

D3.2

Version	1.0
Author	ATOS
Dissemination	PU
Date	23-12-2019
Status	FINAL



D3.2 ElasTest Platform Cloud Modules v.2

Project acronym	ELATEST
Project title	ElasTest: an elastic platform for testing complex distributed large software systems
Project duration	01-01-2017 to 31-12-2019
Project type	H2020-ICT-2016-1. Software Technologies
Project reference	731535
Project website	http://elastest.eu/
Work package	WP3
WP leader	ATOS
Deliverable nature	Report
Lead editor	Orlando Avila-García (ATOS)
Planned delivery date	31-12-2019
Actual delivery date	31-12-2019
Keywords	Open source software, cloud computing, software engineering, operating systems, computer languages, software design & development



Funded by the European Union

License

This is a public deliverable that is provided to the community under a **Creative Commons Attribution-ShareAlike 4.0 International** License:

<http://creativecommons.org/licenses/by-sa/4.0/>

You are free to:

Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

For a full description of the license legal terms, please refer to:

<http://creativecommons.org/licenses/by-sa/4.0/legalcode>



Contributors

Name	Affiliation
Orlando Avila-García	ATOS
Fernando Méndez	ATOS
Tran Quang Thanh	TUB
Piyush Harsh	ZHAW
Andy Edmonds	ZHAW
Micael Gallego	URJC
Eduardo Jiménez	URJC

Version history

Version	Date	Author(s)	Description of changes
0.1	01/11/2019	Orlando Avila-García	Table of content (ToC)
0.2	04/11/2019	Orlando Avila-García	Change in ToC to account for comments from ZHAW
0.3	20/11/2019	ALL	Initial contributions from ATOS (Section 6), TUB (Sections 3 and 4.1) and ZHAW (Sections 4.2 and 5). Adding Introduction (Section 2) by ATOS.
0.4	27/11/2019	Tran Quang, Orlando Avila-García, Andy Edmonds	Adding Section 4.1 by TUB, Conclusions (Sections 8) by ZHAW, and Executive summary (Section 1) by ATOS.
0.5	23/12/2019	Piyush Harsh, Tran Quang, Orlando Avila-García, Andy Edmond	Adding changes by TUB, ZHAW and ATOS to address amendments and comments resulting from the internal review by TUB.
1.0	23/12/2019	Orlando Avila-García	Final release

Table of contents

1	Executive summary	9
2	Introduction.....	10
2.1	Overview and Objectives	10
2.2	Structure of the Document	11
2.3	Target Audiences.....	11
3	ElasTest Cloud Modules	12
3.1	Rationale	12
3.2	Categories	12
3.3	Challenges to Overcome	13
3.4	Roadmap	13
4	Platform Management and Monitoring	15
4.1	ElasTest Platform Manager (EPM)	15
4.1.1	<i>Introduction</i>	<i>15</i>
4.1.2	<i>Baseline Concepts and Technologies</i>	<i>15</i>
4.1.3	<i>Component Design and Architecture</i>	<i>16</i>
4.1.4	<i>Roadmap and Features.....</i>	<i>19</i>
4.1.5	<i>Research results and Future Plan.....</i>	<i>21</i>
4.2	ElasTest Monitoring Platform (EMP).....	22
4.2.1	<i>Introduction</i>	<i>22</i>
4.2.2	<i>Baseline Concepts and Technologies</i>	<i>22</i>
4.2.3	<i>Component Design and Architecture</i>	<i>22</i>
4.2.4	<i>Roadmap and Features.....</i>	<i>23</i>
4.2.5	<i>Research results and Future Plan.....</i>	<i>28</i>
5	Service Lifecycle Management	30
5.1	ElasTest Service Manager (ESM)	30
5.1.1	<i>Introduction</i>	<i>30</i>
5.1.2	<i>Baseline Concepts and Technologies</i>	<i>30</i>
5.1.3	<i>Component Design and Architecture</i>	<i>30</i>
5.1.4	<i>Roadmap and Features.....</i>	<i>32</i>
5.1.5	<i>Research results and Future Plan.....</i>	<i>34</i>
6	SuT Instrumentation	35
6.1	ElasTest Instrumentation Manager (EIM) & Instrumentation Agents	35
6.1.1	<i>Introduction</i>	<i>35</i>
6.1.2	<i>Baseline Concepts and Technologies</i>	<i>35</i>
6.1.3	<i>Component Design and Architecture</i>	<i>35</i>
6.1.4	<i>Roadmap and Features.....</i>	<i>38</i>
7	Data Persistence Management.....	43
8	Conclusions.....	44
9	References.....	45

List of Figures

<i>Figure 1 – Schema of deployment elements within Kubernetes.....</i>	<i>16</i>
<i>Figure 2 – EPM High-level Architecture.....</i>	<i>17</i>
<i>Figure 3 – Worker creation sequence diagram</i>	<i>18</i>
<i>Figure 4 – Kubernetes cluster creation sequence diagram</i>	<i>18</i>
<i>Figure 5 – EMP High-level Architecture.....</i>	<i>22</i>
<i>Figure 6 – New API for the EMP.....</i>	<i>27</i>
<i>Figure 7 – New data visualization interface for the EMP.....</i>	<i>28</i>
<i>Figure 8 – ESM High-level Architecture.....</i>	<i>31</i>
<i>Figure 9 – The ESM graphical user interface showing the import of a service definition.....</i>	<i>32</i>
<i>Figure 10 – EIM High-level Architecture</i>	<i>36</i>
<i>Figure 11 – Main data model classes (resource types) in the EIM</i>	<i>37</i>
<i>Figure 12 – EIM Jenkins build report.....</i>	<i>42</i>

List of Tables

<i>Table 1 – Main features developed between M18 and M36 for the cloud enablers</i>	<i>14</i>
<i>Table 2 – EPM requirements validation summary</i>	<i>20</i>
<i>Table 3 – EMP requirements validation summary</i>	<i>26</i>
<i>Table 4 – ESM requirements validation summary</i>	<i>34</i>
<i>Table 5 – EIM controllability data model</i>	<i>37</i>
<i>Table 6 – EIM requirements validation summary</i>	<i>41</i>

Glossary of acronyms

Acronym	Description
CI	Continuous Integration. This refers to the software development practice with that name.
FOSS	Free Open Source Software. This refers to software released under open source licenses.
IaaS	Infrastructure as a Service. This refers to one of the models of exposing cloud capabilities and services to third parties.
PaaS	Platform as a Service. This refers to one of the models of exposing cloud capabilities and services to third parties.
SaaS	Software as a Service. This refers to one of the models of exposing cloud capabilities and services to third parties.
FaaS	Function-as-a-Service.
Instrumentation	This refers to extending the interface exposed by a software system for achieving enhanced controllability and observability
QoS	In this proposal, Quality of Service refers to non-functional attributes of systems. QoS is related to objective quality metrics such as latency or packet loss. In ElasTest, QoS is particularly important for the characterization of multimedia systems and applications through custom metrics.
QoE	In this proposal, Quality of Experience refers to non-functional attributes of systems. QoE is related to the subjective quality perception of users. In ElasTest, QoE is particularly important for the characterization of multimedia systems and applications through custom metrics.
SiL	A SiL (Systems in the Large) is a large distributed system exposing applications and services involving complex architectures on highly interconnected and heterogeneous environments. SiLs are typically created interconnecting, scaling and orchestrating different SiS. For example, a complex microservice-architected system deployed in a cloud environment and providing a service with elastic scalability is considered a SiL.
SiS	SiS (Systems in the Small) are systems basing on monolithic (i.e. non distributed) architectures. For us, a SiS can be considered a component that provides a specific functional capability to a larger system.
SuT	Software under Test. This refers to the software that a test is validating. In this project, SuT typically refers to a SiL that is under validation.
TO	Test Orchestration. The term orchestration typically refers to test orchestration understood as a technique for executing tests in

	coordination. This should not be confused with cloud orchestration, which is a completely different concept related to the orchestration of systems in a cloud environment.
TORM	Test Orchestration and Recommendation Manager. This is a functional aggregation of ElasTest components that abstracts away the details and exposes to testers the capabilities of the ElasTest orchestration and recommendation engines.
TJob	We define a TJob (Testing Job) as a monolithic (i.e. single process) program devoted to validating some specific attribute of a system. Current Continuous Integration tools are designed for automating the execution of TJobs. TJobs may have different flavours such as unit tests, which validate a specific function of a SiS, or integration and system tests, which may validate properties on a SiL as a whole.
TiL	A TiL (Test in the Large) refers to a set of tests that execute in coordination and that are suitable for validating complex functional and/or non-functional properties of a SiL on realistic operational conditions. We understand that a TiL can be created by orchestrating the execution of several TJob.
ICT	Information and Communication Technology
IT	Information Technology
WP	Work Package
FMC	Fundamental Model Concept
ETM	ElasTest Test Manager
EPM	ElasTest Platform Manager
EMP	ElasTest Monitoring Platform
ESM	ElasTest Service Manager
EIM	ElasTest Instrumentation Manager
EDM	ElasTest Data Manager
TSS	Test Support Service
EUS	ElasTest User Impersonation Service
ESS	ElasTest Security Service
ECE	ElasTest Cost Engine
PoP	Point of Presence
REST	Representational State Transfer
VDU	Virtual Deployment Unit
AWS	Amazon Web Services
AAA	Authentication, Authorization, Accounting
TOSCA	Topology and Orchestration Specification for Cloud Applications
API	Application Programming Interface

SDK	Software Development Kit
SSH	Secure Shell
CPU	Central Processing Unit
R&D	Research and Development
OSBA	Open Service Broker API
SLA	Service Level Agreement
DoA	Description of Actions
UI	User Interface
GUI	Graphical User Interface
VM	Virtual Machine
KVM	Kernel-based Virtual Machine
JDK	Java Development Kit
KPI	Key Performance Indicator
R	Release
MS	Milestone

1 Executive summary

This deliverable reports the increment of the ElasTest Platform cloud testing enablers carried out within WP3 between M18 to M36. This document should be read and understood as an extension of D3.1 [3], which describes the development of such enablers during the first half of the project; the present deliverable focuses on what has been done since then. Thus, this document presents the design specifications and implementation details for that period, covering the Platform Manager (EPM), the Monitoring Platform (EMP), the Service Manager (ESM), the Instrumentation Manager (EIM) and the Data Management (EDM) capabilities of the ElasTest Platform. The key goals for WP3 during this period have been: i) the ability to provide resources and services to execute TJobs and support the complete Platform (EPM and ESM), ii) the ability to provide resources to deploy and execute SuTs (EPM), the ability to provide insights into the Platform (EMP), and the ability to provide controllability of SuTs in order to operationalize realistic execution environments (EIM). The new versions of these enablers released during the M18-M36 period were key to carry out realistic system-level and/or end-to-end testing throughout WP6 and WP7 activities. Furthermore, these components have all been key to sustain core platform functions, such as the ETM (ElasTest Test Manager).

