



Funded by the
European Union

A Proposal to Orchestrate Test Cases

Boni García

boni.garcia@urjc.es

<http://elastest.io>

11th International Conference on the Quality of Information
and Communications Technology (QUATIC 2018)
September 5th 2018 | Coimbra, Portugal

Table of contents

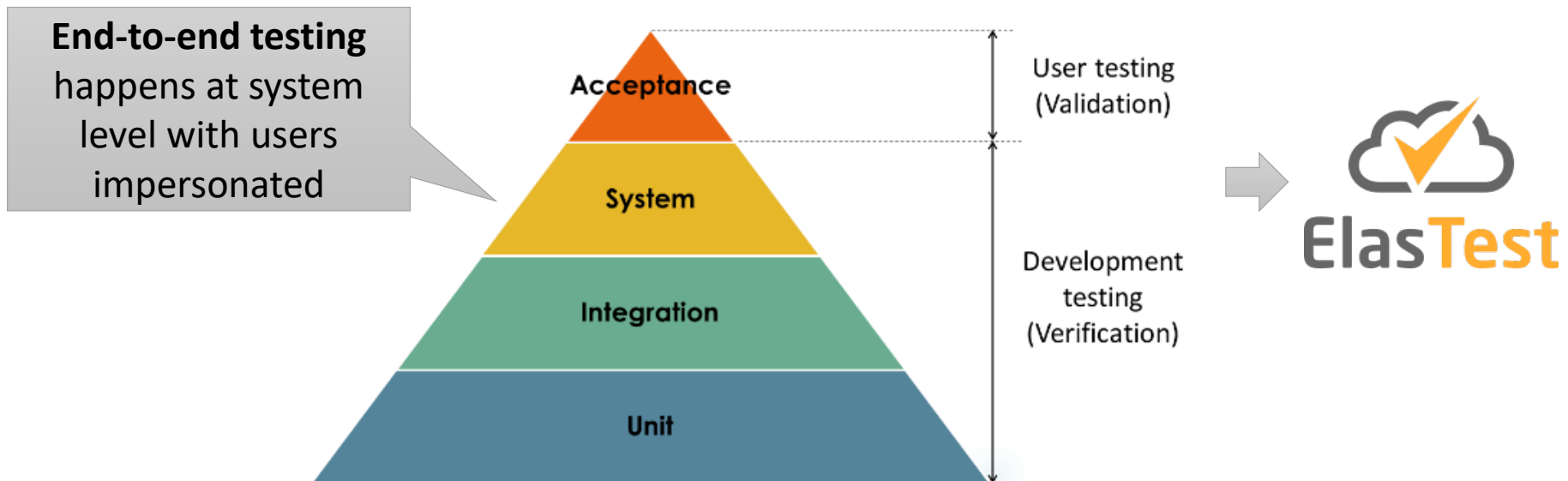


1. Introduction
2. Test orchestration approaches
3. ElasTest: platform for end-to-end testing
4. ElasTest Orchestration Engine
5. Case study: testing WebRTC applications
6. Conclusions and future work

1. Introduction



- Large distributed heterogeneous systems are more and more common (e.g. microservices architectures, cloud native apps, etc.)
- Testing this kind of software is complex, especially to verify the system as a whole



1. Introduction



- In ElasTest, we hypothesize that test cases can be organized to create a complex test scenarios (“divide and conquer” principle)
- We understand the concept of **test orchestration** as a novel way to select, order, and execute a group of tests
- To that aim, different steps are considered:
 1. Topology generation: to define a graph of tests
 2. Test augmentation: to reproduce custom operational conditions of the SUT

Table of contents



1. Introduction
- 2. Test orchestration approaches**
3. ElasTest: platform for end-to-end testing
4. ElasTest Orchestration Engine
5. Case study: testing WebRTC applications
6. Conclusions and future work

2. Test orchestration approaches



- In this stage of the project we have focused in the topology generation problem
- We propose two different approaches for orchestrating tests:
 1. **Verdict-driven orchestration**, i.e. connecting testing jobs (TJobs) using its verdict (i.e., passed or failed) as boolean condition
 2. **Data-driven orchestration**, i.e. connecting TJobs using the test data (input) and test outcomes (output) handled internally by tests

