D2.3	
Version	1.0
Author	TUB
Dissemination	PU
Date	30-06-2018
Status	FINAL

D2.3 ElasTest requirements, use-cases and architecture v1

Project acronym	ELASTEST
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	WP2
WP leader	TUB
Deliverable nature	Public
Lead editor	Varun Gowtham
Planned delivery date	30-06-2018
Actual delivery date	30-06-2018
Keywords	Open source software, cloud computing, software engineering, operating systems, computer languages, software design & development

Funded by the European Union

License

Contributors

Name	Affiliation
Guglielmo De Angelis	CNR
Francisco Díaz	URJC
Andy Edmonds	ZHAW
Boni García	URJC
Nikolaos Stavros Gavalas	RELATIONAL
Felipe Gorostiaga	IMDEA
Francisco Gortázar	URJC
Varun Gowtham	TUB
Pablo Chico de Guzman	IMDEA
Piyush Harsh	ZHAW
Magdalena Kacmajor	IBM
Francesca Lonetti	CNR
Enric Pages	ATOS
Michael Pauls	TUB
David Rojo	ATOS
Cesar Sanchez	IMDEA
Avinash Sudhodanan	IMDEA
Guiomar Tuñon	NAEVATEC

Version history

Version	Date	Author(s)	Description of changes
0.0	15-02-2018	E.Pages M. Gallego	Initial ToC and content
0.1	15-02-2018	E. Pages M. Gallego V. Gowtham	Revised ToC and initial content
0.2	23-05-2018	M. Gallego V. Gowtham	Add component descriptions Add user requirement list Revision and update of use cases section
0.4	05-06-2018	V. Gowtham	Update user requirements table
0.5	05-06-2018	V. Gowtham	Add list of contributors Edit the document
0.6	13-06-2018	E. Pages M. Gallego	Update user requirements table
0.6	13-06-2018	V. Gowtham	Modifiy introduction
0.7	22-06-2018	F. Gortázar	Modify introduction
0.8	26-06-2018	V. Gowtham	Change as per peer review and editing
0.8	26-06-2018	ALL	Internal review
0.9	27-06-2018	V. Gowtham	Reformat the report
0.10	28-06-2018	E. Pages	Modify architecture overview section
1.0	29-06-2018	V. Gowtham	Reorganizing content and editing Final version preparation

Table of contents

List of figures

List of tables

Glossary of acronyms

Acronym	Definition
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), PaaS (Platform as a Service) and SaaS (Software as a Service) MaasS (Mobile as a Service) Baas (Browser as a Service)	This refers to different models of exposing cloud capabilities and services to third parties.
Instrumentation	This refers to extending the interface exposed by a software system for achieving enhanced controllability (i.e. the ability to modify behavior and runtime status) and observability (i.e. the ability to infer information about the runtime internal state of the system).
QoS (Quality of Service) and QoE (Quality of Experience)	In this proposal, QoS and QoE refer to nonfunctional attributes of systems. QoS is related to objective quality metrics such as latency or packet loss. QoE is related to the subjective quality perception of users. In ElasTest, QoS and QoE are particularly important for the characterization of multimedia systems and applications through custom metrics.
SiL (Systems in the Large)	A SiL 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 (Systems in the Small)	SiS are systems basing on monolithic (i.e. non distributed) architectures. For us, a SiS can be seen as 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)	Is an ElasTest functional component that abstracts and exposes to testers the capabilities of the ElasTest orchestration and recommendation engines.
TJob (Testing Job)	We define a TJob 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 T-Jobs. T-Jobs may have different flavors 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 (Test in the Large)	A TiL 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 T-Job.
Test Support Service (TSS)	We define a TSS as a tool which aides in the preparation of tests in different contexts.
ETM (ElasTest Tests Manager)	A core component of ElasTest.
EPM (ElasTest Platform Manager)	A core component of ElasTest.
ESM (ElasTest Service Manager)	A core component of ElasTest.
EDM (ElasTest Data Manager)	A core component of ElasTest.
EIM (ElasTest Instrumentation Manager)	A core component of ElasTest.
ECE (ElasTest Cost Engine)	A test engine provided by ElasTest.
ERE (ElasTest Recommendation Engine)	A test engine provided by ElasTest.
EQE (ElasTest Question & Answer Engine)	A test engine provided by ElasTest.
EOE (ElasTest Orchestration Engine)	A test engine provided by ElasTest.
EUS (ElasTest User Emulator Serice)	A test support service provided by ElasTest.

EDS (ElasTest Device Emulator Service)	A test support service provided by ElasTest.
ESS (ElasTest Security Service)	A test support service provided by ElasTest.
EBS (ElasTest Big-Data Service)	A test support service provided by ElasTest.
EMS (ElasTest monitoring Service)	A test support service provided by ElasTest.
ET (ElasTest ToolBox)	A toolbox provided for integration of ElasTest with external tools.
EJ (ElasTest Jenkins Plugin)	A plugin provided for integration of ElasTest with external tools.
CRUD (Create Read Update Delete)	Standard operations that can be performed on/to a software.
DoA (Description of Action)	A document which lists and described the actions to be performed in a project.
FMC (Fundamental Modelling Concepts)	Framework that provides comprehensive description of software intensive systems.
UML (Unified Modelling Language)	A general purpose, developmental, modelling language in the field of software engineering.
AWS (Amazon Web Services)	A cloud services platform.
AAA (Authentication, authorization and accounting)	Is a framework for intelligently controlling access to computer resources, enforcing policies, auditing usage and providing the information to bill for services.
API (Application Programming Interface)	A set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.
UI (User Interface) GUI (Graphical User Interface)	The UI is an information device which a user can use to interact with a machine. Similarly, GUI is a type of UI that allows users to interact with electronic devices through graphical icons and visual indicators.
OAI (Open API Initiative)	An effort to standardise the description of REST API.

1 Executive summary

ElasTest is a cloud platform designed for helping developers to test and validate large software systems while maintaining compatibility with current continuous integration practices and tools. The platform embraces a microservice like architecture, collectively providing facilities for the tester to deploy testing processes as separate entities. A combination of such testing processes can be leveraged or reused to form a larger testing process which counters the monolithic testing approach.

In this deliverable, we outline the efforts invested as part of task 2.2 and task 2.3 of WP2. The document further describes in the following mentioned sequence:

- The high level use cases with tester as the main user.
- The high-level architecture of ElasTest.
- The efforts invested in collecting user requirements.
- The high-level architecture and description of each component available in ElasTest.

This deliverable focuses on providing the reader an overview of ElasTest. A more detailed understanding of the components is detailed in the component's corresponding work package deliverable. This document in its current form is the first version at month 18.

This deliverable refers to other deliverables D2.2, D3.1, D4.1, D4.2 D5.1, D6.1 and D6.2.

2 Introduction

Nowadays, complex large software systems are proliferating due to the commodity of cloud and the need of elastic applications which pushes developers towards resilient software architectures like microservices. In this project, we concentrate on testing large software systems (i.e. SiL) created by the orchestration of simple components (i.e. SiS). Typically, those software systems are validated using CI tools and methodologies. This approach provides some minimal guarantees in relation to the correctness of the functional properties of the software, but it has very relevant limitations when evaluating other attributes of a software system in real production environments. For example, whenever developers want to validate non-functional features such as scalability, fault-tolerance or data consistency; they need to create complex testing architectures customizing the cloud orchestration mechanisms and managing test scalability by themselves. Things become even more complex when trying to reproduce real-world operational conditions. For example, tasks such as finding out how the system performs in real-networks (e.g. congestion, packet loss, latency, etc.) or evaluating how latency and other QoE parameters degrade with the number of users are relevant challenges. This becomes even more complex when systems manage special types of traffic such as sensor data or multimedia communications, which may follow complex binary protocols with real-time requirements and where the evaluation of QoS and QoE requires complex data processing.

ElasTest is an elastic cloud platform designed for helping developers to test and validate SiL (see definitions above), while maintaining compatibility with current CI practices and tools. For this, ElasTest bases on three principles:

- **Instrumentation of SuT:** ElasTest offers the facility to instrument the SuT based on the tester requirements. Such SuTs can be deployed on a native machine or on a cloud.
- **Test Orchestration:** ElasTest provides the facility to orchestrate one or more TJobs that assess the SuT. The orchestration is at the heart of the platform able to apply novel techniques to form Test in Large (TiL) as a combination of TJobs. Furthermore, it exploits the reuse of TJobs.
- **Test Recommendation:** To ease the tester's job, ElasTest offers a novel solution of recommending tests to a user. This feature optimizes the tester productivity.