2 Introduction

2.1 Overview and Objectives

In recent years, **cloud testing** and more specifically “testing in the cloud” has arisen to facilitate testing of large-scale distributed systems [12]. In this new paradigm, cloud-based environments and infrastructures are used to carry out **realistic** system-level and/or end-to-end testing. This includes collecting logs and measurements of the performance of the application and data services, and infrastructure resources comprising the environment, and allowing for the use of data analytics and data visualization techniques on them.

Cloud technologies therefore facilitate the cost-effective construction of large-scale production-like testing environments; however, while this approach brings the possibility to pursue innovative and more effective testing solutions, it also brings important challenges [12]. This deliverable presents a set of cloud testing enablers to address some of the challenges in the **test execution** domain—as presented by the authors of this deliverable in a conference paper [24]:

Firstly, the distributed application needs to be deployed, therefore an adequate scalable environment is needed. Secondly, the application needs to be exercised by the test in different ways depending on the attributes or functionalities to check, which sometimes means using services like browsers, IoT device emulation or data processing and management tools. Thirdly, to test extra-functional properties, the application's operational environment needs to be controlled during testing to resemble real production conditions. Finally, during the testing process the different systems that comprise the application need to be monitored, their logs collected, and every piece of information that might render useful for a post-analysis retrieved and presented in a meaningful way. This is usually called observability, and in distributed applications might be the only way to understand why a test failed, given the impossibility of debugging the whole thing.

This deliverable aims at presenting how the collection of ElasTest cloud testing modules seek to provide the testers with cloud testing capabilities to accomplish the test enactment process mentioned above, including the operationalization of realistic production environments. These services are part of the ElasTest platform and are deployable as cloud-native components in a wide range of infrastructures, either together or separately from the ElasTest platform [13].

Notice this deliverable reports the evolution of the cloud testing enablers in the scope of the WP3 from M18 to M36. In order to avoid the duplication of content, those aspects of the components which have remained unchanged since M18 are not described again here; please refer to D3.2 [6] for a full comprehension of those components.

These cloud testing enablers have been realized as software components and subsystems required to run the ElasTest Platform: Platform Manager (EPM), Monitoring Platform (EMP), Service Manager (ESM), Instrumentation Manager (EIM) and its deployable Instrumentation Agents. Data Manager (EDM) is also included as one of the

software components of WP3; however, its section in this deliverable appears empty because no upgrades were required for that component since M18.

2.2 Structure of the Document

The structure of this deliverable is as follows: Section 1 introduces the document, its objectives and motivation. Section 2 presents the ElasTest Cloud modules describing how they are categorised and its overall roadmap since M18—the roadmap up to that point in the lifetime of the project is already described in D3.1 [6]. The following four sections describes the cloud enablers for managing and monitoring the ElasTest Platform in a target cloud provider (Section 4), the mechanism and interfaces offered for managing the on-demand cloud-based test support services offered by the Platform (Section 5), the mechanisms used to instrumentalize the software under test (Section 6), and the service offering data persistence capabilities to the rest of components of the Platform (Section 7). Finally, in Section 8, some conclusions of our work on cloud testing enablers from M18 to M36 are drawn.

2.3 Target Audiences

The primary audience of this document are internal ElasTest technicians from WP3, WP4, WP5 and WP6 involved in the development of the ElasTest platform. In addition, this document would also be useful to developers and testers interested in cloud testing, in general, and the approach taken by ElasTest, in particular. Finally, QA managers and software architects seeking and/or evaluating testing platforms can use this document to know technical details about ElasTest in order to assess its suitability and/or compare it with alternative platforms and tools.

3 ElasTest Cloud Modules

3.1 Rationale

New advances in ICT technology influence the way software is developed and tested, the proliferation of large-scale applications targeting thousands of users that can be connected concurrently and expect real time interactions; makes the testing strategy a crucial aspect for the release management process of the applications.

Nowadays cloud technologies are creating advantages for organizations that adopt it such as: speed, agility, scalability, accessibility and flexibility; therefore ElasTest aims to extend the adoption of the aforementioned benefits offered by the cloud to testers through the creation of a cloud platform (ElasTest Platform) designed for helping to validate large software systems that require complex test suites and validation processes.

Since the irruption of the cloud computing (together with the virtualization era) as a disruptive technology, the increased use of the cloud introduced new business opportunities and challenges during the last years allowing developers to apply more easily the principles of mass production into the IT world. The current panorama reveals that a whole range of IT functions can be thought of as commodity services.

The ElasTest cloud components described within this report is concerned with the management and monitoring of the resources that the platform needs to operate; as well as of the lifecycle management associated to the on-demand testing support services catalogue which can be requested by the ElasTest Platform user dynamically. In addition to the cloud components in charge of the platform management, the report also includes other kind of cloud-based component not targeting the platform itself but offering management capabilities over the software system under evaluation.

3.2 Categories

The different categories identified have a direct relationship with the tasks described within the “WP3 Cloud components”. Task 3.1 implements the enablers for the platform components to be deployed in a target cloud being able as well to monitor its usage recovering in seamless way information related to the runtime execution of the platform. Task 3.2 implements the appropriate mechanism enabling the lifecycle management of the Test Support Services catalogue offered by ElasTest. Finally, Tasks 3.3 & 3.4 are devoted to the instrumentation capabilities offered over the software under evaluation.

As it has been introduced in the previous paragraph, different categories have been considered:

- A. Software modules for managing the computational resources of the platform.
- B. Software modules for managing the cloud-based services offered by the platform.
- C. Software modules for managing the applications under test.

3.3 Challenges to Overcome

ElasTest is a platform designed to facilitate the build, execution and reporting of end-to-end tests of complex distributed applications. These types of applications present some properties like elasticity and fault tolerance that need to be tested with end-to-end tests. To execute these complex not just distributed applications but also scalable tests runs, enabling resources and supporting services are needed. The primary reason that such elements must be provided is to remove the tester from the responsibility of having to manage these resources and services themselves and in doing so allow them to focus on their core business focus, writing complete tests that validate the SuT.

Not only resources and services should be provided for TJobs, additional cloud components must also be provided in order to allow ElasTest deploy and execute a SuT on the behalf of the tester; also, deploy and execute the components required to run the ElasTest platform itself. In summary WP3's main goals for the M18-M36 period has been the following:

- to provide resources and services to execute TJobs and support the complete ElasTest platform through the EPM and ESM,
- to provide resources to deploy and execute SuTs through the EPM,
- to provide necessary insights into ElasTest platform, that is, the current and past state of ElasTest core components in order to facilitate stable operation of the platform itself, through the EMP, and
- to provide controllability of SuTs in order to operationalize realistic execution environments, suitable for validating non-functional properties on realistic operational conditions, through the EIM.

The key aim and contribution of WP3 to ElasTest is to provide the enabling facilities required by the ETM (TORM) to carry out its task of orchestration and executing tester supplied TJobs. As such it can be thought of as the enabling platform for the ETM.

3.4 Roadmap

ElasTest uses an Agile Management methodology, which is suitable for innovation management. This methodology has been designed for transforming ideas into profitable products. For this, it focuses on learning and discovering how to fit a technology into the market instead on how to carry out the technological developments themselves. See D2.5 for more details.

Following such a methodology, the different cloud testing enablers underwent several upgrades between M18 and M36 to achieve their goals (outlined in Section 3.3):

- EPM (ElasTest Platform Manager)
- EMP (ElasTest Monitoring Platform)
- ESM (ElasTest Service Manager)
- EIM (ElasTest Instrumentation Manager)
- EDM (ElasTest Data Management)

The table below lists the cloud enablers' main features developed in that increment spanning from M18 (the date of the previous report D3.1 [6]) to M36:

Component	Feature
EPM	Fast and Flexible Worker creation
EPM	Kubernetes Cluster on OpenStack and AWS
EPM	Dynamic Kubernetes Cluster Management
EPM	EPM deployment on Kubernetes (within ElasTest)
EMP	API extension to enable ECE usage-based cost computation
EMP	Visualization engine data storage optimization
EMP	Visualization engine support to dynamic dashboards
EMP	EMP deployment on Kubernetes (within ElasTest)
ESM	Catalog of public services
ESM	Services deployment by using Kubernetes templates
ESM	ESM deployment on Kubernetes (within ElasTest)
EIM	Portability of the instrumentation agents and manager
EIM	Scalability of the instrumentation manager
EIM	Persistence of agent configuration data by means of EDM
EIM	Controllability of CPU, network and container failures
EIM	EIM deployment on Kubernetes
EDM	N/A (no upgrades required)

Table 1 – Main features developed between M18 and M36 for the cloud enablers

4 Platform Management and Monitoring

The following section introduces the core components in charge of the management and monitoring of the platform; and provides the details of the requirements, architecture, interfaces and features for each of them.

4.1 ElasTest Platform Manager (EPM)

4.1.1 Introduction

The ElasTest Platform Manager (EPM) is designed to serve as an interface for the ElasTest components/consumers (e.g. TORM, Test Support Services, etc.) and the underlying cloud platform. The EPM abstracts the connection to the cloud infrastructure by providing Software Development Kits (SDK) or REST APIs to allow service consumers manage virtual resources in a target cloud environment. Hence, ElasTest becomes fully agnostic to the cloud services. Through the EPM, ElasTest can seamlessly control the cloud technologies that the consortium considers as appropriate (e.g. OpenStack, AWS, Docker, etc.).