2. Test orchestration approaches



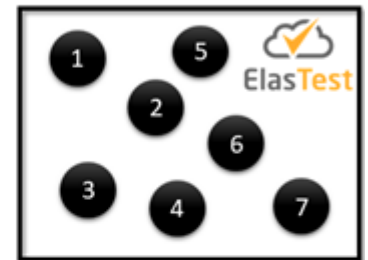
1. *Verdict-driven orchestration*

- TJobs are considered black-boxes, since we only know its result (verdict)
- All TJobs are executed without a given ordering
- Using a topology notation, we can select and order a group of TJobs
- We can also execute in parallel a group of TJobs

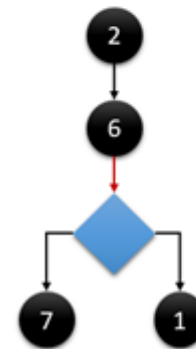
a) *Black-box* T-Job



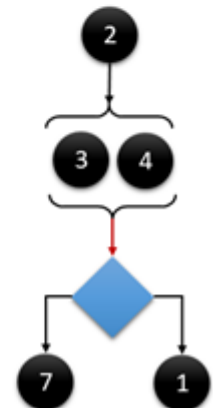
b) T-Jobs in ElasTest



c) *Verdict-driven* test orchestration



d) *Verdict-driven* test orchestration with parallelization



2. Test orchestration approaches



2. *Data-driven orchestration*

- In addition to the verdict, in this schema we consider the input and output data in the TJobs
- This kind of TJobs coexists with the previous type (black-box)
- We use the same topology notation to select and order a group of TJobs
- The output data of TJobs is used to feed next level in the graph

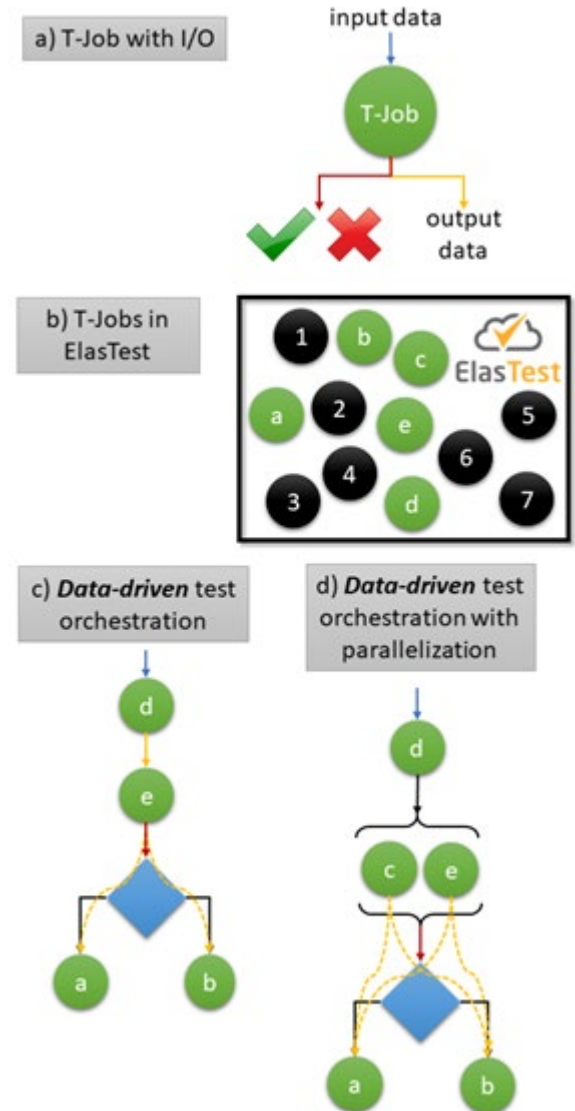


Table of contents



1. Introduction
2. Test orchestration approaches
- 3. ElasTest: platform for end-to-end testing**
4. ElasTest Orchestration Engine
5. Case study: testing WebRTC applications
6. Conclusions and future work

3. ElasTest: platform for end-to-end testing



- ElasTest is an open source platform aimed to ease the end-to-end testing activities for different types of distributed applications and services
- ElasTest manages the full testing lifecycle, deploying and monitoring the SUT, executing the end-to-end tests and exposing the results to software engineers and testers



<http://elastest.io/>

3. ElasTest: platform for end-to-end testing



- ElasTest architecture:

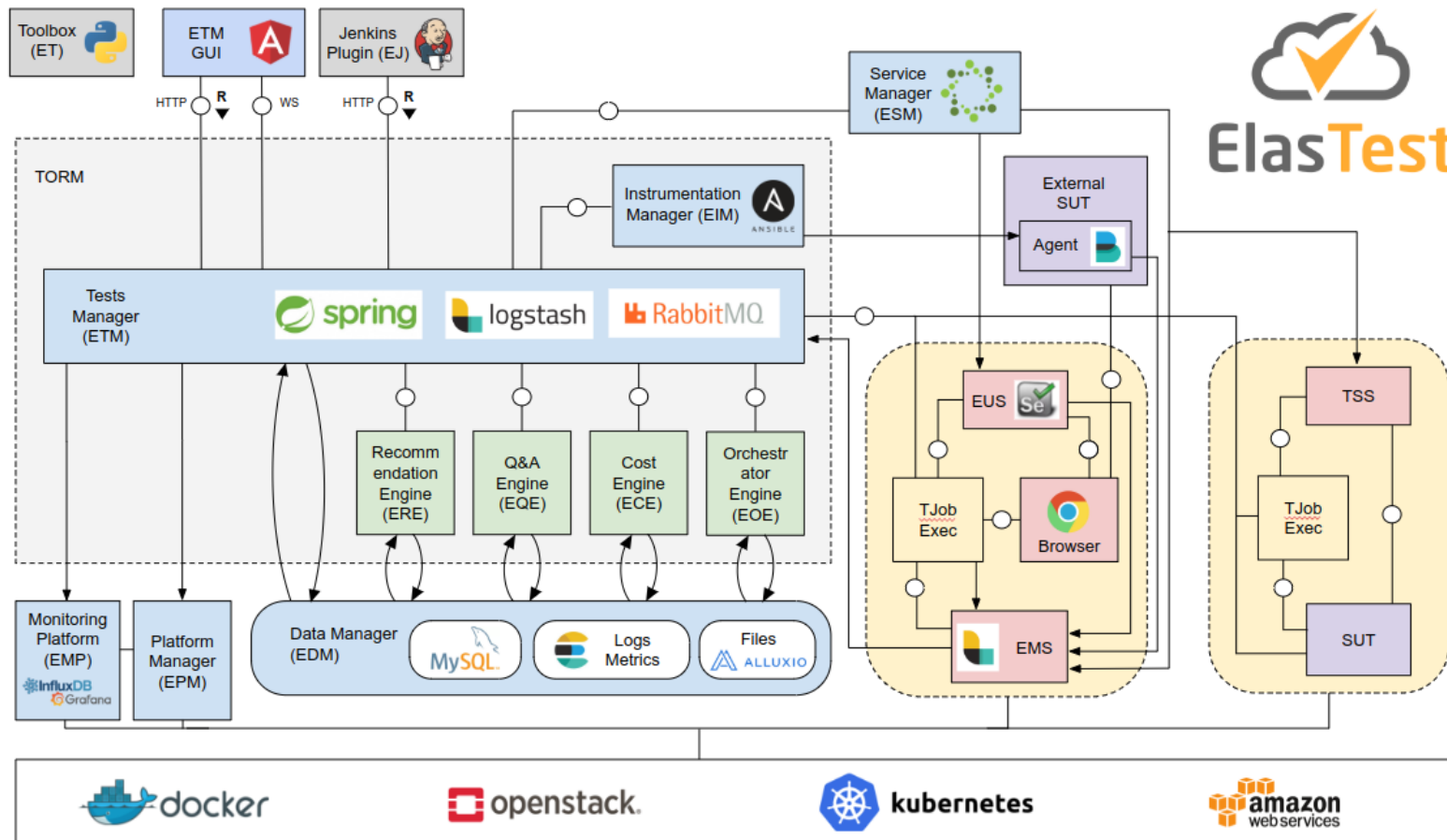


Table of contents



1. Introduction
2. Test orchestration approaches
3. ElasTest: platform for end-to-end testing
- 4. ElasTest Orchestration Engine**
5. Case study: testing WebRTC applications
6. Conclusions and future work

4. ElasTest Orchestration Engine



- The **ElasTest Orchestration Engine (EOE)** is the component responsible of implementing our concept of orchestration within ElasTest (i.e. select, order, and execute a group of tests)
- We use the **pipelines Jenkins notation** as a basis for the topology generation
- It has been implemented as a ***shared library*** which exposes a simple API to orchestrate jobs in Jenkins
 - Note that ElasTest's testing jobs (TJobs) can be executed also as Jenkins jobs

4. ElasTest Orchestration Engine



- The orchestrator library API is summarized in the following table:

Method	Description
<code>runJob(String jobId)</code>	Method to run a job given its identifier (<i>jobId</i>). This method returns a boolean value: <i>true</i> if the execution of the job finishes correctly and <i>false</i> if fails
<code>runJobDependingOn(boolean verdict, String job1Id, String job2Id)</code>	Method allows to run one job given a boolean value (typically a verdict from another job). This boolean value is passes in the first argument (called <i>verdict</i> in the method signature). If this value job with identifier <i>job1Id</i> is executed. Otherwise it is executed <i>job2Id</i>
<code>runJobsInParallel(String... jobs)</code>	This method allows to run a set of jobs in parallel. The jobs identifier are passes using a variable number of arguments (<i>varargs</i>)

4. ElasTest Orchestration Engine



- The exit condition for the orchestrated job can be configured:

Method	Description
EXIT_AT_END	The orchestration finishes at the end (option by default)
EXIT_ON_FAIL	The orchestration finishes when any of the TJobs fail
EXIT_ON_PARALLEL_FAILURE	The orchestration finishes when any a set of parallel TJobs fail

- The verdict of a group of parallel jobs can be also configured:

Method	Description
AND	The verdict of a set of jobs executed in parallel is <i>true</i> only if all the jobs finish correctly
OR	The verdict of parallel jobs is <i>true</i> when at least one of the jobs finishes correctly

4. ElasTest Orchestration Engine



- Example on the orchestration library usage:

```
@Library('OrchestrationLib') _  
  
// Config  
orchestrator.setContext(this)  
orchestrator.setParallelResultStrategy(ParallelResultStrategy.OR)  
orchestrator.setExitCondition(OrchestrationExitCondition.EXIT_ON_PARALLEL_FAILURE)  
  
// Graph  
def result1 = orchestrator.runJob('myjob1')  
def result2 = orchestrator.runJobDependingOn(result1, 'myjob2', 'myjob3')  
def result3 = orchestrator.runJob('myjob4')  
def result4 = orchestrator.runJobsInParallel('myjob5', 'myjob6')  
  
if (result4) {  
    orchestrator.runJob('myjob7')  
}  
else {  
    orchestrator.runJob('myjob8')  
}
```



Table of contents

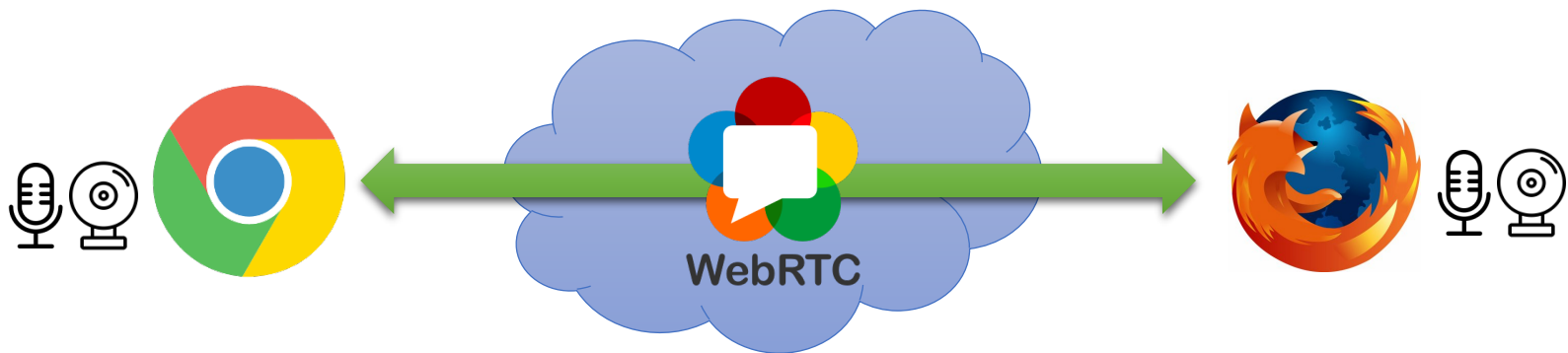


1. Introduction
2. Test orchestration approaches
3. ElasTest: platform for end-to-end testing
4. ElasTest Orchestration Engine
- 5. Case study: testing WebRTC applications**
6. Conclusions and future work

5. Case study: testing WebRTC applications



- **WebRTC** is the umbrella term for a number of technologies aimed to bring Real Time Communications to the Web
 - W3C (JavaScript APIs): *getUserMedia*, *PeerConnection*, *DataChannels*
 - IETF (protocol stack): ICE, SDP, TURN, STUN, ...