These principles are complemented by a set of tools aimed at supporting testing on different contexts:

- Browsers as a service, for UI testing
- Emulators and actuators as a service, for testing of IoT applications
- Security as a service, for assessing the security properties of large software systems
- Monitoring as a service, for providing dynamic probes in a domain specific language capable of capturing the high level behaviour of the system and raising alarms
- Big Data as a service, for capturing and processing all the data of the different services

2.1 Core concepts and design principles

The microservice in the context of ElasTest and the rest of the document is referred to as component, this is due to the fact that we do not follow the microservice architecture closely, and favor flexibility over formality. The platform is dynamic in nature in which the composition of the platform depends on which components that are active at any given time depending on the testing process (TJob) that is running.

The nomenclature of components and the relevance to ElasTest is detailed in Section 4. In this subsection we describe the basic principles used when designing the individual components of ElasTest.

Following requirements are set forth for each component:

1. Define the features offered by the component.
2. Define the communication mechanism to interact with the component.
3. Define the component lifecycle.
4. Define how the component is maintained and used.

The following text shows how each requirement is addressed in the context of ElasTest.

1. Define the features offered by the component:
   Each component acts as a standalone entity providing certain features. The implementation details of the features is left to the component such that the

details of implementation is contained in the component. The said requirement equally applies to TJobs, where it provides the freedom to the tester to choose any method of implementation suitable to implement the features of the TJobs. The component may further use internal components also following a microservice like architecture.

2. Define the communication mechanism to interact with the component:
   The component is responsible to expose the features contained in it to the outside world. The features are exposed using a RESTful API interface. This interface offers the facility to create, read, modify and update the feature or resources exposed by the component. To have a common understanding, the OpenAPI Initiative Specification (OAIS)[1] is used by all the components to define and implement the interface such that not only other components in ElasTest but also the tester is able to interact with the component with ease. Each component therefore documents the features exposed using the OAIS and makes it available to the tester.
3. Define the component lifecycle:
   The component lifecycle points to duration time of time when the features of the component are required while using ElasTest. The lifecycle of each component is clearly understood and it is by this method there is an understanding with other components when a resource or feature of a component is made available. Certain components of ElasTest are alive during the time ElasTest is running when the user starts the platform. Certain other components are born when a TJob is started and terminated when TJob finishes while some other come alive on the demand of the tester.
4. Define how the component is maintained and used.
   Each component is a software implementation. Keeping up with the pace of software development is at the core of ElasTest. For this CI methods are used such that the components are kept up to date. Furthermore, mechanisms are provided in ElasTest to update the platform and also understand the status of each component.

The focus of this deliverable is to provide the reader a high level overview of ElasTest by documenting the combined efforts of Tasks 2.2 and T2.3 as obtained from DoA [1].

It is important to note that, this deliverable presents an overview of ElasTest in general and in particular, each component is presented in high level detail. A deeper understanding of each component can be obtained from the component's corresponding work package deliverables D3.1 [3], D4.1 [4], D4.2 [5], D5.1 [6] and D6.2 [8].

2.2 Structure of the document

The deliverable explains the efforts by first presenting the high level use cases (in Section 3). In the next part of the document (Section 4), the conceptual diagram of ElasTest is discussed followed by an explanation of overall architecture diagram of ElasTest. In the same section, the user requirements targeted by each component of

[1] https://www.openapis.org/about

ElasTest is presented. The rest of the document (Sections 5, 6, 7 and 8) is dedicated to presentation of the high level descriptions of each component. Section 9 concludes this document outlining the future work.

2.3 Target audiences

This deliverable is relevant for any developer or tester in general but not restricted. Furthermore, this deliverable can be useful for stakeholders in the management as well as in the academia.

3 Use Cases

A TJob that is to be executed has to undergo one of the CRUD (create, read, update and delete) operations by the user. Furthermore, a SuT associated with a TJob also needs to undergo CRUD operations from the user. Considering the interaction between ElasTest and TJob a set of high level use cases can be derived. Complementing to the CRUD operations are the recommendation facilities embedded into ElasTest and the cost of running a TJob.

ElasTest is a platform for executing tests. For that reason, the main user is the tester. ElasTest also can be operated by an administrator, but the use cases for the administrator are considered secondary and are not described here.

We identify the following main use cases:

1. UC1 - Define a SuT deployed by ElasTest.
2. UC2 - Define a SuT deployed outside ElasTest.
3. UC3 - Define a TJob and execute it.
4. UC4 - Define a TJob using a TSS and execute it.
5. UC5 - Inspect TJob information after execution.
6. UC6 - Define a TiL and execute it.
7. UC7 - Ask for test recommendations.
8. UC8 - Calculate TJob costs.
9. UC9 - Define a TJob and execute it from Jenkins.

3.1 Define a SuT deployed by ElasTest

The actions needed to define a SuT deployed by ElasTest are described below:

Actions

1. Tester will create a new Project or select an existing one.
2. Tester will create a SuT description. SuT description will have the following information:
   a. Basic information: Name and description
   b. How SUT is going to be deployed by ElasTest:
      - **With Commands Container:** The tester will specify a container image and a sequence of commands written as a bash script that will be executed to deploy the SuT.
      - **With Docker image:** The tester will specify only a docker image corresponding to the SuT.

- **With Docker Compose:** The tester will specify a docker-compose.yml file contents to deploy the SuT.

c. if it is necessary to wait for a http port to be available before tests can be executed.
d. Parameters with default values (that will be prompted each time the SuT is deployed in the context of a TJob).

3.2 Define a SuT deployed outside ElasTest

The actions needed to define a SUT already deployed outside ElasTest are described below:

Actions

1. Tester will create a new Project or select an existing one.
2. Tester will create a SuT description. SuT description will have the following information:
   a. Basic information: Name and description
   b. With a SUT already deployed, the instrumentation can be configured in the following ways:
      - **No instrumentation:** No information for SuT will be gathered during TJob execution.
      - **Instrumented by ElasTest:** ElasTest use EIM to instrument the SuT. All configuration needed to connect to SuT and install and configure instrumentation agents have to be provided.
      - **Manual instrumentation:** ElasTest shows in the Graphical User Interface (GUI), the information needed by the tester to properly configure the instrumentation agents in the SuT.

3.3 Define a TJob with SuT and execute it

The actions needed to define a TJob associated to a SuT and execute it are described below:

Actions

1. Tester will implement some tests for a SuT using a testing library in some programming language. For example, she will use JUnit in Java. This test will interact with the SuT in some way. For example using a REST API.
2. Tester will upload the tests to some source code repository.
3. Tester will create a new Project or select an existing one.
4. Tester will create a TJob in ElasTest indicating:
   a. Basic information: Name
   b. What SuT should be tested in that TJob
   c. How to obtain and execute the tests to be executed. This is defined with:
      - Commands to download tests code, compile it and execute it.
      - Environment docker image to execute the commands.
5. Tester will execute the TJob using ElasTest Web interface.

a. ElasTest will execute the SuT (if necessary) and will wait until ready (if configured so).
b. ElasTest will execute the tests (executing the specified commands). Tests will exercise SuT by means of some network protocol.
c. When tests are finished, ElasTest will shutdown the SuT if necessary.

6. During TJob execution, tester can see the following information gathered from Tests and SuT execution in real time:
    a. TJob: logs, CPU, memory, IO consumption and other information provided by ElasTest services.
    b. SuT: logs, CPU, memory, IO and other. If SuT is composed by several components (for example, having different containers deployed with docker-compose) the tester will see metrics of every part of the system.

Notes:

A TJob can be defined without a SuT associated to it in any way. The main difference is that ElasTest won't wait for SuT to be ready and, of course, no information will be displayed and recorded for it.

3.4 Define a TJob using TSS and execute it

The actions needed to define a TJob using a TSS and execute it are described below:

Actions

1. Tester will implement some tests for a SuT using a testing library in some programming language. For example, she will use JUnit in Java. Also, the test code will control some of the ElasTest Test Support Services (TSSs) by means of a remote interface. For example, the tests can exercise a web application SuT using the browsers provided by the ElasTest User impersonation Service (EUS).
2. Tester will upload the tests to some source code repository.
3. Tester will create a new Project or select an existing one.
4. Tester will create a TJob in ElasTest indicating:
    a. Basic information: Name
    b. What SuT should be tested in that TJob
    c. How to obtain and execute the tests to be executed.
    d. What TSSs are needed to execute the test. For example, the EUS service.
5. Tester will execute the TJob using ElasTest Web interface.
    a. ElasTest will start all TSSs specified by the user in the TJob configuration.
    b. ElasTest will execute the SuT (if necessary) and will wait until ready (if configured so).
    c. ElasTest will execute the tests (executing the specified commands). Tests will use TSSs by means of a remote protocol. For example, tests can request to EUS service some web browsers to interact with a web application SuT.
    d. When tests are finished, ElasTest will shutdown TSSs and the SuT if necessary.
6. During TJob execution, tester can see the information gathered from Tests and SuT execution in real time. It also can see the information provided by the TSSs used. For example, the browsers provided by EUS can be displayed in real time when tests are being executed.