4.1.2 Baseline Concepts and Technologies

The EPM itself is implemented in Java making use of the Spring framework¹. Data persistency is provided via SQL where by default it uses an in-memory database (HyperSQL²). Nevertheless, other SQL databases (e.g. MySQL³) can be easily integrated by changing the configuration inside the main properties file following the spring configuration guide. Two approaches are supported by the EPM in the meaning of the consumer can either make use of the EPM's data model or TOSCA⁴ to describe the deployment scenario or use directly templates of a certain technology. Thanks to the modular approach, other virtualization infrastructures can be easily supported by providing adapters for certain technologies. This adapter mechanism is provided via gRPC⁵ which manages the communication between the EPM itself and the corresponding adapter.

In addition, the EPM makes indirectly use of several supporting services by configuring the virtual instances for the purpose of log forwarding (e.g. Logstash⁶) or monitoring (e.g. Dockbeat⁷) which are then provided indirectly to other services for further processing, such as, the ElasTest Monitoring Service, ElasTest Monitoring Platform, or the ElasTest Test Manager.

¹ Spring Framework, <https://spring.io/>

² HyperSQL, <https://spring.io/>

³ Oracle MySQL, <https://www.mysql.com/>

⁴ OASIS Topology and Orchestration Specification for Cloud Application, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

⁵ A high performance, open-source universal RPC framework, <https://grpc.io/>

⁶ Logstash, <https://www.elastic.co/products/logstash>

⁷ DockBeat, <https://www.elastic.co/blog/dockbeat-a-new-addition-to-the-beats-community>

The EPM and all the available adapters are delivered as Docker containers which are available in Docker Hub. In addition, several docker-compose files are provided in the GitHub repositories to start easily the EPM with the additional components and services to ease the deployment and configuration.

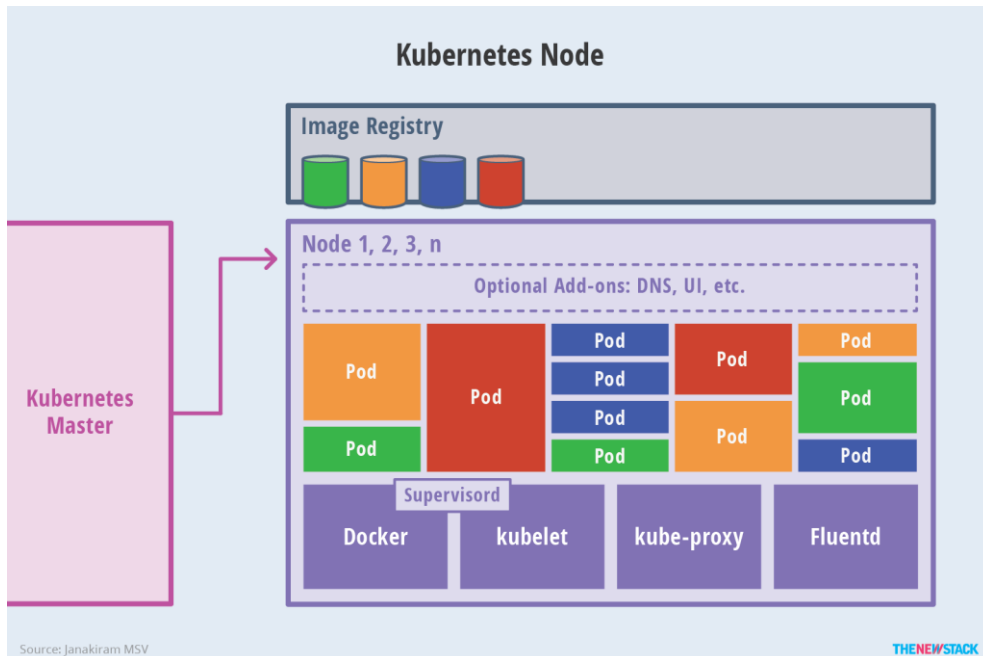


Figure 1 – Schema of deployment elements within Kubernetes

Supporting Kubernetes⁸ deployment is the most important requirement in the phase 2 of ElasTest. This capability has been implemented to be able to provide a bigger audience of end-users. Kubernetes is an open source system for automating deployment, scaling, and management of containerized applications (It was started by Google in 2014 and later donated to the Cloud Native Computing Foundation). With Kubernetes we can quickly deploy our applications, scaling it according to our needs, without having to stop anything in the process. The high-level architecture of a Kubernetes cluster (which is, in Kubernetes terminology, the equivalent of a datacenter) is depicted in the figure below.

4.1.3 Component Design and Architecture

Figure 2 describes the high-level architecture of EPM. The core functionalities include: to allocate, terminate, update virtual resources (e.g. compute, network) and request information of those as well, to execute run-time operations, and to register and configure new workers. For maintaining state and to allow the user to retrieve state and information of the allocated virtual resources, the EPM maintains data in a repository. The EPM adopts modular architecture where the Core is decoupled from so called “*EPM Adapters*” that provide an abstracted way to interact with any kind of cloud environment. The northbound interface is exposed to the Core and abstracted in such a way, that the Core do not need to take care about the type of the target cloud

⁸ <https://justanotherdevblog.com/2017/02/22/kubernetes-an-overview-bf47b0af1865/>

environment - it just needs to know to which adapter to send the requests. The southbound interface is dependent on the type of target cloud environment. This allows an easy way to provide any kind of cloud environment by providing an adapter without changing anything in the core. The EPM Adapter takes also care about the configuration of logging and monitoring of the virtualized resources by receiving that information by the EPM component either defined by the Consumer itself or the default configuration.

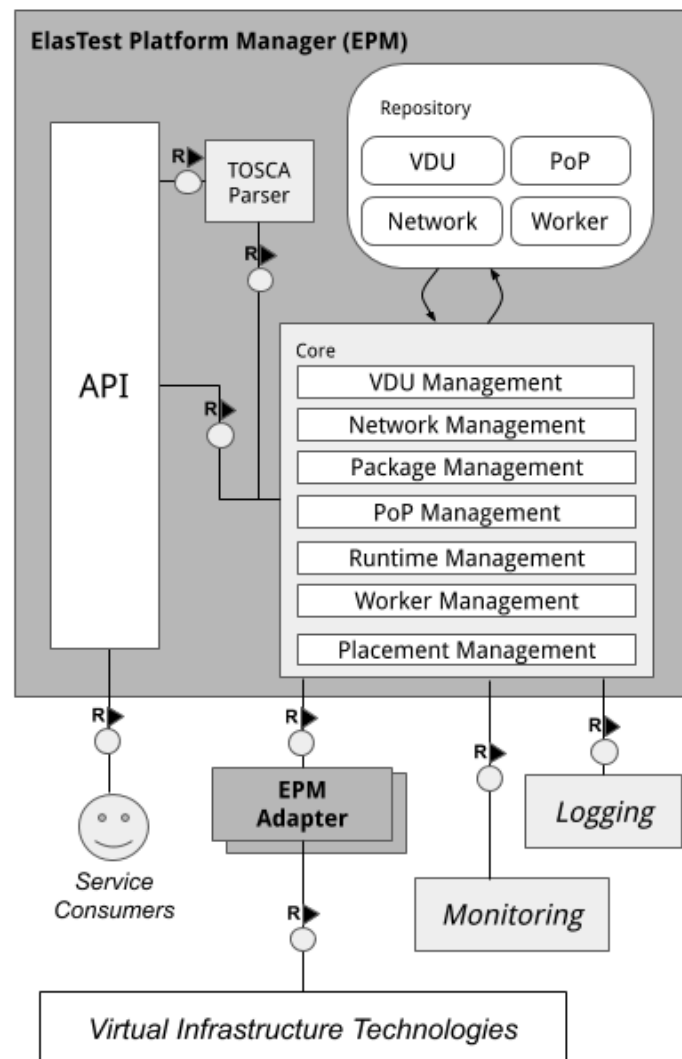


Figure 2 – EPM High-level Architecture

The EPM provides two different options to describe and deploy virtual resources: “*All-in-One*” and “*Step-by-Step*”. The former is designed to make use of template-dependent technologies such as docker-compose or Ansible, to give consumers the freedom to use pre-existing templates. Such template with additional metadata will be forwarded directly to the target virtual infrastructure to trigger the deployment as a whole (e.g. SuT, TSS). The latter, on the other hand, targets IaaS infrastructures (e.g. Docker, OpenStack, AWS) where EPM receives the resource description which is compliant to the EPM internal information model or TOSCA. This allows consumers to use the same definition for various cloud infrastructures.

