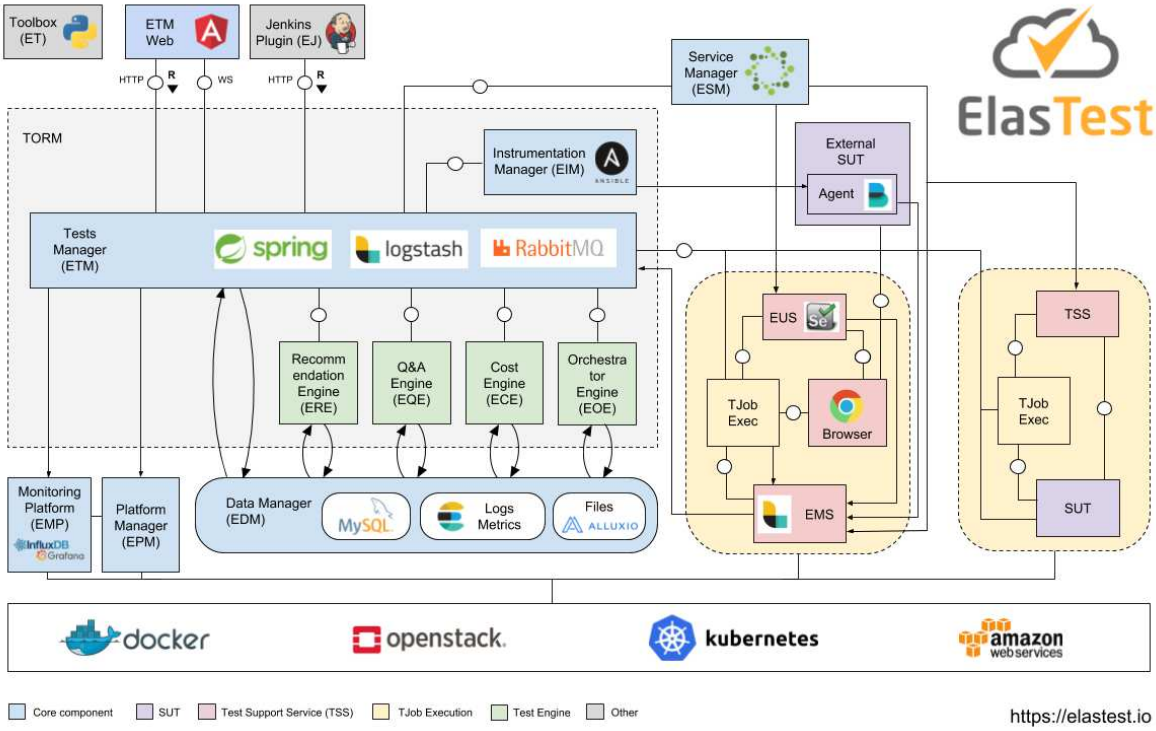

Figure 1: ElasTest Architecture

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 that is deployed on some premises. Nevertheless, the TORM also foresees the possibility to directly handle the deployment of a SuT instance over the cloud, thus facilitating effective test deployment and execution. Like in the case of the supporting services, currently the deployment of a SuT leverages the Docker and OpenStack technologies.

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 predefined set of Instrumentation Agents is automatically available to the testers. In this case the SuT can be internally monitored, and even controlled from the ElasTest platform, thus 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. A detailed

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

4. How to Use ElasTest

ElasTest fits perfectly within standard continuous integration infrastructures, thanks to its integration with external tools like Jenkins and TestLink. Such integration has been built with the idea that whenever any teams need testing, they could readily use ElasTest and its services. In addition to these integrations, ElasTest provides a rich Web user interface, focused on the three tasks depicted in the previous section: SuT and test definition, test execution, and root cause analysis.

When the Jenkins integration is used, the user has total freedom on how to start and stop the application. Additionally, ElasTest will collect information about the system as logs and metrics, will provide services like browsers on demand, and will allow the user to visualize all the information of the testing session through the appropriate tools from the ElasTest UI.

If ElasTest is used for controlling the lifecycle of the application, then the user just provides a deployment descriptor, and everything happens under ElasTest control. The SuT in this case could even be run on top of the ElasTest platform itself (instead of running on Jenkins).

Whatever way teams choose to use ElasTest, they will be presented with very relevant services for the testing task at hand. From browsers to IoT device emulators, including Big Data analysis tools, and monitoring probes, the limit as to what can be accomplished in an end-to-end test is just the imagination, or the needs of business.

Troubleshooting problems in the application detected by test failures becomes a much simpler task with the aid of tools like the Log Analyzer. Logs should not be inspected as plain text, nor individually. Rather, distributed system logs should be presented together, so that the context of the whole operation across different services can be visualized. On top of this log presentation, several features for enabling pattern search and highlights have been built. It is now really simple to mark all the log entries containing a specific pattern in a different color for easy identification, without hiding any other log lines, as we show in Figure 2.