3.5 Inspect TJob information after execution

The actions needed to inspect the information gathered during a past execution of a TJob are described below:

Actions

1. Tester will navigate through the web interface to find some past TJob execution.
2. Then screen showing past TJob executions has the following information:
   a. Every test executed has the following information associated to it:
      - Test class
      - Test name
      - Execution duration
      - Execution status: ERROR, FAIL or SUCCCESS
      - Files generated during test execution
      - Logs from test, SuT and other TSS (for example browsers console) generated during the test execution
   b. Logs can be inspected using the Log Analyzer tool, a powerful tool for filtering, marking, comparing, etc.

3.6 Define a TiL and execute it

The actions needed to orchestrate a set of TJob creating a TilL are described below:

Actions:

1. Tester will create a pipeline job in Jenkins with ElasTest plugin installed and configured
2. Tester will define a TiL with one of the following operators for combining TJobs:
   a. Sequence operator: It runs a TJob after another.
   b. Parallel operator: It runs several TJobs in parallel and emit common verdict of all of them based on the configuration (AND or OR).
3. Tester will execute the Jenkins job and ElasTest plugin will execute Jobs as defined in the TiL.

3.7 Ask for test recommendations

ElasTest Recommendation Engine (ERE) can be used by testers to ask how a given feature should be tested. ERE is trained with features and their corresponding tests. In that way, when a feature is presented to ERE, it can return tests used to test similar features in the projects used the train ERE's model.

The actions needed to use ERE in such way are:

Actions

1. Tester will navigate to ERE screen (instantiating ERE if necessary)
2. Tester will select the data set used for recommendations
3. Tester will ask for a test given a feature specified in natural language
4. ERE will show several test implementations with a confidence score associated to each one.

3.8 Calculate TJob costs

ElasTest Cost Engine (ECE) can be used by testers to know how much TJobs can cost (before they are executed) or how much they really cost (after their execution). To do that, the resources used by TJobs (basically platform resources and test support services) have a cost associated to them.

The actions performed to calculate TJob costs using ECE are the following:

Actions

1. Tester will navigate to ECE screen (instantiating ECE if necessary) and the list of the TJobs will appear.
2. Tester will select one TJob and the cost information will be shown

3.9 Define a TJob and execute it from Jenkins

ElasTest Jenkins Plugin (EJ) can be used to integrate ElasTest with Jenkins. The main features provided by EJ are: a) to send logs and tests results from Jenkins jobs to ElasTest (for later inspection) and b) to use ElasTest TSS from tests executed inside Jenkins jobs. Assuming a Jenkins installation with the EJ properly installed and configured, it can be used with the following actions:

Actions

1. Tester will create a pipeline Job in Jenkins
2. Tester will create a job wrapping all job steps into elastest(){...} syntax. This will send Job logs to ElasTest.
3. Tester will include as parameter the list of TSS to be used by the job tests.
4. When Jenkins job is executed, ElasTest Jenkins plugin will request to ElasTest to instantiate the required TSSs and provide the test code the remote endpoints where that TSSs will be available.
5. When the job have finished. EJ will undeploy the TSSs and the test results will be sent from Jenkins to ElasTest.
6. Tester will use a link provided in the Jenkins job execution page to open an ElasTest screen that shows the logs and metrics gathered during job execution.

4 Architecture Overview

4.1 ElasTest Platform (Functional Architecture)

ElasTest is a cloud platform designed for helping developers to test and validate large software systems while maintaining compatibility with current continuous integration practices and tools. For this, ElasTest bases on three principles:

1) Instrumentation of the software under test through observability and controllability agents so that it reproduces real-world operational behavior.
2) Test orchestration combining intelligently testing units for creating a more complete test suite.
3) Test recommendation using machine learning and cognitive computing techniques for recommending testing actions and providing testers with

friendly interactive facilities for decision making.

ElasTest enables developers to test large software systems through complex test suites created by orchestrating simple testing units (so-called TJobs). This orchestration mechanism is one of the main novelties of the ElasTest project and its precise conception, formalization and consolidation is one of our main research objectives. From the perspective of the tester, a TJob is software that, upon execution, performs some testing actions against the software under test. From this perspective, the TJob is a "testing unit". In order of not to constrain the freedom and flexibility of the tester, we do not assume any kind of property for the TJob neither from the technological (i.e. language, framework, etc.) nor from the semantics perspective (i.e. model, behavior, etc.) Our only assumption is that the TJob accepts some input parameters and that, upon execution, generates an outcome (i.e. output parameters). The expected values of such outcome constitute the TJob oracle.

The conceptual representation of the ElasTest architecture is shown in Figure 1. This conceptual representation, created at the time of the proposal, was the starting point of our architecture design. It consists of a number of software modules that testers can install into public or private clouds.

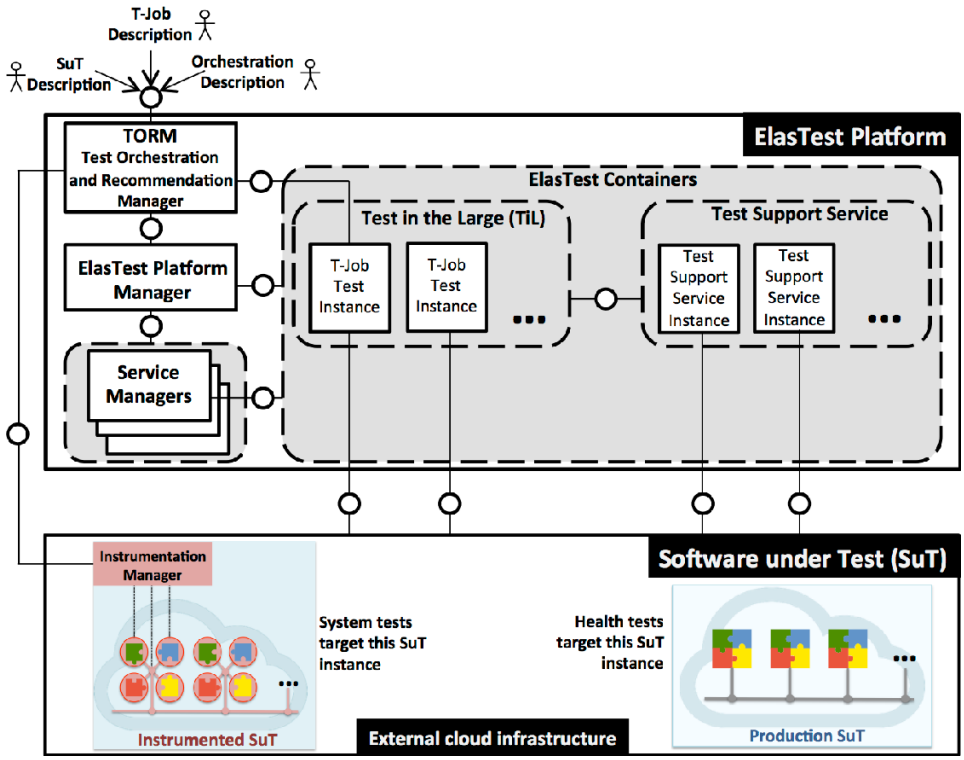

Figure 1. Conceptual representation of the ElasTest architecture and its relation with the SuT.

A more detailed overview of the functional architecture of the ElasTest platform is shown in Figure 2. The intermediate architecture design presented within this document has been produced after accomplishing the procedures defined in the methodology described in section 4.3 of this report. Both figures, the conceptual

representation and the functional architecture overview are based on the Fundamental Model Concept (FMC) which primarily provides a framework for the comprehensive description of software-intensive systems. It is based on a precise terminology and supported by a graphical notation which can be easily understood. In order to know how to interpret the block diagrams and their communications, please refer to the FMC cheat sheet [9].