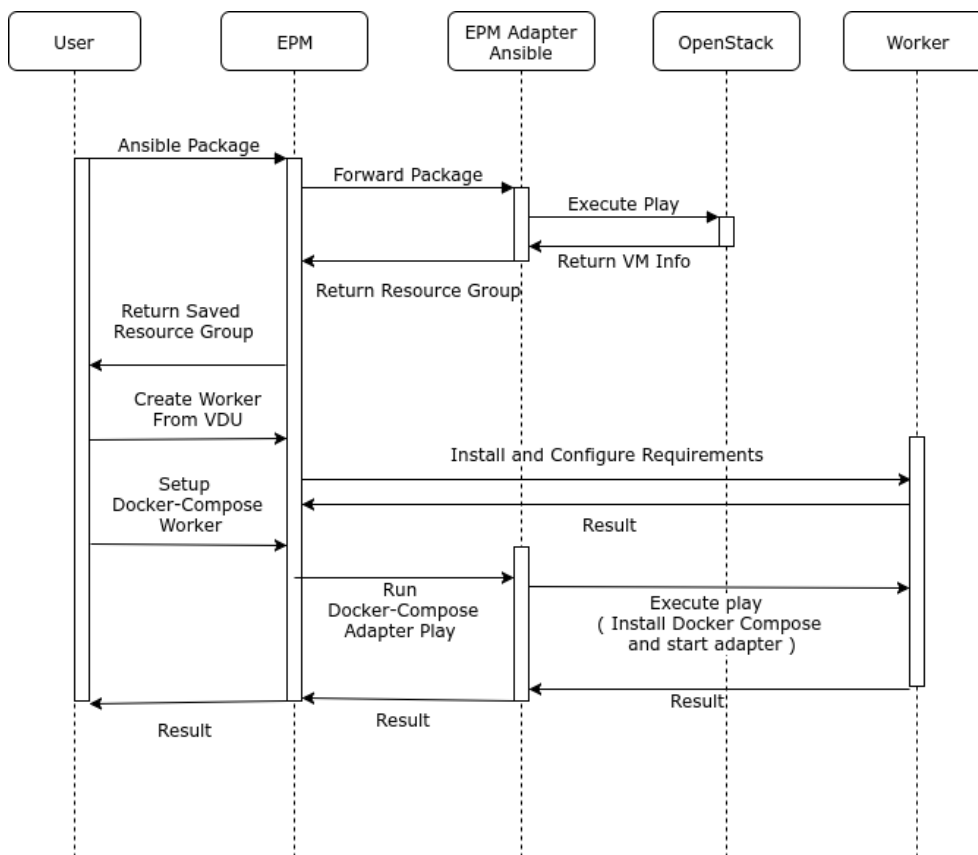


Figure 3 – Worker creation sequence diagram

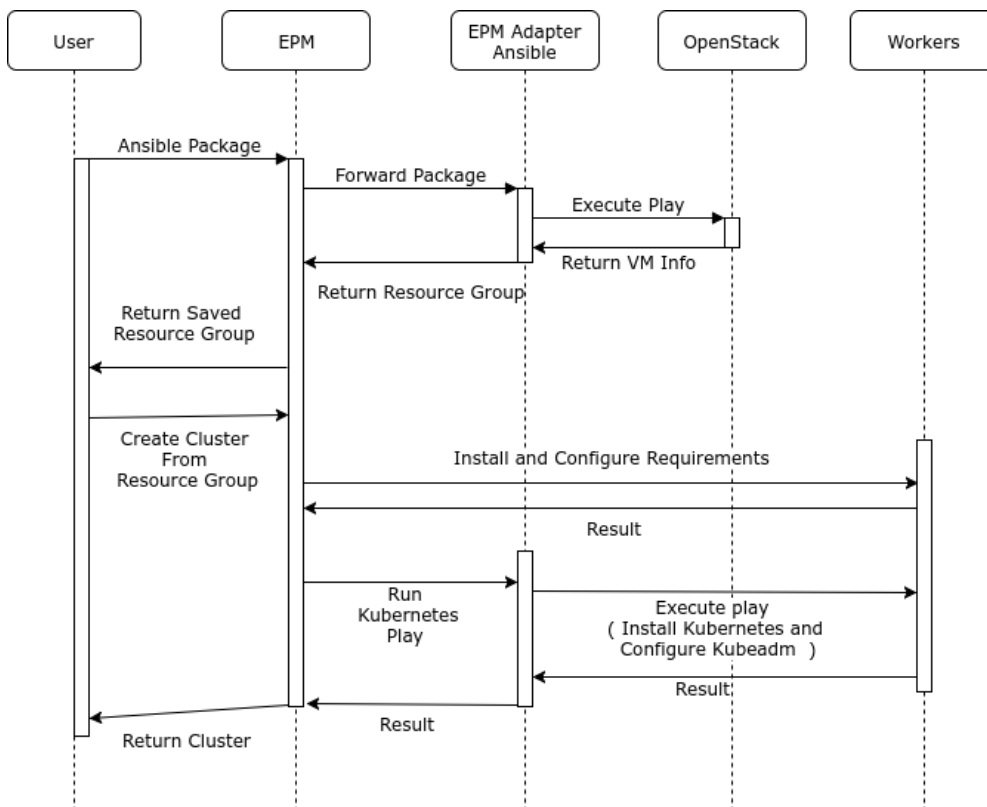


Figure 4 – Kubernetes cluster creation sequence diagram

4.1.4 Roadmap and Features

In the following sections we present the final set of requirements that has driven the implementation of the component, and the details of the final increments of its development roadmap (since M18).

4.1.4.1 Requirements Validation Summary

Requirements from EPM are being tracked as part of WP2 activities, and details can be found in D2.3 [5] and D2.5 [15]. The following table shows the status of validation of those requirements at M36 as well as the test coverage achieved for this component.

#	Title	As a <type of user>	I want <some good>	So that <some reason>
EPM 1	Abstraction of underlying virtualization technologies	ElasTest Component (TORM, Test Support Services, etc.)	to be able to orchestrate and manage virtualized resources (compute, network, storage) on any virtualization technology under consideration	the consumer does not need to care about the target virtualization technology but can define needed resources in a generic way.
EPM 2	Providing Northbound API	ElasTest Component	to interact with the EPM	the consumer can make use of the EPM via a RESTful API
EPM 3	Providing SDKs	ElasTest Component	to interact with the EPM	developers of other components can easily integrate with the EPM by making use of SDKs (libraries) provided for different languages (e.g. java, python)
EPM 10	Containers monitoring	ElasTest Component	to monitor instances managed by the EPM	the consumer can retrieve monitoring information for further evaluation and troubleshooting
EPM 11	Log forwarding	ElasTest Component	to forward logs to the configured endpoint	other parties can access those logs which can be used for further troubleshooting and debugging
EPM 15	Support for OpenStack	ElasTest Component	to be able to allocate, manage and terminate VMs via OpenStack	the consumer can make use of OpenStack as a virtualization technology

EPM 16	Kubernetes Cluster on OpenStack	ElasTest Component	to be able to deploy Kubernetes in OpenStack	the consumer of the EPM want to deploy SUTs on top of Kubernetes
EPM 7	Platform - Linux support	User	to run the EPM in Linux as the OS with native docker	the user of ElasTest has the free choice of the underlying OS where ElasTest is running
EPM 8	Platform - Mac support	User	to run the EPM in Mac OS as the OS with docker for Mac	the user of ElasTest has the free choice of the underlying OS where ElasTest is running
EPM 9	Platform - Windows support	User	to run the EPM in Windows with the docker toolbox as the OS and docker for Windows	the user of ElasTest has the free choice of the underlying OS where ElasTest is running

Table 2 – EPM requirements validation summary

4.1.4.2 Road Map

Some important updates carried out from M18 to M36 are listed as follows:

1. *Fast and Flexible Worker creation*
 - a. Improved Worker Creation
 - i. Create a VM & Register it as a Worker - 2 API Calls
 - ii. Supported Types: Docker Compose, Docker, Kubernetes
 - b. Improved Worker Setup
 - i. Worker setup - Ansible Plays
 - ii. Worker type - set on registration
2. *Kubernetes Cluster on OpenStack and AWS*
 - a. Cluster Model and API
 - b. Create & Delete Clusters
 - c. Set up Clusters from fresh VMs
 - d. Create Kubernetes Cluster
 - e. Install Kubernetes on VMs
 - f. Configure Cluster using *Kubeadm*
 - g. Add node to an already Running Cluster
 - h. Worker becomes part of a Cluster using Ansible Play
3. *Dynamic Kubernetes Cluster Management*
 - a. Manual scale out and in of workers

- b. Provide Java SDKs via Maven Central
- c. Migration of adapters to new repository

4.1.4.3 Code Reports

The EPM code repository can be found on GitHub (see table below) and is licenses using Apache 2.0. Within that repository, there is detailed documentation about its usage and provides API.

Subcomponent	Code repository
EPM	https://github.com/elastest/elastest-platform-manager
Adapter - Docker	https://github.com/tub-elastest/epm-adapter-docker
Adapter - docker-compose	https://github.com/tub-elastest/epm-adapter-docker-compose
Adapter - Ansible	https://github.com/tub-elastest/epm-adapter-ansible
Adapter - Virtual Box	https://github.com/tub-elastest/epm-adapter-vbox
Adapter - Open Baton	https://gitlab.fokus.fraunhofer.de/ogo/elastest-openbaton-adapter
SDK Client - Java	https://github.com/tub-elastest/epm-client-java
SDK Client - Python	https://github.com/tub-elastest/epm-client-python

4.1.5 Research results and Future Plan

The EPM has been delivered to satisfy all the needs of the ElasTest platform. The flexible architecture allows adapting with different types of requirements and its application in ElasTest ecosystem has been described in a recent WP3 publication to UCC19 [14]. Through the EPM, ElasTest can seamlessly control the cloud technologies that the consortium considers as appropriate (e.g. OpenStack, AWS, Docker, Kubernetes). Several targets are under considerations including not only the extending/developing new adapters for existing NFV Orchestrator (e.g. OSM, OpenBaton) to support telco testing requirements, but also supporting deploy domain specific external SUTs (e.g. virtual mobile core network).

4.2 ElasTest Monitoring Platform (EMP)

4.2.1 Introduction

ElasTest monitoring platform is a suite of tools made of targeted agents that collect various metrics from host nodes, as well as critical ElasTest service processes periodically, plus metric aggregators and visualizers. The principal goal EMP is to enable monitoring of ElasTest core components themselves, which allows the operator of ElasTest to diagnose possible fault lines quickly, remove bottlenecks thereby ensuring reactivity and fluidity of overall test framework.

4.2.2 Baseline Concepts and Technologies

EMP concepts and technologies have not changed since M18 for the most part (please refer to D3.1 [6], Section 4.2.2 for a full explanation). The only significant change has been in the underlying Java runtime switch from OpenJDK to IBM's open source OpenJ9 runtime (see Section 4.2.3).

4.2.3 Component Design and Architecture

The EMP baseline architecture has remained unchanged from D3.1 [6].