Figure 2: ElasTest Log Visualization

5. ElasTest Demonstrators

To verify whether ElasTest's ambitious goals are achieved and to monitor progress, ElasTest has set precise validation metrics on both the efficiency and the effectiveness of the testing process. The key metrics target time-to-market, tester's productivity and satisfaction, maintenance effort and QoE. ElasTest will assess these and several other Key-Performance Indicators in several iterations over four vertical demonstrators. All the assessments will be made publically available to guide ElasTest adopters.

The four ElasTest demonstrators cover different application domains: 5G networks, web applications, WebRTC, and IoT.

- The 5G Networks Vertical Demonstrator is driven by the research institute Fraunhofer FOKUS. It addresses the challenge of deploying innovative services in the telecommunication domain. The expected KPIs for 5G networks include ultra-low latency, high availability, high capacity and ultra-reliability. The Fraunhofer FOKUS Open5GCore toolkit[1] is an implementation of the carrier-grade network towards the 5G environment and its management. It mirrors, in a prototypical form, the pre-standard advancements on the core network, radio network integration, distributed management and virtualization. Its aim is to accelerate research and development as well as knowledge transfer towards partners. A core network consists of multiple components with specific functionalities, therefore a testing

[1] https://www.open5gcore.org/

platform able to facilitate complex SuT deployments as well as end-to-end performance tests is required. By making use of the Elastest SuT deployment, monitoring and test orchestration features, the task of analyzing the performance of the Open5GCore network deployment becomes straightforward and uncomplicated. This allows the development efforts to remain focused on 5G core network features while the 5G KPIs monitoring for the targeted use-cases is handled in multiple deployment scenarios using the ElasTest Platform.

- The Web Vertical Demonstrator is focused on using ElasTest in an E-commerce platform and is provided by Atos Worldline. One of the most important aspects for an E-commerce client is to be sure the platform is working correctly and -as expected- from all kind of browsers and versions of those browsers. With ElasTest it is possible to do this without any additional installation. At the same time ElasTest allows the Quality Analyst (QA) to test the application manually or automatically leveraging (e.g., on Selenium[2]) the most widespread technology for automated testing. Once ElasTest has been installed and configured, visualizing different kind of metrics of the server, like memory consumption or CPU usage, is really straightforward for the QA. Saving logs from all executions, having all the execution information correctly saved for further analysis for the near future is done automatically and makes different deployments and result comparison really easy.

- FullTeaching is an open-source application to make online classes easy for teachers as well as students. It supports multiple communication channels between teachers and students (forums, chats, and real-time video sessions) and is provided as a WebRTC demonstrator for the ElasTest project by Naeva Tec. “Traditional testing” for this kind of web application that includes real-time WebRTC communications is difficult and won’t provide a completely tested system. We need to be able to test what the user is experiencing during the communication session, not only how the system and the application behaves under specific circumstances. We call this Quality of Experience (QoE) testing. QoE is defined by the International Telecommunication Union (ITU) as “The overall acceptability of an application or service, as perceived subjectively by the end-user” [5]. It is a quite abstract definition, but the key is that QoE is subjective to the user and this is the main challenge for automatic QoE testing. The approach we follow, when testing QoE in WebRTC is defining some objective parameters such as transmission rate, delay, jitter, bit error rate, packet loss rate, etc. and assigning them some thresholds by which we consider the communication successful. Currently these parameters are not easy to measure on automatic tests and nearly impossible to use as acceptance metrics. ElasTest is improving the capability to take these measures and is expected to allow us to use them as acceptance metrics on QoE tests. This will make a big difference and bring to the table a whole new set of possibilities for testing QoE on WebRTC communications. It will also give us the ability to increase the overall quality of this kind of applications while reducing time to market.