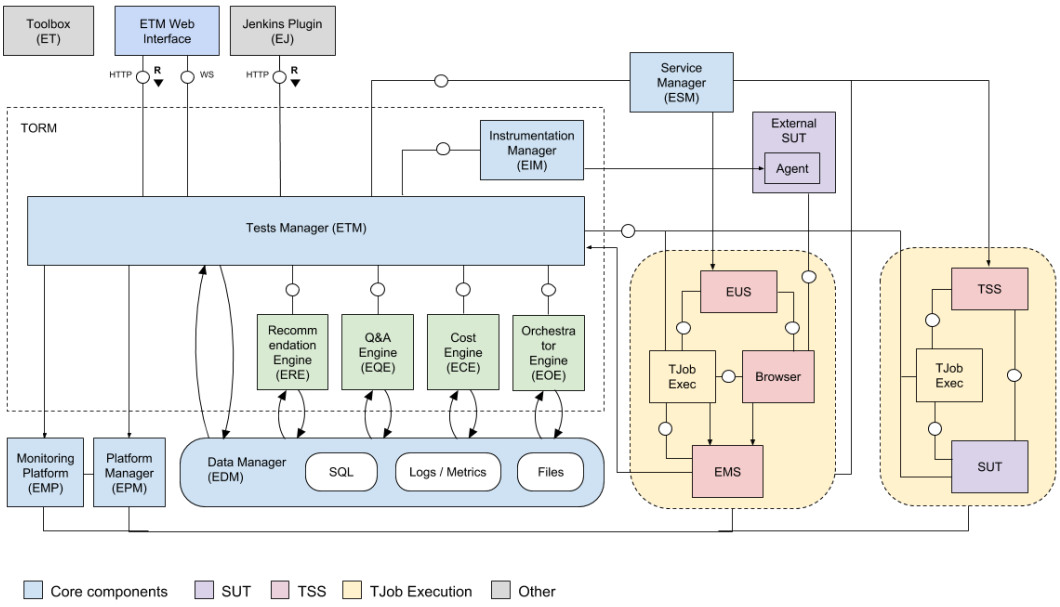

Figure 2. Functional architecture overview of the ElasTest platform.

Table 1 illustrates the building blocks of the ElasTest system; the individual software components of the platform maps with the blocks depicted in the aforementioned figure, each of them constitutes a fine-grained SOA.

We envisaged the ElasTest platform to be implemented as a distributed and scalable system, which allows the testing of large software systems created by the orchestration of simple components. Nowadays, those large systems are mainly validating the correctness of the software under evaluation using CI tools and DevOps methodologies among other available options. Despite this, very relevant limitations exist when we try to evaluate other attributes of the software such as non-functional features in real production environments or real-world operational conditions. ElasTest is a cloud platform designed for helping developers to test and validate large software systems while maintaining compatibility with current continuous integration practices and tools.

The resultant components are categorized as follows:

- ElasTest Core Components: These components constitute the enablers of the platform. They have the responsibility of providing management mechanism for the platform, the tests jobs and the software under evaluation.
- ElasTest Test Engines: The engines offers additional capabilities that can be used by the platform or the test support services, thanks to our modular architecture different engines may be plugged.

- ElasTest Test Support Service (TSS): These comprise reusable cloud services used to support the testing of the software under evaluation.
- ElasTest Integrations with External Tools: These comprise of the tools and plugins used for integration of ElasTest with external tools.

Table 1. Building blocks of ElasTest.

Component Name	Role
Core Components	
Test Manager (ETM)	It is the brain of ElasTest and the main entry point for developers.
Platform Manager (EPM)	It is the interface between ElasTest components and the cloud infrastructure.
Platform Monitoring (EMP)	It is a service that monitors the core modules of ElasTest platform.
Service Manger (ESM)	It delivers, on request/demand, service instances of particular service types.
Data Manager (EDM)	It provides the persistence layer services for all components.
Instrumentation Manager (EIM)	It controls and orchestrates the agents that are deployed on the software under evaluation.
Test Engines	
Cost Engine (ECE)	It estimates the cost to make developers cost aware of running a test.
Recommendation Engine (ERE)	It is a cognitive system designed to leverage recommendations based on learned knowledge.
Question & Answer Engine (EQE)	It accepts questions asked in natural language and tries to identify user's intentions and needs.
Orchestrator Engine (EOE)	It orchestrates and executed in coordination a set of test jobs for creating more complex test suite.
Test Support Services	
User Impersonation Service (EUS)	It is devoted to provide the mechanism for emulation of users in end-to-end tests.
Device Emulator Service (EDS)	It emulates devices used in Internet of Things (IoT) applications.
Security Service (ESS)	It facilitates the security testing of the software under evaluation.
Big-Data Service (EBS)	It provides a scalable computing engine based on

	big-data technologies
Monitoring Service (EMS)	It provides a monitoring service suitable for inspecting the execution of the software under evaluation.
ElasTest Integrations with External Tools	
ElasTest Jenkins plugin	It is devoted to provide the mechanism for using ElasTest via Jenkins CI system.
ElasTest Tools	This provides tools to install and configure ElasTest in the easiest way possible.

ElasTest 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.
One of the most repeated "*mantras*" since the beginning of the project is that we need to ensure to '*do not reinvent the wheel*' duplicating available systems that has already previously been created or optimized by others. Accordingly, at early stages of the design phase we have been working on the identification of available technologies and systems that we can extend or re-use, instead of starting from scratch, we evaluated the current support systems that we can adopt to complement the functionalities of our components. Figure 3 below shows the mapping between different technologies and the ElasTest building-blocks.

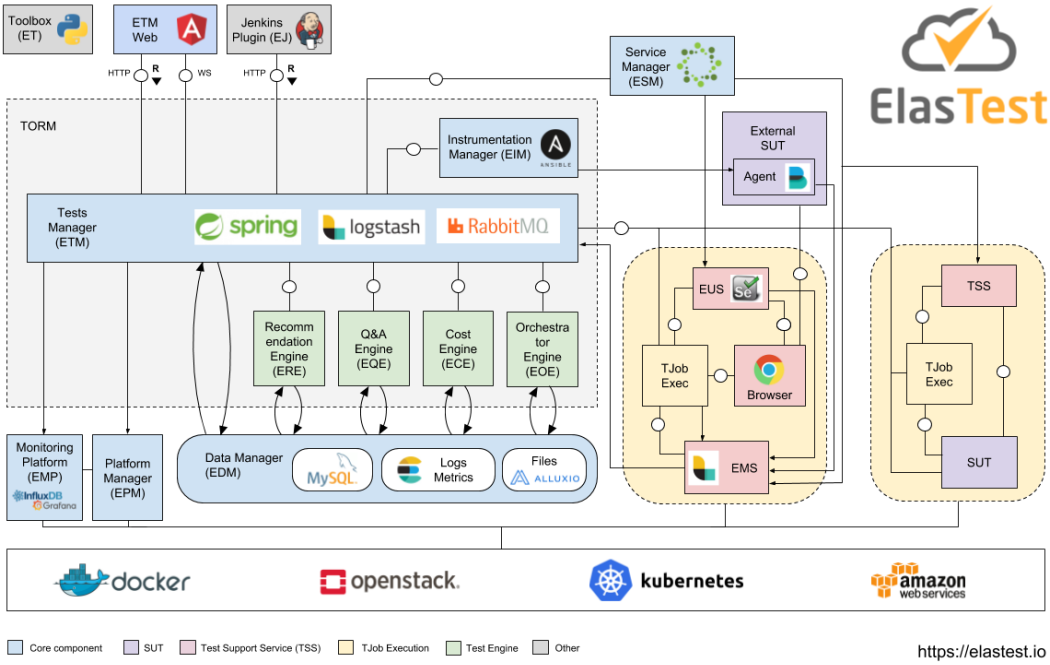

Figure 3. Architecture reference - support systems overview.

4.2 User Requirements

This subsection presents the user requirements and explains the methodology used to collate the requirements in the process of realizing ElasTest. The requirements were collected from each partner in the consortium with emphasis on their respective domain of expertise. Although testers and developers were recognized as end users, this may not restrict the reachability of ElasTest. ElasTest in general is potential candidate of interest for academia and industry which include various classes of end users.

The user of Elastest can belong to either of the below mentioned category:

- ElasTest user : A user who can use ElasTest as a normal user towards running a TJob and SuT associated with test support service/s.
- ElasTest admin: A user category who has privilege access to ElasTest. Certain features of ElasTest are made available only to admin. An admin can be considered as a privileged user.

At the time of review in month 18, the user roles of user and admin are not built into the platform.

The SMART criteria[2] was chosen to document the requirements targeting the end users of ElasTest who were identified to be testers and developers. This criteria helps in setting goals, by defining objectives with clear milestones and an estimation of accomplishing the goal.

The user requirements are listed as user story in the following format:

As a *<type of user>* , I want *<some goal>* so that *<some reason>*.

Once the user story was defined for a requirement, it was treated as a goal and tracked in the project for it's availability. User requirements collected were subdivided based on the four categories of components as discussed in previous subsection. The requirements thus collected are listed in the rows of separate tables dedicated to each category. In general the columns of table can be understood as follows:

- **Column 1 (ID):** Is the unique ID of a requirement associated in the corresponding row of the table.
- **Column 2 (Component):** Shows the component name which is proposing the requirement. The abbreviation of each component found in this column can be found in glossary of acronyms.
- **Columns 3 (Title):** Presents the title of the requirement.
- **Column 4, 5 and 6 (User Story):** Collectively documents the user story associated with the title listed in column 3.
    - Column 4 (User type) : Presents which user type is the requirement relevant to.
    - Column 5 (Goal) : Consolidates the goal of the requirement.
    - Column 6 (reason) : Explains the need for the requirement for the user listed in column 4.