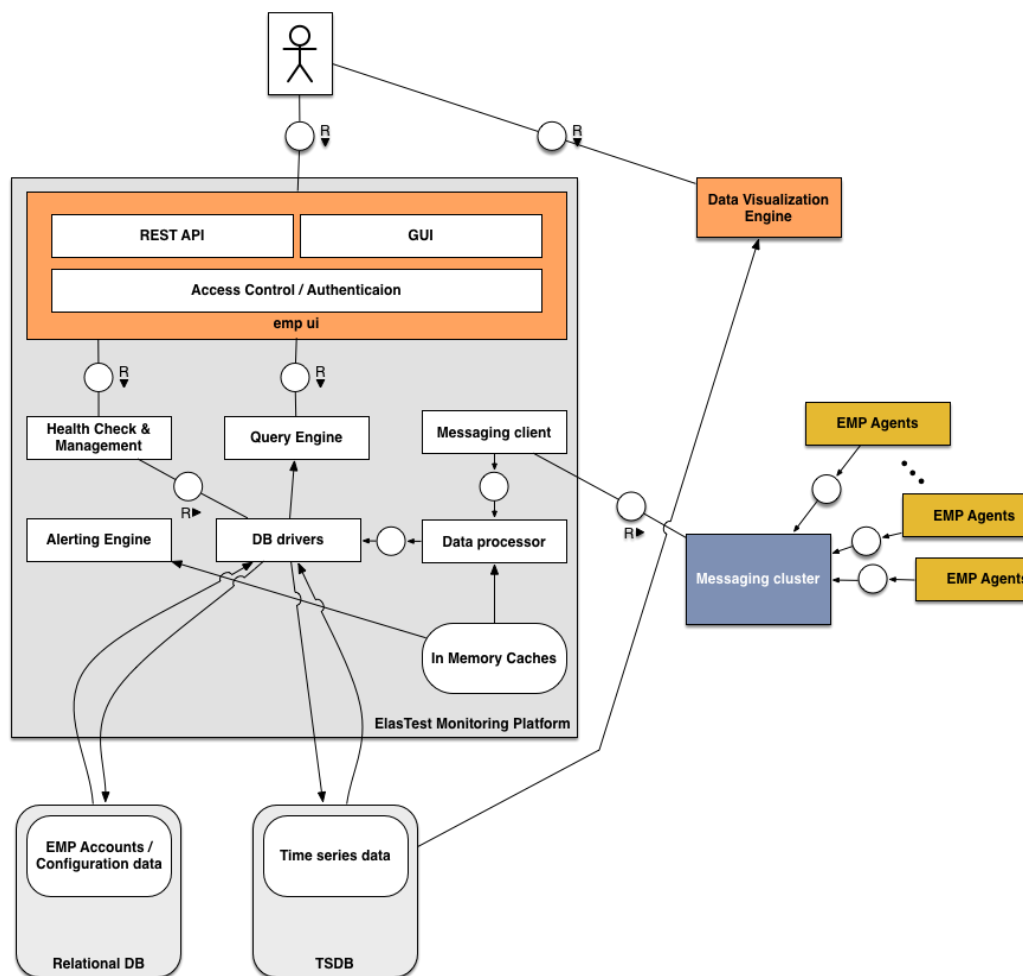


Figure 5 – EMP High-level Architecture

However, the underlying Java runtime was changed: from OpenJDK to IBM's open source OpenJ9 runtime. This switch was done to reduce the memory footprint of the module which was necessary to enable deployment in Kubernetes clusters over physical or virtual infrastructure clusters with rather limited resources.

Since the original architecture was already conducive for delivery as a set of container processes, deployment of EMP as Kubernetes pods was straightforward barring slight update in EMP dashboard definition which was needed due to changed container naming strategy within Kubernetes.

4.2.4 Roadmap and Features

In the following sections we present the final set of requirements that has driven the implementation of the component, and the details of the final increments of its development roadmap (since M18).

4.2.4.1 Requirements Validation Summary

Requirements from EMP are being tracked as part of WP2 activities, and details can be found in D2.3 [5] and D2.5 [15]. The following table shows the status of validation of those requirements at M36 as well as the test coverage achieved for this component.

#	Title	As a <type of user>	I want <some good>	So that <some reason>	Validation test
EMP 1	Monitoring spaces	Complex application, System developer, Integrator	To be able to specify a separate monitoring space for the overall application	I can get easy, properly segregated access to my overall metric / log data	API-EMP-001
EMP 2	Monitoring subspaces	Complex application, System developer, Integrator	To be able to further separate metric and log stream of an application sub-component / microservice from rest of the components	I can easily locate the data stream coming from one component versus looking at a large set of data points from all possible metric generation sources in my large application (possibly distributed)	API-EMP-002
EMP 3	API authentication and authorization	Monitoring system user	My access to be authenticated and properly logged for safety as well as	No one else to be able to access the data streams from my services as	API-EMP-001 API-EMP-002 API-EMP-003 API-EMP-006

			auditing purposes	they may contain sensitive data	
EMP 4	Receive system metric streams	ElasTest platform operator, Monitoring user	To be able to send relevant system metrics into the monitoring system	I can analyze data trends later or in real time	API-EMP-004 API-EMP-005
EMP 5	Persist system metric streams	ElasTest platform operator, Monitoring user	My data points to be stored for a specified period in time	I can do detailed offline analysis of trends and / or investigate bottlenecks / problem areas with my application	API-EMP-004 API-EMP-005
EMP 6	Receive application log streams	ElasTest platform operator, Monitoring user	To use same service preferably to send my log messages too	I can do a proper correlation study of service degradation as observed from logs and the environment metric data	API-EMP-004 API-EMP-005
EMP 7	Persist application log streams	ElasTest platform operator, Monitoring user	My data points aka log parts to be stored for a specified period of time	I can do offline / historical data analysis	API-EMP-004 API-EMP-005
EMP 8	Data query capability	Monitoring user	To be able to see stored data points	I can analyze data trends and observe system trends	API-EMP-006
EMP 9	Metric visualization	Monitoring user, Application developer, Operator	To be able to see charts / graphs visualizing data points in a meaningful way	I can comprehend quickly trends over time from metric streams from my services	GUI-EMP-001
EMP 10	Cross space/subspace correlated query capability	Monitoring user, Application developer, Operator	To perform advance inter- space/domain data query	I can gain insight into correlation among various services on each other's performance	N/A

EMP 11	Health check capability	Application developer, Operator	To be able to monitor the liveness of set of target services	I know as soon as possible when a service is down in order to react in a timely manner for restoring it	API-EMP-007
EMP 12	Alerting capability	Operator	To be alerted if one of my services becomes dead	I can react quickly to restore the service	API-EMP-007
EMP 13	Online expression evaluation against metric data stream	Operator	To ensure that the minimum service availability and contracts with my users are always supported and if any violation is notified to me as soon as possible	I satisfy my service level agreement terms	N/A
EMP 14	RESTful APIs	Monitoring user, Application developer, operator	To easily integrate with the monitoring service with clearly defined interfaces	I can send metrics and perform control operations through my application code logic rather than interacting with the monitoring service in a standalone detached mode	API-EMP-001 API-EMP-002 API-EMP-003 API-EMP-008
EMP 15	Availability of commonly used metric collectors (agents)	Operator, Application developer	To easily collect and send most commonly used system metrics into the monitoring platform	I can concentrate more on my system specific instrumentation and monitoring	N/A
EMP 16	Showing in ElasTest GUI monitoring information of all components	user	See all metrics and other monitoring information in ElasTest GUI	I can know the status of the system	GUI-EMP-001 API-EMP-006

EMP 17	API for querying TJob resource consumption parameters	Elastest cost engine process	Given a TJob ID, resource consumption data across all known executions	I can compute the true cost of TJob execution based on cost models defined in ESM	API-EMP-006
EMP 18	Monitor Kubernetes clusters	ElasTest operator	To quickly see the status of my Kubernetes clusters	I can quickly identify hotspots and take corrective measures	N/A
EMP 19	Individual container metrics visualization	Monitoring system user	To see CPU, memory and networking stats for any container and not just the core components of ElasTest	I can visually see any abnormal spikes in data	GUI-EMP-001 API-EMP-006

Table 3 – EMP requirements validation summary

In the above table the mapping between component requirements and automated validation tests are shown. This information is a summary of the work done at WP6.

A couple of requirements namely 10, and 13 have been deprioritized in the light of need of remaining requirements by other components in ElasTest. These retain their significance in terms of research worthiness and have been explained in the last subsection dealing with future plans. Remaining requirements have been fulfilled until the time of writing of this deliverable.

4.2.4.2 Road Map

The following sections present the main feature updates in EMP since D3.1 [6].

4.2.4.2.1 API Updates

api-controller : API Controller

Show/Hide | List Operations | Expand Operations

GET	/v1/api/	getApis
GET	/v1/api/endpoint	getEndpointInfo
GET	/v1/api/key/{userid}	locateUserKey
POST	/v1/api/pingback/	createPingBack
GET	/v1/api/pingback/{pingid}	locatePingData
POST	/v1/api/series/	createSeries
POST	/v1/api/space/	createSpace
POST	/v1/api/user/	createUser
GET	/v1/api/user/{userid}	locateUserData
GET	/v1/dashboard/	showDashboard
GET	/v1/dashboardsrc	showDashboardIframeSrc
GET	/v1/error	returnError

basic-error-controller : Basic Error Controller

Show/Hide | List Operations | Expand Operations

controller : Controller

Show/Hide | List Operations | Expand Operations

elas-test : Elas Test

Show/Hide | List Operations | Expand Operations

GET	/v1/extension/elastest/api/	getApis
GET	/v1/extension/elastest/tjobstat/{tjobid}	getTJobStats

[BASE URL: / , API VERSION: 1]

Figure 6 – New API for the EMP

Since D3.1, a few extra APIs have been added to EMP, these primarily were added to enable true usage-based cost computation within ElasTest Cost Engine (ECE). More information on ECE and its evolution since D4.1 can be found in D4.2. The additional API endpoints have been developed keeping backward compatibility of remaining APIs, thereby ensuring all integration remained valid all through the development phase.

4.2.4.2.2 GUI Updates



Figure 7 – New data visualization interface for the EMP

EMP visualization engine underwent a few major iterations. The data layer was modified to optimally store data coming in from various agents. Host and Container-Name were changed from type “field” to “tags” which allowed EMP to support dynamic dashboards. Now it is possible to select metrics charts per host node where ElasTest components have been deployed. Similarly, now it is possible to see any container process metrics from the drop-down list.

4.2.5 Research results and Future Plan

A collaborative peer-reviewed conference paper which included the EMP architecture and feature set was accepted as part of CloudAM workshop at the “12th IEEE/ACM International Conference on Utility and Cloud Computing.” One of the targets to be achieved in the near term is the ability to support correlated queries by the end users, the initial support towards this was already implemented as part of this phase of work since D3.1 [6]—the capability to query for a TJob usage data over a queried time window. Distributed traceability has become a major need in the developer community recently, and a tool to enable FaaS (Function-as-a-Service) debugging is also fast becoming a necessity. EMP already supports log ingestion; a correlated query over logs

from multiple “series” within the same ‘space’ (see D3.1) can fulfill some of the needs of this emerging developer community, so it is worth looking into.