[2] https://www.seleniumhq.org/

- The IoT vertical from TUB (Technische Universitaet Berlin) demonstrates the rapid prototyping and testing of IoT applications. IoT applications are built upon the concept of using data from one or more sensors, collectively applying a logic to make a decision and based on the decision signalling an actuator to take action. In the real world, sensors sample physical quantities such as temperature, humidity, and pressure to name a few, while actuators represent a class of devices that receive the order to start or stop or continue operating on other devices. For example, in an application monitoring the temperature of a room, a temperature sensor must be connected to the application via appropriate electronics. Suppose the logic in application is to check whether the temperature is too hot. The sensor data is compared against a threshold, if it is above the threshold, an actuator is signalled, and in its turn the actuator signals an alarm. Composing an IoT application in real world requires the use of real sensors and actuators. Testing IoT applications thus requires the capability to alter operations and conditions of the IoT application and thereby to verify the robustness of application. IoT applications may need to be replicated and scaled up to increase their coverage as a user wants. However, building and testing scalable applications can be expensive and time consuming because of the various physical parts and connections that need to be taken care. ElasTest overcomes the challenge of building and testing scalable IoT applications. The ElasTest Device Emulator Service (EDS) provides emulated sensors and emulated actuators, which imitate their real world counterparts. ElasTest addresses the challenges in scaling up IoT testing by using the emulated sensors and actuators provided by EDS to build and rapidly prototype complex scalable applications. Users can ask for multiple emulated sensors and actuators from the EDS. Then, they wire these in a desired configuration and run them. Thanks to ElasTest, the IoT application can be deployed as a SuT and tested against with test jobs. The tester is capable of injecting faults or test conditions in a running SuT, while the EDS can alter the behavior of any emulated sensor or actuator.

Some preliminary results from project experimentation on the above four demonstrators are already available in the project web site [10].

6. ElasTest at a Glance

Today end-to-end testing of large distributed applications remains effort-prone and mostly manual. ElasTest aims at improving the efficiency and effectiveness of the testing process and ultimately improve the quality of large software systems in the Cloud. ElasTest offers a flexible open source testing platform for rapid and accurate end-to-end testing. ElasTest will reduce the time-to-market of software projects, increasing the quality of the resulting software product, reducing the possibility of failures. It will improve the perception of the software both by the end user and the developers who can perform more complex testing in less time. ElasTest will be demonstrated on four different types of applications, including web, 5G mobile, real-time video communications, and Internet-of-Things.

The ElasTest project is carried out by an European consortium composed of the following partners and institutions. The project is led by the Spanish University Rey Juan Carlos and involves the research Centre Consiglio Nazionale delle Ricerche, the Technische Universität Berlin, Zürich University for Applied Sciences and IMDEA Software Institute. It counts with the participation of different institutions from industrial domain such as Atos Spain & Atos Worldline, Fraunhofer, IBM, NAEVATEC and Relational.

The ElasTest Community supports and fosters the interactions between the core ElasTest members and the community. Anyone can join the ElasTest Community to connect and interact for making ElasTest a reality. The ElasTest community welcomes anyone who want to participate, we expect contributors to the Open Source Community working on the ElasTest platform to make the unique e2e platform!

To test ElasTest one can access this link: https://elastest.io/docs/
Websites are: http://elastest.io for software releases & http://elastest.eu/ for project and research information.

Three channels are available to talk in real time with us to ask questions, and solve doubts.

- Google Group: https://groups.google.com/forum/#!forum/elastest-users for any doubt about ElasTest features or how to use it.
- GitHub issues: https://github.com/elastest/elastest/issues
- Stack Overflow https://stackoverflow.com/questions/tagged/elastest to get support from the community join us and contribute to ElasTest!

Also, keep in touch with us through ElasTest social media:

- Medium blog! https://medium.com/@elastest
- Twitter is @elastestio

The project started in January 2017 and lasts until December 2019.
The project belongs to Horizon 2020 program, within the ICT-10 "Software Technologies" topic, under grant Agreement nº 731535 The total EU funding is 5 M€. The authors alone are responsible for the content.

References

1. ElasTest project Description of Action (DoA) – part B.
2. Bertolino, A., 2007, May. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering* (pp. 85-103). IEEE Computer Society.
3. Apache 2.0 license terms. https://www.apache.org/licenses/LICENSE-2.0. Accessed on 07 March 2017.

4. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION. Communications Networks, Content and Technology. 11 November 2016.
5. ITU-T. Vocabulary for performance, quality of service and quality of experience. Recommendation ITU-T P.10/G.10. 2017.
6. Xiaoying Bai, Muyang Li, Bin Chen, Wei-Tek Tsai, and Jerry Gao. 2011. Cloud testing tools. In IEEE Proc. 6th International Symposium on Service Oriented System Engineering (SOSE), 2011. IEEE, 1–12.
7. 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 Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings (ICSE '18). ACM, New York, NY, USA, 17-20.
8. 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.
9. ElasTest Consortium. D2.2 SotA revision document v1. 2018.
10. ElasTest Consortium. D7.1 ElasTest Validation methodology and its results v1. 2018.

Glossary

AWS - Amazon Web Services
EDES - ElasTest Device Emulator Service
E2E - End to end
EPM - ElasTest Platform Manager (EPM)
ITU - International Telecommunication Union
IoT - Internet of Things
KPI - Key performance indicators
QA - Quality Analyst
QoE Quality of Experience
QoT Quality of Testing
RTC - Real-Time Communications
SuT - System under test
TiS Testing in the small
TiL Testing in the Large
TORM - Test Orchestration and Recommendation Manager
UI - User interface