[2] SMART Criteria, https://en.wikipedia.org/wiki/SMART_criteria

- **Column 7 (Status):** This field is dedicated to mark if the status of feature associated with the requirement having the following possibilities:
    - AVAILABLE : shows the feature is made available in ElasTest by the respective component.
    - BACKLOG : shows the feature is planned to be made available ElasTest.
- **Column 8 (Release):** This column tracks the release version of ElasTest, when the feature was first made available when the Status column is marked AVAILABLE. Else it signifies the future release version when the planned feature would be made available by the component in ElasTest, if the Status column is marked with BACKLOG. The methodology for defining a releases is discussed in subsection 4.3.

The table template discussed above helps us to document a requirement, by defining a milestone and verifying it.

As a result of efforts carried out in WP7, the project has received a list of requirements. One of the requirements was Test Link integration which was added as feature into ElasTest to facilitate the execution of one of the demonstrators. Furthermore, instrumentation with external agents instead of instrumenting code in ElasTest was another notable addition to the list of user requirements as a result of evolution of the platform.

The user requirements list collected until the month 18 of the project can be found in the following tables dedicated to each category of components:

- Table 2. User Requirements List - ElasTest Core Components.
- Table 3. User Requirements List - ElasTest Test Support Services.
- Table 4. User Requirements List - ElasTest Test Engines.
- Table 5. User Requirements List - ElasTest Integration with External Tools.

Table 2. User Requirements List - ElasTest Core Components.

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
ETM1	ETM	Manage projects	ElasTest user	(Create, Read, Update and Delete) CRUD operations on projects	I can create, edit, remove and update test projects to group TJobs and SuTs.	AVAILABLE	R1-R2
ETM2	ETM	Create SuTs	ElasTest user	to create SuTs	I can specify how to start a SuT with the following options: Deployed by ElasTest (Docker, Docker-compose commands) or Deployed Elsewhere.	AVAILABLE	R3
ETM3	ETM	Manage SuTs	ElasTest user	CRUD operations on SuTs	I can create, edit, remove and update SuTs.	AVAILABLE	R3
ETM4	ETM	Create TJobs	ElasTest user	to create TJobs	I can specify what SUT should be tested and how to execute tests against it.	AVAILABLE	R3
ETM5	ETM	Manage TJobs	ElasTest user	CRUD operations on TJobs	I can create, edit, remove and update TJobs.	AVAILABLE	R3
ETM6	ETM	Execute TJobs	ElasTest user	to execute a TJob	I can have logs, metrics and tests results recorded for further inspection.	AVAILABLE	R3
ETM7	ETM	Dashboard	ElasTest user	to see projects and last TJob executions in a single screen	I can have an overview of the status of the	AVAILABLE	R3

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
					platform.		
ETM8	ETM	Review TJob executions	ElasTest user	to review finished TJob executions	I can see what happened, especially in executions with failed tests.	AVAILABLE	R3
ETM9	ETM	Test Support Services	ElasTest user	to specify what TSSs must be ready to use when a TJob is executed	The tests in TJob can use selected TSS when testing the SuT.	AVAILABLE	R1-R2
ETM10	ETM	Log analyzer	ElasTest user	to analyse, filter and mark logs gathered during TJob execution	the problem troubleshooting is easier than looking to plain log.	AVAILABLE	R3
ETM11	ETM	Test case execution	ElasTest user	to review easily all information gathered during one specific test (logs, events and files)	I can focus on information related to a test (possibly failed).	AVAILABLE	R4
ETM12	ETM	TestLink info management	ElasTest user	to see TestLink projects, test cases, suites, builds and test plans in ElasTest interface	I can see that information integrated with other TJobs and projects.	AVAILABLE	R4
ETM13	ETM	TestLink Test plan execution	ElasTest user	to execute TestLink Test plans using browsers provided by ElasTest and recording all information from SuT and browsers	I can associate all that information to a bug report in case of test failure.	AVAILABLE	R4
ETM14	ETM	Test Engines	ElasTest user	to start, use and stop a Test Engine	I can start the engine only when needed	AVAILABLE	R4

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
ETM15	ETM	Show platform information	ElasTest admin	see the version and compilation date of ElasTest components	I can see if platform is updated or not.	AVAILABLE	R3
ETM16	ETM	Core components integration	ElasTest user	to see core components' GUI integrated in the main ElasTest GUI	I can see the platform integrated.	AVAILABLE	R4
ETM17	ETM	Show logs and metrics in realtime	ElasTest user	to see logs and metrics from SuT and Tests execution	I can know what happened with SuT and Tests in case I want to solve any problem.	AVAILABLE	R1-R2
ETM18	ETM	ElasTest micro	ElasTest user	a reduced version of ElasTest	I can try it with a very reduced resource requirements.	BACKLOG	R5
ESM1	ESM	Public catalog to allow end users to "install" and "unsinstall" services in a current ElasTest instance	ElasTest user	to install a TSS from a public catalog/registry	new TSSs can be installed in an ElasTest instance.	AVAILABLE	R4

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EIM1	EIM	Non-Intrusive	ElasTest user	agents as less intrusive as possible	I can produce low overhead of the instrumentation.	AVAILABLE	R1-R2
EIM2	EIM	Lightweight	ElasTest user	lightweight agents	it may need to be deployed within the SuT.	AVAILABLE	R1-R2
EIM3	EIM	Interoperability across OS distributions and version	ElasTest user	to maintain interoperability across different OS distributions	the agents should be designed to consume well established operating system interfaces to guarantee interoperability (at least on Linux systems).	AVAILABLE	R1-R2
EIM4	EIM	Allow instrumentation of AWS resources	ElasTest user	to instrument Amazon Web Services (AWS) resources using native AWS monitoring capabilities (CloudWatch)	I can inspect what happens in those resources from ElasTest when executing tests against that AWS SuT.	BACKLOG	R5-Final
EIM5	EIM	Allow instrumentation of OpenStack resources	ElasTest user	to instrument OpenStack resources (VMs, ObjectStorage, networking) using native OpenStack monitoring capabilities (Ceilometer)	I can inspect what happens in those resources from ElasTest when executing tests against that OpenStack SuT.	BACKLOG	R5-Final

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EIM6	EIM	Allow instrument Kubernetes resources	ElasTest user	to instrument kubernetes resources (Pods, services, containers, nodes) using native Kubernetes monitoring capabilities (Prometheus)	I can inspect what happens in those resources from ElasTest when execute tests against that Kubernetes SuT.	BACKLOG	R5-Final
EIM7	EIM	Provide fine grain configuration of Beats agents	ElasTest user	to configure the event stream names	I can recognize the events in ElasTest GUI.	BACKLOG	R4
EIM8	EIM	Controllability	ElasTest user	To control SuT	I can simulate real conditions (network bandwidth, failures, etc...).	BACKLOG	R4
EPM1	EPM	Providing Northbound API	ElasTest user	to interact with the EPM	the consumer can make use of the EPM via a ReSTful API.	AVAILABLE	R1-R2
EPM2	EPM	Providing SDKs	ElasTest user	to interact with the EPM	the developers of other components can easily integrate with the EPM by making use of SDKs (libraries) provided for different languages (e.g. java, python).	AVAILABLE	R1-R2

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EPM3	EPM	Instance lifecycle operations	ElasTest user	to be able to execute lifecycle operations such as start/stop, remove instances and retrieving information of the instance at runtime	the consumer of the EPM has full flexibility of executing lifecycle operations with the virtualized instances for a proper management at runtime.	AVAILABLE	R1-R2
EPM4	EPM	Instance management operations	ElasTest user	to be able to execute operations such as executing commands inside the instances and downloading/uploading files	the consumer of the EPM has full flexibility of accessing and interact with the virtualized instances.	AVAILABLE	R1-R2
EPM5	EPM	Platform - Linux support	ElasTest 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.	AVAILABLE	R3
EPM6	EPM	Platform - Mac support	ElasTest 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.	AVAILABLE	R1-R2
EPM7	EPM	Platform - Windows support	ElasTest user	to run the EPM in Windows with 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.	BACKLOG	R4

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EPM8	EPM	Log forwarding	ElasTest user	to forward logs to the configured endpoint	other parties can access those logs which can be used for further troubleshooting and debugging.	AVAILABLE	R1-R2
EPM9	EPM	Platform Elasticity	ElasTest user	elasticity provided by the EPM	either other ElasTest components can be scaled dynamically or the virtualized resources requested by other ElasTest components themselves.	BACKLOG	R5
EMP1	EMP	Monitoring spaces	ElasTest admin	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.	AVAILABLE	R1-R2
EMP2	EMP	Monitoring subspaces	ElasTest admin	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	AVAILABLE	R1-R2

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
					(possibly distributed).		
EMP3	EMP	API authentication and authorization	ElasTest admin	my access to be authenticated and properly logged for safety as well as auditing purposes	no one else is able to access the data streams from my services as they may contain sensitive data.	AVAILABLE	R1-R2
EMP4	EMP	Receive system metric streams	ElasTest admin	to be able to send relevant system metrics into the monitoring system	I can analyze data trends later or in real time.	AVAILABLE	R1-R2
EMP5	EMP	Persist system metric streams	ElasTest admin	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.	AVAILABLE	R1-R2
EMP6	EMP	Data query capability	ElasTest admin	to be able to see stored data points	I can analyze data trends and observe system trends.	AVAILABLE	R1-R2
EMP7	EMP	RESTful APIs	ElasTest admin	to easily integrate with the monitoring service with clearly defined interfaces	I can send metrics and perform control operations through my application code logic	AVAILABLE	R1-R2

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
					rather than interacting with the monitoring service in a standalone detached mode.		
EMP8	EMP	Availability of commonly used metric collectors (agents)	ElasTest admin	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.	AVAILABLE	R4
EMP9	EMP	Showing in ElasTest GUI, monitoring information of all components	ElasTest admin	to see all metrics and other monitoring information in ElasTest GUI	I can know the status of the system.	BACKLOG	R5

Table 3. User Requirements List - ElasTest Test Support Services.