5 Service Lifecycle Management

The following section introduces the core components in charge of the management of the lifecycle of the test support services within the ElasTest platform; and provides the details of the requirements, architecture, interfaces and features for it.

5.1 ElasTest Service Manager (ESM)

5.1.1 Introduction

Delivering service instances to the end-users on-demand is meant to be a seamless task done with efficiency. To accomplish this, the ESM provides the possibility to deploy services using Docker-compose- and Kubernetes-based capabilities through YAML files, which describe the diverse components that constitute a service. All of this has been made available through the ElasTest Service Manager API, based upon the 2.12 release of the Open Service Broker API, with ElasTest-specific extensions, such as the ones required for the registration of services.

For the specifics of how a TSS should be presented to the ESM, D5.1. information and the minor updates contained within D5.2. Do note however, that efforts to minimize backward-compatibility have been made and to date no significant changes are required.

5.1.2 Baseline Concepts and Technologies

For the specifics of the Baseline Concepts and Technologies, including information related to the Open Service Broker API, OSBA and Billing, and the ESM Data Model, refer to the information contained in D3.1 [6].

5.1.3 Component Design and Architecture

There are several components that constitute the ESM; for a detailed description of these components refer to D3.1 [6], since here we provide the novelties related to each component that have changed since M18.

- **Workflow:** the ESM provides the deployment of a service as a static construction of tasks, where once a service is registered in the catalog, along with the respective manifest and plan, the instantiation of it follows: backend identification, deployment of the components described in the service manifest descriptor, information collection and distribution, and finally de-provision of components on-demand.
- **Kubernetes Implementation:** this capability has been implemented to be able to provide a bigger audience of end-users the benefits of the ESM for provisioning services. The ESM now operates upon Kubernetes allowing for it to be easier to scale based on different replica groups. As services can also be deployed upon Kubernetes, this allows service authors to take avail of the scaling primitives made available by Kubernetes. The Kubernetes backend is used via loading the credentials that the Kubernetes cluster provides to the ESM pod, and then through the cluster API.

- **Updating a Service Instance:** this functionality has been replaced by a higher priority feature, which is the deletion of a service definition (plus plan and manifest) capabilities in the ESM's API.
- **Service Import:** a requested feature was that to use the service definitions of the TSS listed in each service's repository (for example: the EUS service definition⁹) or indeed any service description available at a URL, which has been provided in the ESM's UI, as shown in the following figure:

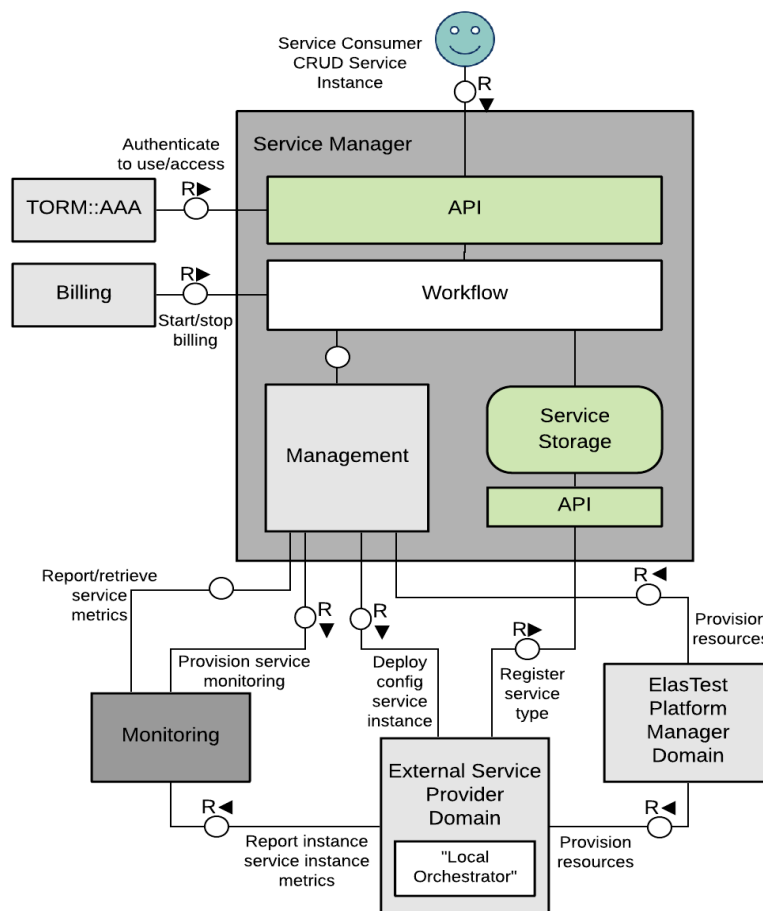


Figure 8 – ESM High-level Architecture

The ESM continues to have a number of interactions with external entities such as the EPM, AAA and Monitoring. Regarding Billing, the ESM is the entity that informs Billing components e.g. cost estimation with information required to make decisions on billing. This is exemplified by the Cost Estimation Engine (CEE) delivered out of WP4. It makes all its decisions based on costs that are provided by the ESM.

⁹ <https://github.com/elastest/elastest-user-emulator-service/blob/master/elastestservice.json>

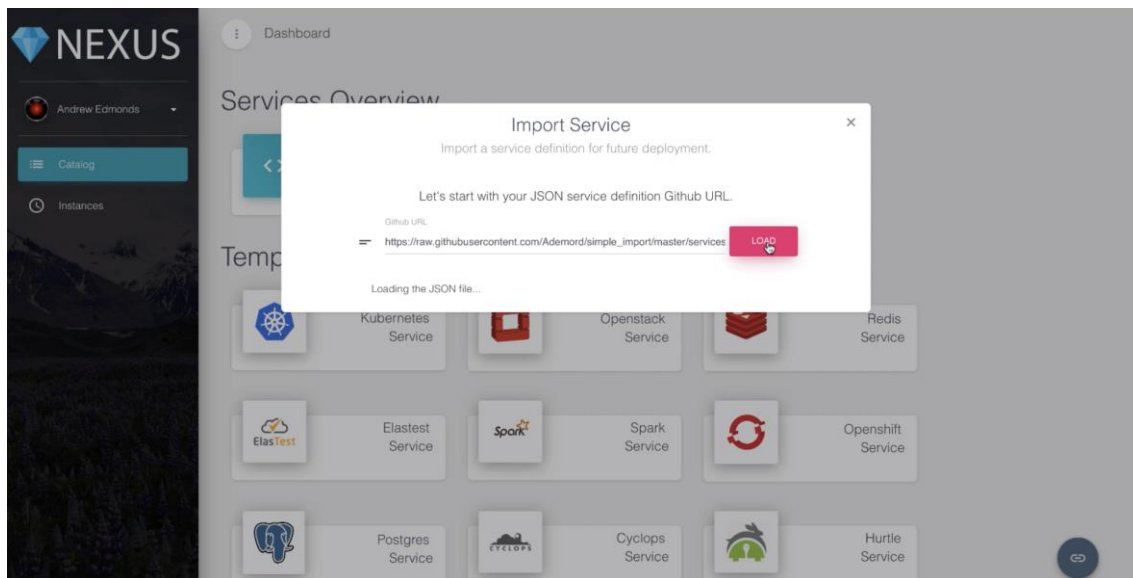


Figure 9 – The ESM graphical user interface showing the import of a service definition

For the details of a Service Lifecycle in the ESM, from Design to Disposal phase, please refer to D3.1 [6].

5.1.3.1 Sequence Diagrams

In D.2.3, the sequence diagrams related to the ESM only showed the interactions with other external components and services. As such it provides a black box view. There are further details that are illustrated in sequence diagrams (see D3.1 [6]) that show the internal details of the ESM or in other words the “white box” view.

5.1.4 Roadmap and Features

In the following sections we present the final set of requirements that has driven the implementation of the component, and the details of the final increments of its development roadmap (since M18).

5.1.4.1 Requirements Validation Summary

Requirements from ESM are being tracked as part of WP2 activities, and details can be found in D2.3 [5] and D2.5 [15]. The following table shows the status of validation of those requirements at M36 as well as the test coverage achieved for this component.

#	Title	As a <type of user>	I want <some good>	So that <some reason>	Validation test
ESM 1	Create a Support Service Instance	TORM	to create a service instance	additional test functionality can be used	API-ESM-010 API-ESM-011 API-ESM-012 API-ESM-013
ESM 2	Collect Information from Service Instance	TORM	to get a list of my service instances	I can reuse and understand what I have running	API-ESM-017 API-ESM-018 API-ESM-019

ESM 3	Deprovision Service Instance	TORM	to delete a service instance	I don't get charged for it	API-ESM-020
ESM 4	Configure Service instance	TORM	to configure a service instance with new or updated parameters	the software can run as desired by end user	API-ESM-014 API-ESM-015
ESM 5	Register TSS offer	TORM	to register a TSS	I can offer my software as a service to the TORM	API-ESM-001 API-ESM-002 API-ESM-003 API-ESM-004 API-ESM-005
ESM 6	Update TSS offer	TORM	to update a TSS's technical and business description	I can change my offer	API-ESM-006 API-ESM-007
ESM 7	Delete TSS offer	TORM	to delete a TSS	I do not offer it anymore	API-ESM-021
ESM 8	Register Service Instance with Monitoring	ESM	to register the service instance an endpoint with a monitoring service	the TSS provider can ensure guarantees offered to the TORM/end-user are met and adjustments can be made to ensure this	This is functionality mainly looked after by EMP and also some tests in test_measurer.py ¹⁰
ESM 9	Public catalog to allow end users to "install" and "uninstall" services in the current ElasTest instance	User	to install a TSS from a public catalog/registry	new TSSs can be installed in an ElasTest instance	N/A
ESM 10	Allow deployment of services based on a Kubernetes	TORM	to register a service with a Kubernetes YAML description	Kubernetes and docker-compose can be supported by the ESM	API-ESM-022

¹⁰ https://github.com/elastest/elastest-service-manager/blob/master/tests/test_measurer.py

YAML
descriptor

Table 4 – ESM requirements validation summary

In the above table the mapping between component requirements and validation tests are shown. This information is a summary of the work done out of WP2's architecture.