5. Case study: testing WebRTC applications



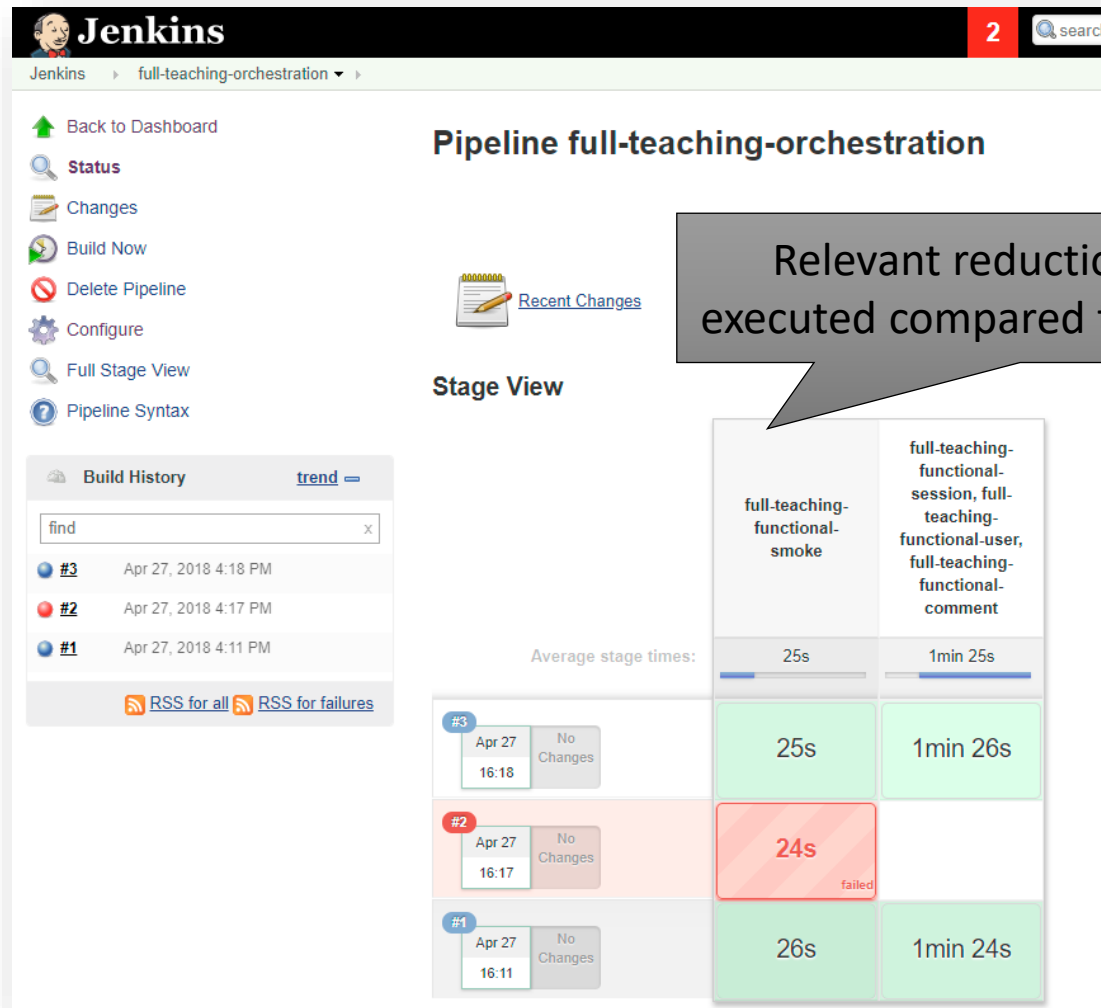
- Our case study is based on OpenVidu, an open source videoconferencing WebRTC framework
- Question driving this study: *“Is the orchestration approach presented in this paper capable of improving somehow the testing process of an existing test suite?”*



openVidu

<http://openvidu.io/>

5. Case study: testing WebRTC applications



Relevant reduction of time to be executed compared to the original suite

Table of contents



1. Introduction
2. Background
3. ElasTest: platform for end-to-end testing
4. ElasTest Orchestration Engine
5. Case study: testing WebRTC applications
6. **Conclusions and future work**

6. Conclusions and future work



- ElasTest is an open source platform aimed to ease end-to-end tests for heterogenous large distributed systems
- In ElasTest we are implementing the concept of test orchestration understood as a novel way to select, order, and execute a group of tests
- The topology generation for orchestrated test is based in the Jenkins pipeline notation
- In the near future we plan to extend our approach in different ways: data-driven orchestration and test augmentation