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EMS1	EMS	Simple filtering rules	ElasTest user	to be able to add subscriptions so that EMS filters events according to conditions expressed in my subscriptions	I can better select which events to receive.	AVAILABLE	R4
EMS2	EMS	Deploy sampled-based and signal-based signals	ElasTest user	to extract a field from certain types of events to be considered as sampled values from a signal, which I want to reconstruct and work with.	I can build on it, aggregating its values and combining them with each other to synthesize useful information.	IN PROGRESS	R5
EMS3	EMS	Deploy correlation machines	ElasTest user	to create notification events based on received events, their relative timing and arrival and other contextual information	I can delegate to the EMS the finding of patterns of events in the stream of observations from the test.	IN PROGRESS	R5-Final
EMS4	EMS	Unsubscribe endpoints	ElasTest user	to stop the flow of events to a subscribed endpoint	I can remove it safely.	IN PROGRESS	R5
EMS5	EMS	Undeploy routing rules	ElasTest user	to stop applying certain rules of routing	I can regroup the events in new ways.	IN PROGRESS	R5
EMS6	EMS	Undeploy machines	ElasTest user	to stop executing certain monitoring machine	the system avoids computing data which I no longer need.	IN PROGRESS	R5

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EMS7	EMS	Subscribe websockets	ElasTest user	to receive events over a websocket	I can easily describe tests that guide the testing process depending on the observations of the current test.	AVAILABLE	R4
ESS1	ESS	Unprotected URL detection	ElasTest user	to receive list of unprotected URLs	I can protect them and prevent attackers from stealing sensitive information of the users of my web site.	AVAILABLE	R3
ESS2	ESS	Insecure cookie detection	ElasTest user	to receive list of insecure cookies (sensitive cookie values sent via http channel)	I can secure them and prevent attackers from impersonating the users of my web site.	AVAILABLE	R3
ESS3	ESS	Scanning for common vulnerabilities (unauthenticated)	ElasTest user	to be able to automatically test whether my Web Application is vulnerable to common Web Application security weaknesses.	I can avoid the situation in which malicious actors cannot easily hack my web site by exploiting the most common vulnerabilities.	AVAILABLE	R4

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
ESS4	ESS	Scanning for common vulnerabilities (unauthenticated)	ElasTest user	to do a deep security scan covering parts of my Web application that are otherwise difficult to be reached by automatic Web vulnerability scanners	I can avoid the situation in which common Web application vulnerabilities are not missed by my security scanner due to the reachability issue.	IN PROGRESS	R5
ESS5	ESS	Login Oracle Attack Detection	ElasTest user	to be able to detect privacy-related attacks against the users of my Web application	the privacy of my users are not compromised to malicious third parties.	IN PROGRESS	R5
ESS6	ESS	Replay Attack Detection	ElasTest user	to be able to prevent replay attacks against my shopping cart applications	malicious users cannot shop for free or shop by paying less by replaying payment tokens.	BACKLOG	R6-Final
ESS7	ESS	False positives Reduction	ElasTest user	to be able to see security vulnerabilities that are relevant for my Web applications	I don't have to waste time looking into security issues that are not relevant for my Web application.	BACKLOG	R6-Final
EUS1	EUS	W3C WebDriver compatibility	ElasTest user	to support standard W3C WebDriver API (based on JSON messages over REST)	EUS is backwards compatible with existing technologies such as Selenium and Appium.	AVAILABLE	R1-R2

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EUS2	EUS	Basic media evaluation	ElasTest user	to read audio level and RGB colors of given UI elements	it can be used as test oracle to feed test assertion.	BACKLOG	R6-Final
EUS3	EUS	Event subscription	ElasTest user	to subscribe to UI elements	tests can receive events notification.	AVAILABLE	R4
EUS4	EUS	Measure end-to-end latency of a WebRTC session	ElasTest user	to measure end-to-end latency	tests can know whether or not a WebRTC service has operational real-time performance rates	BACKLOG	R6-Final
EUS5	EUS	Measure quality (audio|video) of a WebRTC session	ElasTest user	to measure full-reference QoE indicators both for audio and video	tests can find out the quality of the WebRTC media in an easy way.	BACKLOG	R6-Final
EUS6	EUS	Remote control	ElasTest user	to monitor remote sessions for browsers and mobile	I can watch in real-time and interact with browser/movie sessions.	AVAILABLE	R1-R2
EUS7	EUS	WebRTC stats	ElasTest user	To read WebRTC statistics	I can test/read WebRTC QoS indicator in a seamless way.	AVAILABLE	R4
EUS8	EUS	Browser logging gathering	ElasTest user	to read browser logs	testers are aware of the underlying logging info to trace potential failures.	AVAILABLE	R3
EUS9	EUS	Mobile logging gathering	ElasTest user	to read mobile logs	testers are aware of the underlying logging info to trace potential failures	BACKLOG	R6-Final

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EDS1	EDS	Minimal orchestrator EDS	ElasTest user	to orchestrates sensors and actuators	the demonstrator can initiate and connect sensors and actuators and connect them with a logic.	AVAILABLE	R4
EDS2	EDS	Minimal EDS orchestration	ElasTest user	to call and initiate required sensors and actuators	the sensors and actuators are live and can provide data.	AVAILABLE	R4
EDS3	EDS	Demonstrator logic	ElasTest user	to connect sensors and actuators via logic	an IoT application can be realized.	AVAILABLE	R4
EDS4	EDS	Scalability of an application	ElasTest user	a combination of demonstrator applications	I can test an SiL.	AVAILABLE	R4
EDS5	EDS	Reusability of an application	ElasTest user	A combination of demonstrator applications	I can form an SiL as a combination of SiS.	AVAILABLE	R4
EDS6	EDS	Mechanisms for collecting QoS	ElasTest user	to collect QoS metrics	I can analyse QoS	BACKLOG	R6-Final
EDS7	EDS	Add basic set of sensors (about 7 types)	ElasTest user	to use various sensors	I can use in my IIoT application.	BACKLOG	R5
EDS8	EDS	Add basic set of actuators	ElasTest user	to use various actuators	in my IIoT application.	BACKLOG	R5

Table 4. User Requirements List - ElasTest Test Engines.

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EOE1	EOE	Topology generation	ElasTest user	to define some kind of test orchestration notation	users can define a TiL by aggregating different TJobs.	AVAILABLE	R5
EOE2	EOE	Jenkins DSL notation	ElasTest user	to leverage Jenkins shared library technology to create orchestration topology	users can define a TiL by aggregating different TJobs.	IN PROGRESS	R6-Final
ERE1	ERE	Data Preprocessing	ElasTest admin	to automatically preprocess user data	I can minimize time and effort spent on data preparation.	AVAILABLE	R1-R2
ERE2	ERE	Data Load	ElasTest admin	to upload user data to cloud	it can be fed to the machine learning model.	AVAILABLE	R1-R2
ERE3	ERE	Training on User Data	ElasTest admin	to launch the execution of ML algorithms	I can train machine learning model on user provided data.	AVAILABLE	R1-R2
ERE4	ERE	Flexible Storage	ElasTest admin	A flexible solution for storing user data	I can choose storage type that fits best the size of my datasets.	AVAILABLE	R3
ERE5	ERE	Authentication	ElasTest admin	to log in and authenticate as a registered user	I can get access to proprietary services.	AVAILABLE	R3
ERE6	ERE	Admin Dashboard	ElasTest admin	a separate role and UI for managing data load and training	I can ensure control over resource-consuming procedures.	AVAILABLE	R3