5.1.4.2 Code Reports

The ESM is implemented using python and this code is tested using the Python *unit test* module. There are currently 97 tests that are run against all configurations of the ESM. These tests consist of both unit and integration tests. Currently the code coverage is at approximately 80%, which is calculated using codecov.io¹¹ every time a new build of the ESM is done. As shown, the code coverage for the ESM for the last 6 months has remained stable. Note that currently code complexity is not calculated.

5.1.5 Research results and Future Plan

The ESM has been delivered to satisfy all the needs of the ElasTest platform and it now delivers all required TSS-related functionality to TJobs and authors of TJobs. The ESM has been described in a recent WP3 publication to UCC19¹² where it's described in a wider context. Investigations into replacing the default compute infrastructure with hypervisor-based execution of containers has revealed that with the appropriate hypervisor and image conversion tooling, the same "docker" experience can be maintained. To enable debugging of ESM provisioned services has been suitably addressed by the integration of the EMP work. Moving forward from where the ESM is at, work related to function-as-a-service and provisioning services using edge-based infrastructure (e.g. k3s¹³) is being considered with hypervisor-based compute facilities.

¹¹ <https://codecov.io/gh/elastest/elastest-service-manager>

¹² <https://www.ucc-conference.org>

¹³ <https://k3s.io>

6 SuT Instrumentation

The following section introduces the core components in charge of the management of the instrumentation of the software under test (SuT) within the ElasTest platform; and provides the details of the requirements, architecture, interfaces and features for it.

6.1 ElasTest Instrumentation Manager (EIM) & Instrumentation Agents

6.1.1 Introduction

Tasks 3.3 & 3.4 oversee designing and implementing the Instrumentation Agents and the Instrumentation Manager (EIM), respectively. In this section we report the novelties or changes those testing services underwent between M18 and M36.

6.1.2 Baseline Concepts and Technologies

As explained already in D3.1 [6], we refer as instrumentation to extending the interface exposed by a software system for achieving enhanced controllability (i.e. the ability to modify the operational environment at runtime) and observability (i.e. the ability to infer information about the runtime internal state of the system).

Technologies used in the development of EIM have not changed since M18 (please refer to D3.1 [6], Section 6.1.2 for a full explanation).

6.1.3 Component Design and Architecture

The EIM lets testers to exercise observability and controllability through the so-called instrumentation agents: software agents that are deployed along the software under test (SuT), at operating system-level (i.e. physical machine, virtual machines or containers). They can be of two types:

- Observability, through which the Agent collects all information relevant for testing or monitoring purposes (e.g. energy consumption, resources utilization, etc.).
- Controllability, through which the agent can force custom behaviours on the host's network, CPU utilization, memory consumption, process life cycle management, failure injection, etc.

The present report covers controllability, as this was the focus between M18 and M36; and the observability capability didn't receive changes since M18. For a full description of such capabilities as well as initial steps towards controllability support, please refer to D3.1 [6], Section 6.1.3 for a full description.

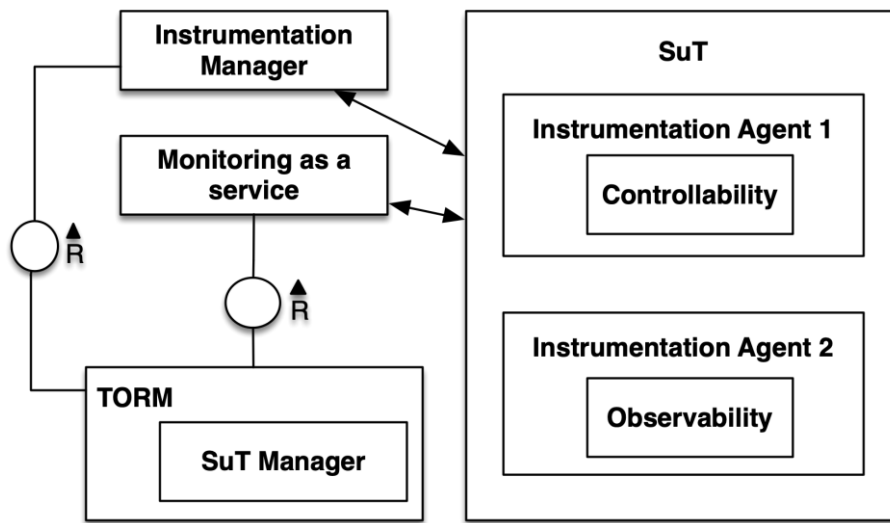


Figure 10 – EIM High-level Architecture

ETM (within TORM) consumes the EIM (see Figure 10) to install observability agents in SuT at test deployment time. These agents cover the most common metrics of compute, storage and networking resource utilization. This means testers don't have to worry about the collection of these types of metrics to observe the performance of their tests. However, the controllability agents must be explicitly installed by the tester by making requests to the EIM in their test code, at test runtime. In this way, testers are the sole responsible for synthesizing custom operational conditions for their tests.

6.1.3.1 API Updates

Since D3.1, the EIM API and data model have been upgraded to enable SuT controllability, to operationalize realistic execution environments in non-functional properties validation scenarios. Additional API endpoints—in the “controllability” path—and one new class (or API resource type) in the data model were developed to support controllability within the EIM. Their design kept backward compatibility, thereby ensuring all the integrations remained valid all through the rollout of the upgrade.

Publickey		▼
GET	/publickey	Retrieve public key
Agent		▼
GET	/agent/{agentId}	Retrieve a agent
DELETE	/agent/{agentId}	Delete a agent
GET	/agent	Returns all existing agents
POST	/agent	Register an agent
POST	/agent/{agentId}/{actionId}	Submit an action to an agent
POST	/agent/controllability/{agentId}/stress	Submit an action to an agent
POST	/agent/controllability/{agentId}/packetloss	Submit an action to an agent
DELETE	/agent/controllability/{agentId}/unchecked	Uncheck an agent
AgentConfiguration		▼
GET	/agentconfiguration	Returns all existing agent configurations
GET	/agentconfiguration/{agentId}	Retrieve an agent configuration

The data model has remained the same apart from the new types related to the realization of the controllability feature in the EIM. Figure 11 shows the main classes (resource types) engaged in the SuT observability and controllability.

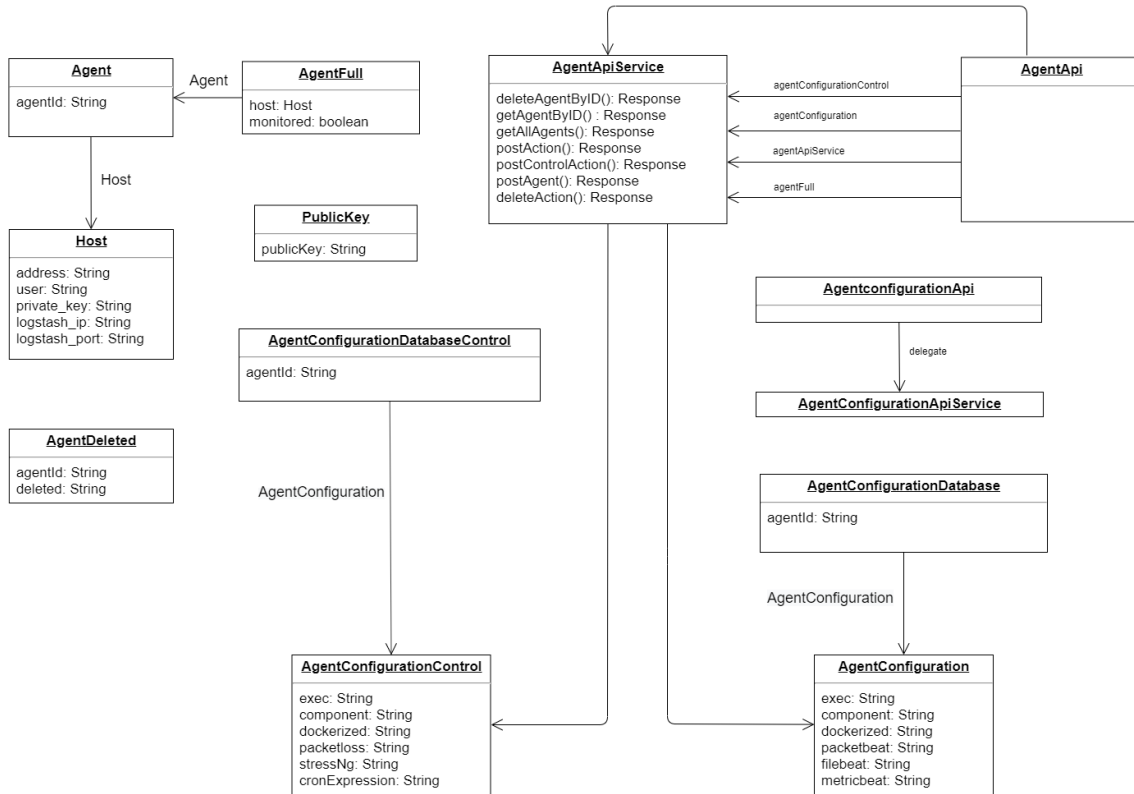


Figure 11 – Main data model classes (resource types) in the EIM

AgentConfigurationControl is the new resource type for controllability, carrying the configuration of the agent, in charge of executing controllability commands on the host machine where the SuT as well as the agent are deployed. The following table shows the attributes of this new type, incorporated in the EIM API to support the management of controllability actions.

Name	Description	Schema
<i>exec</i>	This entity defines the ID of the agent	String
<i>component</i>	Target host	String
<i>type</i>	Action name = {StressNg, PacketLoss, ContainerFailure}	String
<i>value</i>	The value of the control	String
<i>dockerized</i>	It uses docker path	String
<i>cronExpression</i>	Time expression	String

Table 5 – EIM controllability data model

The rest of API resource types remained unchanged, as described in D3.1.

6.1.4 Roadmap and Features

In the following sections we present the final set of requirements that has driven the implementation of the component, and the details of the final increments of its development roadmap (since M18).

6.1.4.1 Requirements Validation Summary

Requirements from EIM are being tracked as part of WP2 activities, and details can be found in D2.3 [5] and D2.5 [15]. The following table shows the status of validation of those requirements at M36 as well as the test coverage achieved for this component.