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
ERE7	ERE	Configure default settings	ElasTest user	to configure and save default settings for queries	I can choose the model that I want to query.	AVAILABLE	R3
ERE8	ERE	Tester UI	ElasTest user	a user interface	I can query for, view and interact with recommendations generated by ERE.	AVAILABLE	R1-R2
ERE9	ERE	Recommend TJobs to reuse	ElasTest user	based on a natural language descriptions, receive recommendations on automated test cases to reuse	I can increase code reusability, save time and effort.	AVAILABLE	R1-R2
ERE10	ERE	Recommend manual test cases to reuse	ElasTest user	based on functionality description, receive recommendation on manual test steps to reuse	I can increase knowledge reuse, improve test cases quality, support less experienced testers.	BACKLOG	R6-Final
ERE11	ERE	Recommend new TJobs for incoming features	ElasTest user	based on natural language description, receive newly generated code for automated test cases	I can save time and resources spent on test automation.	AVAILABLE	R4
ERE12	ERE	Learning from tester feedback	ElasTest user	a convenient way to amend received recommendations and return them to the system	The feedback is used for re-training to improve future recommendations.	BACKLOG	R6-Final
ERE13	ERE	Inline help	ElasTest user	to have access to inline help as I navigate through the UI	I can get immediate explanations and tips for using various features.	AVAILABLE	R4

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
ERE14	ERE	End-to-end preprocessing pipeline	ElasTest admin	to automatically crawl a software repository (Java) and extract relevant training data	I can eliminate manual effort required to gather and analyse data.	AVAILABLE	R5
ERE15	ERE	Pre-trained models	ElasTest admin	ERE to provide off-the-shelf pre-trained model that can be customized using my own data	I can leverage software engineering knowledge captured in large open source repositories, decrease training time and cost.	BACKLOG	R6-Final
EQE1	EQE	Data Preprocessing	ElasTest admin	to automatically preprocess user data	I can minimize manual effort on data preparation.	IN PROGRESS	R4
EQE2	EQE	Data Load	ElasTest admin	to load user data	it can be fed to a machine learning model.	IN PROGRESS	R4
EQE3	EQE	Training on user data	ElasTest admin	to launch training	I can generate a Q&A model trained on user data.	BACKLOG	R6-Final
EQE4	EQE	Interactive UI	ElasTest user	An interactive UI	I can easily input questions and read responses within the conversation flow.	BACKLOG	R6-Final
EQE5	EQE	Advices on efficient queries	ElasTest user	Ask the Q&A system how to query the recommender	I can formulate efficient queries.	BACKLOG	R6-Final

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EQE6	EQE	Learning about new test cases	ElasTest user	Ask Q&A about new test cases appropriate for a given project	Leverage testing knowledge captured in open software repositories.	BACKLOG	R6-Final

Table 5. User Requirements List - ElasTest Integration with External Tools.

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
ET1	ET	Install full latest version	ElasTest user	to install the latest full ElasTest	testers can configure SuTs and TJobs with all the capabilities.	BACKLOG	R6-Final
ET2	ET	Install lite latest version	ElasTest user	to install the latest lite ElasTest	testers can configure SuTs and TJobs with and use the basic capabilities of ElasTest.	BACKLOG	R6-Final
ET3	ET	Install full specific version	ElasTest user	To install a specific version of the full ElasTest	testers can configure SuTs and TJobs with all the capabilities.	BACKLOG	R6-Final
ET4	ET	Install lite specific version	ElasTest user	to install a specific version of the lite ElasTest	testers can configure SuTs and TJobs with and use the basic capabilities of ElasTest.	BACKLOG	R6-Final
ET5	ET	Check ElasTest Status	ElasTest user	to check the ElasTest status (running/stop/failed/unstable...)	the user can check if the platform is ready to be used.	AVAILABLE	R1-R2
ET6	ET	Start full latest version	ElasTest user	to start the latest full ElasTest docker image	testers can configure SuTs and TJobs with all the capabilities.	AVAILABLE	R4
ET7	ET	Start lite latest version	ElasTest user	to start the latest lite ElasTest	testers can configure SuTs and TJobs with and use the basic capabilities of ElasTest.	AVAILABLE	R1-R2

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
ET8	ET	Start full specific version	ElasTest user	to start a specific version of the full ElasTest	testers can configure SuTs and TJobs with all the capabilities.	AVAILABLE	R4
ET9	ET	Start full latest version without ERE	ElasTest user	to start the latest full ElasTest (without the ElastTest Recommendation Engine)	testers can configure SuTs and TJobs with all the capabilities except ElastTest Recommendation Engine.	AVAILABLE	R1-R2
ET10	ET	Start lite specific version	ElasTest user	to start a specific version of the lite ElasTest	testers can configure SuTs and TJobs with and use the basic capabilities of ElasTest.	AVAILABLE	R1-R2
ET11	ET	Retrieve connection information	ElasTest user	to know where to connect to access the ElasTest platform	users can connect to the Platform to operate.	AVAILABLE	R1-R2
ET12	ET	Retrieve information of deployed components	ElasTest user	to know which components are available in the running ElasTest	I can obtain information of each of the components, such as status, port, consuming resources etc.	BACKLOG	R3
ET13	ET	AWS Cloud Elastest Deployment	ElasTest user	to configure and Run an Elastest Instance in the AWS cloud, with no or little effort	I am able to have a fully operating ElasTest running in the cloud.	BACKLOG	R4

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
ET14	ET	AWS Cloud Elastic Elastest	ElasTest user	to configure and Run an Elastest that can be seamlessly elastic	I am able to launch (virtually) any number of TJobs as the resources would be elastic.	BACKLOG	R6-Final
EJ1	EJ	Install plugin	Tester (Jenkins User)	to install ElasTest Plugin with default plugin installer	ElasTest configuration properties can be set on Jenkins Configuration and ElasTest can be used in Jenkins Jobs.	IN PROGRESS	R5
EJ2	EJ	Install plugin for pipelines	Tester (Jenkins User)	to install ElasTest Plugin with default plugin installer	ElasTest configuration properties can be set on Jenkins Configuration and ElasTest can be used in Jenkins Pipelines.	IN PROGRESS	R5
EJ3	EJ	Global configuration of ElasTest platform	Tester (Jenkins User)	to configure global ElasTest settings: - Version - type (lite/full)	the plugin can manage the ElasTest platform with the appropriate configuration.	IN PROGRESS	R5
EJ4	EJ	Jenkins Managed ElasTest	Tester (Jenkins User)	ElasTest plugin to be able to launch a ElasTest with the specified global configuration	any job can be able to launch and use this ElasTest.	IN PROGRESS	R5

ID	Component	Title	As a <type of user>	I want <some goal>	so that <some reason>	Status	Release
EJ5	EJ	External Managed ElasTest	Tester (Jenkins User)	ElasTest plugin to be able to use an ElasTest running in other location (local or external)	any job can be able to use this ElasTest.	BACKLOG	R6-Final
EJ6	EJ	Configure docker Image SuT	Tester (Jenkins User)	to configure in a Job, ElasTest to recognize and launch a provided an image of a SuT	ElasTest can work with that SuT.	BACKLOG	R6-Final
EJ7	EJ	Configure externally hosted SuT	Tester (Jenkins User)	to configure in a Job, ElasTest to recognize an externally hosted SuT	ElasTest can work with that SuT.	BACKLOG	R6-Final
EJ8	EJ	Configure personalized SuT	Tester (Jenkins User)	to provide an script (mvn, sh, py...) that the ElasTest plugin will use to launch a SuT	ElasTest can work with that SuT.	BACKLOG	R6-Final
EJ9	EJ	Configure docker image TJob	Tester (Jenkins User)	to configure ElasTest to recognize and launch a provided an image of a TJob	ElasTest can execute that TJob.	BACKLOG	R6-Final
EJ10	EJ	Configure personalized TJob	Tester (Jenkins User)	to provide n script (mvn, sh, py...) that the ElasTest plugin will use to launch a TJob	ElasTest can execute that TJob.	BACKLOG	R6-Final
EJ11	EJ	Export results	Tester (Jenkins User)	to export the result of the tests executed in a readable format	I can read detailed results	BACKLOG	R6-Final
EJ12	EJ	Export logs	Tester (Jenkins User)	to export all the generated logs for SuT and TJobs	I can retrieve them for further operations.	BACKLOG	R6-Final

4.3 Methodology

Figure 4 shows the ElasTest agile methodology and its application on the work packages.