#	Title	As a <type of user>	I want <some good>	So that <some reason>	Validation test
EIM 1	Non-Intrusive	ElasTest user	instrumentation agents to be as less intrusive as possible	to produce low overhead of the instrumentation on the software under test (SuT).	API-EIM-001
					API-EIM-002
					API-EIM-003
					API-EIM-004
					API-EIM-005
					API-EIM-006
					API-EIM-007
					API-EIM-008
					API-EIM-009
EIM 2	Lightweight	ElasTest user	instrumentation agents to be as lightweight as possible	to deploy them within the software under test (SuT).	API-EIM-001
					API-EIM-002
					API-EIM-003
					API-EIM-004
					API-EIM-005
					API-EIM-006
					API-EIM-007
					API-EIM-008
					API-EIM-009
EIM 3	Configuration Management	ElasTest component (TORM)	to deploy automatically instrumentation agents in the target infrastructure	to achieve automated installation of instrumentation agents across target compute environments, such as bare metal, VMs, cloud instances (IaaS such as AWS or	API-EIM-001
					API-EIM-002
					API-EIM-003
					API-EIM-004
					API-EIM-005
					API-EIM-006
					API-EIM-007
					API-EIM-008

				OpenStack) and container platforms (such as Kubernetes).	API-EIM-009
EIM 4	Interoperability	ElasTest user	to maintain interoperability across different operating systems (OS) and distributions	the agents should be designed to consume well - established operating system interfaces to guarantee interoperability; supporting Linux systems at least.	API-EIM-001 API-EIM-002 API-EIM-003 API-EIM-004 API-EIM-005 API-EIM-006 API-EIM-007 API-EIM-008 API-EIM-009
EIM 5	Agent management	ElasTest component (TORM)	to perform CRUD operations to manage the lifecycle of instrumentation agents	EIM offers northbound interfaces which controls and orchestrates the operation of instrumentation agents.	GUI-EIM-002 API-EIM-001 API-EIM-002 API-EIM-003 API-EIM-004 API-EIM-005 API-EIM-006 API-EIM-007 API-EIM-008 API-EIM-009
EIM 6	Persistence	ElasTest component	to store configuration data in structured database (Mysql-like)	I can persist and query my structured data about the agents. (R3 only support MongoDB).	GUI-EIM-002 API-EIM-001 API-EIM-002 API-EIM-003 API-EIM-004 API-EIM-005 API-EIM-006 API-EIM-007 API-EIM-008 API-EIM-009
EIM 7	Observability	ElasTest user	to extend the interface exposed by a software system	I can have the ability to collect the logs and metrics and	GUI-EIM-002 API-EIM-007

			for archiving enhanced observability of the software under test (SuT)	performance data from the software under test (SuT) through the instrumentation agents.	
EIM 8	Portability	ElasTest component	to be able to install the instrumentation agents and manager across different compute environments.	to enable the installation, configuration and provisioning of the EIM along the rest of the ElasTest platform, and its instrumentation agents in the supported SuT environments.	GUI-EIM-002 API-EIM-006 API-EIM-007
EIM 9	Scalability	ElasTest component	to provide a scalable solution to the instrumentation of the software under test for both observability and controllability	to avoid the degradation of test (and the overall ElasTest platform) performance when running an increasing number of instrumentalized SuTs.	GUI-EIM-002 GUI-EIM-003 API-EIM-006 API-EIM-007
EIM 14	Controllability of CPU stress	ElasTest user	to achieve enhanced controllability of the stress level of the CPU	to simulate a rise in the load/use of the CPU of the machine running a software under test (SuT)	GUI-EIM-002 API-EIM-008 API-EIM-009
EIM 15	Controllability of Network failures	ElasTest user	to achieve enhanced controllability of the stress level of the network interface	to simulate network packet loss of the machine running a software under test (SuT)	GUI-EIM-003 GUI-EIM-005 API-EIM-001 API-EIM-002 API-EIM-003 API-EIM-004 API-EIM-005
EIM 16	Controllability of Container failures	ElasTest user	to control the failures (breakdown) of containerized	to simulate failures of some or all replicas of some or all	

	software components	components of a software under test (SuT)
--	------------------------	---

Table 6 – EIM requirements validation summary

In the above table the mapping between component requirements and automated validation tests are shown. This information is a summary of the work done at WP6.

The only feature that we could not validate through automated tests is EIM16 due to the late implementation in the lifespan of the project. Nevertheless, this requirement was property tested and validated through conventional manual testing techniques.

6.1.4.2 Road Map

The GUI components of the instrumentation solutions did not change since M18 (please refer to D3.1, Section 6.1.4.1 for more details). This is also the case for the EIM API, which remains the same, because the controllability feature is implemented as a new type of agent configuration, which are in turn managed through the endpoint and operations designed for observability.

The EIM features released between M18 (included) and M36 are shown in the following list, organized by ElasTest official released:

R5-Final

- **Portability**: To be able to install the instrumentation agents and manager across different compute environments.
- **Scalability**: To provide a scalable solution to the instrumentation of the software under test for both observability and controllability

R6

- **Persistence** (prototype): To store configuration data in structured database (Mysql-like)

R7a

- **Persistence** (integrated): To store configuration data in structured database (Mysql-like)

R7b


- **Controllability of CPU overload**: To achieve enhanced controllability of the stress level of the CPU.
- **Controllability of Network failures**: To achieve enhanced controllability of the stress level of the network interface.
- **E2E tests**: to evaluate the correctness of the controllability features integrated into the ElasTest Platform.


R7-Final


- **Controllability of Container failures**: To control the failures (breakdown) of containerized software components through chaos testing techniques.


6.1.4.3 Code Reports

EIM has been integrated with the CI system that uses Jenkins for automated tests and builds after every commit. Also, it has been included *end-two-end* related on WP6 reports


elastest-instrumentation-manager


 We are still working to improve the service!
 Please report any incidence or sugerence to elastest@naevatec.com


 add description

All 









































S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav	# Issues
		eim	1 day 19 hr - #543	N/A	1 min 51 sec	 	-
		eim-comp-images	1 day 19 hr - #661	N/A	12 min	 	-
		eim-e2e-test	6 mo 29 days - #201	N/A	7 min 47 sec	 	-
		eim-e2e-test-elastest	23 hr - #620	23 hr - #619	13 min	 	-
		eim-e2e-test-sandbox	1 yr 1 mo - #9	1 yr 1 mo - #8	17 sec	 	-
		eim-junit	8 mo 0 days - #467	N/A	9 min 27 sec	 	-
		eim-rest-api	1 day 19 hr - #436	N/A	2 min 12 sec	 	-
		eim-rest-api-docker-Pipeline	1 yr 8 mo - #7	2 yr 0 mo - #2	6 min 10 sec	 	-
		eim-sut-deploy-image	4 mo 19 days - #9	4 mo 19 days - #8	11 min	 	-
		eim-sut-deploy-image-tester	3 mo 23 days - #19	3 mo 23 days - #17	2 min 23 sec	 	-

Figure 12 – EIM Jenkins build report

7 Data Persistence Management

From M18, The ElasTest Data Management (EDM) service did not undergo any change so its design specification and implementation details have remained the same, as described in D3.1. **Please refer to that deliverable for more details about this component.**

8 Conclusions

This deliverable has reported the evolution of the cloud testing enablers in the scope of the WP3 from M18 to M36. The key goals of this period and the work package itself have been delivered in the form of four important capabilities of the ElasTest Platform:

- ability to provide resources and services to execute TJobs and support the complete Platform through the EPM and ESM,
- ability to provide resources to deploy and execute SuTs through the EPM,
- ability to provide necessary insights into the Platform, that is, the current and past state of ElasTest core components in order to facilitate stable operation of the platform itself, through the EMP, and
- ability to provide controllability of SuTs in order to operationalize realistic execution environments, suitable for validating non-functional properties on realistic operational conditions, through the EIM.

With these components implemented and delivered, they have been continually used to carry out realistic system-level and/or end-to-end testing throughout WP6 and WP7 activities. These components have all been key to high level components such as the TORM. In general, they facilitate the cost-effective construction of large-scale production-like testing environments in the cloud.

All the components of WP3 have been clearly documented, licensed under the open source Apache 2.0 license and various academic publications have been successfully made available to the relevant communities. Finally, the resulting outputs and discovered future work, as detailed per component section, for all components will continue beyond the life of ElasTest.

9 References

- [1] ElasTest project Description of Action (DoA) – part B. Amendment 1. Reference Ares (2017)343382. 23 January 2017.
- [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] D2.3. ElasTest requirements, use-cases and architecture v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.
- [6] D3.1. ElasTest Platform Cloud Modules v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.
- [7] D4.1. Test Orchestration basic toolbox v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.
- [8] D4.2. Test recommendation engines v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.
- [9] D5.1. ElasTest Test Support Services v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.
- [10] D6.1. ElasTest Continuous Integration and Validation System v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.
- [11] D6.2. ElasTest platform toolbox and integrations v1. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.
- [12] Antonia Bertolino, Guglielmo De Angelis, Micael Gallego, Boni García, Francisco Gortázar, Francesca Lonetti, and Eda Marchetti. 2019. A Systematic Review on Cloud Testing. *ACM Computing Surveys (CSUR)*, 52, 5, Article 93 (Aug. 2019). <https://doi.org/10.1145/3331447> (to be published).
- [13] Boni García, Micael Gallego, Francisco Gortázar, and Luis López-Fernández. 2017. ElasTest, an Open-Source Platform to Ease End-to-End Testing. In *Challenges and Opportunities in ICT Research Projects*, Volume 1: EPS Madrid 2017. INSTICC, SciTePress, 3–21. <https://doi.org/10.5220/0007904700030021>
- [14] Piyush Harsh, Juan Francisco Ribera Laszkowski, Enric Pages, Orlando Avila-García, Tran Quang Thanh, Francisco Gortázar Bellas and Micael Gallego Carrillo. 2019. Cloud Enablers for Testing Large-Scale Distributed Applications. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*,

CloudAM workshop, Auckland University of Technology, New Zealand (Dec. 2019). ACM ISBN 978-1-4503-7044-8/19/12. <https://doi.org/10.1145/3368235.3368838>.

- [15] D2.5. ElasTest requirements, use-cases and architecture v2. Grant Agreement number: 731535 - ELASTEST - H2020-ICT-2016-2017/H2020-ICT-2016-1. EUROPEAN COMMISSION.