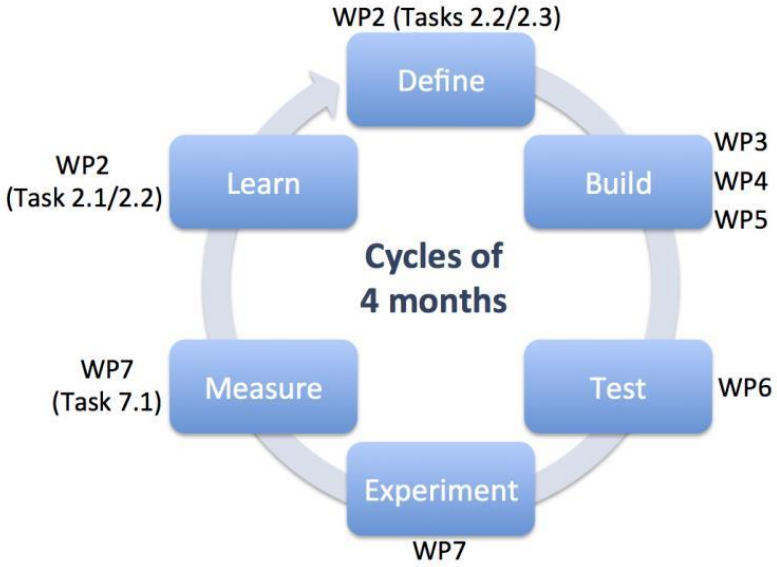

Figure 4. ElasTest work agile methodology

The agile methodology selected is based on executing incremental iterations on a *Build-Measure-Learn* feedback loop which validates that the implemented technology is valuable and responds to real needs. On every of these iterations, the built technologies are discussed between the technical WPs (WP3, WP4, WP5) and the vertical demonstrators (WP7) in order to refine their roadmap.

The cycles have a duration of 4 months, we call each of these cycles a *Release (R)*, a total of 9 releases have been planned during the project duration. The development tasks until R6 shall provide the platform validated in a lab, while the last 3 shall demonstrate it through the vertical demonstrators.

During the first cycle we have focused on developing, based on our initial component designs, proof and concepts as well as performing adaptations to the previous baseline technologies/components in order to identify and start solving the integration issues that the consortium were able to anticipate at this stage. Hence, in an initial stage most components only implements a small subset of the requirements depicted in section 4.2. Within the second cycle, and once the CI environment was ready, we have focused mainly on providing value to the platform users by implementing the required component functionalities while at the same time we organized regular meetings keeping an eye on the market and vertical demonstrators needs, in order to maintain the project aligned with the industry requirements. At month 18, we were able to cover four of these incremental cycles which include the release of the first integrated version of the platform (Milestone_5).

In order to achieve our goals, common conventions and approaches have been agreed within the consortium. You can find below the most relevant conventions:

- Fundamental Model Concept (FMC) has been selected to provide understandable block diagrams of the platform as well as for each of the submodules that constitutes the platform.

- Unified Modeling Language (UML) is the general-purpose modeling language selected to define the Data Model diagrams and Sequence diagrams across the components.
- Early discussions are promoted across technical WPs in order to ensure that all components have the same level of understanding on the platform. The discussions have been organized in the following small working groups:
    - Persistence Working Group
    - Monitoring Working Group
    - Test Management Working Group
    - Data Management Working Group
- To facilitate the communication across developers different Slack channels (#) are used in the project.
- For the fine-grained management of the previous and ongoing tasks each WP leader manages a Trello board.
- The platform is designed as a Service Oriented Infrastructure (SOI) where the direct interaction between software modules uses to be synchronous through REST APIs, however for certain cases the systems within ElasTest will be able to react asynchronously based on events forwarded by other modules or systems of the platform.
- The interactions and the information exchanged between components have been captured early during the design phase through the specification of the interfaces exposed by the components following the *OpenAPI* initiative.
- Software components releases follow Semantic Versioning approach which proposes a simple set of rules and requirements that dictate how version numbers are assigned and incremented.

4.4 Components Specification

Chapter 4 illustrates the functionality and interactions between the ElasTest software systems by means of sample use cases. Detailed specifications for the individual software modules are then provided in Sections 5 to 8.

A software module is in general characterized by its objectives, system prerequisites, technical requirements and interactions with other software modules:

- Objectives: Objectives are the capabilities that the software modules provide to the platform and/or to other software modules or systems within the architecture.
- System Prerequisites and Technical Requirements Specification: Each software module should fulfil certain requirements to be efficient and useful to the overall platform. Each of them may have basic prerequisites, or may require certain technologies or configuration data to specify system parameters or to build/adapt the software modules.
- Interactions among Software Modules: Each software module interacts with other modules by relying on inputs from and/or providing outputs to other modules. Exchanged information can be static or dynamic. The direct interaction between software modules uses to be synchronous through REST APIs, however for certain cases the systems within ElasTest will be able to react

asynchronously based on events forwarded by other modules or systems of the platform.

- Additional information: Besides the above three characteristics, the description of the software modules may contain further information about the specific implementation (such as, functions, existing implementations, possible adaptations and scalability of the modules).

5 ElasTest Core Components

These are the set of components needed to initialize the platform at the very beginning when the platform is started. More information can be derived from Table 1 in section 4. For an in depth understanding of the core components, the reader is referred to D3.1 [3] of work package 3.

5.1 ElasTest Tests Manager (ETM)

The ElasTest Tests Manager (ETM) is the brain of ElasTest and the main entry point for developers. ETM provides a web interface to be used by testers (main users) and administrators. It also provides a remote API currently used by ElasTest Jenkins Plugin (EJ). Other tools are also using this API to integrate with ElasTest Platform.

5.1.1 Objectives

The main objectives of ETM can be grouped in the following areas:

- **Test management:** The main use case of ElasTest is allow users to manage jobs to execute tests (TJobs). The tester can specify test jobs, execute them and analyze its results (in real time and after execution). The main information gathered during TJob's execution are test results, logs and metrics. ETM provides to the user a rich user interface specifically tailored to ease the management of that kind of information. For example, is worth noting the **Log Analyzer**, a part of ElasTest designed to inspect all logs gathered during test executions. All these features allows the tester discover more easily (compared with state of the art tools) the root of the problem when a test fails.
- **SUT management:** Testers are able to specify the Software under Test (SuT) associated to TJobs so that tests can be executed against it. SuT can be deployed by ElasTest (using Docker or Docker Compose) or can be already deployed outside ElasTest. In the latter case, ElasTest can instrument the SuT (by means of ElasTest Instrumentation Manager) or can be instrumented manually. The instrumentation is used to gather SuT information (metrics and logs) to provide observability.
- **Test Support Services:** Besides all features provided by ElasTest than can be used by testers in the web interface, ElasTest also provides services that can be managed programmatically from the test code. These services are called Test Support Services (TSS) and allows the tester to create powerful tests with ease. For example, one of the most used TSS when testing web applications is the ElasTest User Impersonation Service (EUS), that provides web browsers to

tests. Browsers, controlled by WebDriver protocol, can exercise the web application SUT and assert the expected result. When a TJob is specified in ElasTest, it defines what TSSs are needed. When a TJob is executed, ETM first instantiates required TSSs and then execute test code, providing to it the network endpoint to use those TSSs.

- **Test Engines:** ElasTest core functionality is provided by ETM and other core components like Data Manager (EDM), Platform Manager (EPM), etc. But this core functionality is augmented by means of so called Test Engines (TE). A Test Engine is a component that provides complementary features. ElasTest currently offers two engines: ElasTest Cost Engine (ECE) (responsible to manage the cost of TJob executions) and ElasTest Recommendation Engine (ERE) (that provides recommendations about tests to the user). More engines are also planned to be included in the following releases. ETM is responsible to manage the lifecycle of Test Engines. Specifically, ETM loads the engine when requested by the user (by delegating it to ESM), it embeds graphical user interface into the main ElasTest one, and also it provides the remote API to allow engines to access to all information about TJobs, executios, SuTs, etc.
- **Test execution comparison:** ElasTest provides monitoring services for the SuT. That is, a tester is able to know the CPU, memory and IO consumption of the SuT while the test is executing against it. In some cases, it is important to compare several executions of the same test against different SuT configurations. ElasTest will provide the feature of comparing the information gathered during the execution of related tests.

ETM is the controller of all ElasTest core features. It also exposes the public remote API for clients and the internal API needed for Test Engines and Test Support Services.

5.1.2 Systems Prerequisites and Technical Requirements specification

As the other ElasTest core components, ETM will be deployed as a docker container. In that sense, it is necessary a Docker engine to execute it. Also, it is needed some sort of coordination between all components to communicate each other. That management is provided by ElasTest Toolbox (ET).

5.1.3 Component Design

Figure 5 shows main ElasTest components in a FMC diagram. As you can see, ElasTest Test Orchestration and Recommendation Manager (TORM) is composed by ETM, the Test Engines (TEs) and the ElastTest Instrumentation Manager (EIM). ElasTest Tests Manager (ETM) is the entry point of ElasTest Platform and is the central part of it. ETM uses ElasTest Data Manager (EDM) to persist and query all kinds of information. For example, when a previous test execution is shown in the web interface, all information is loaded from data manager (files, logs, test status, etc). ETM uses ElasTest Platform Manager (EPM) to control the resources used in ElasTest to execute TJobs and internal SuTs.

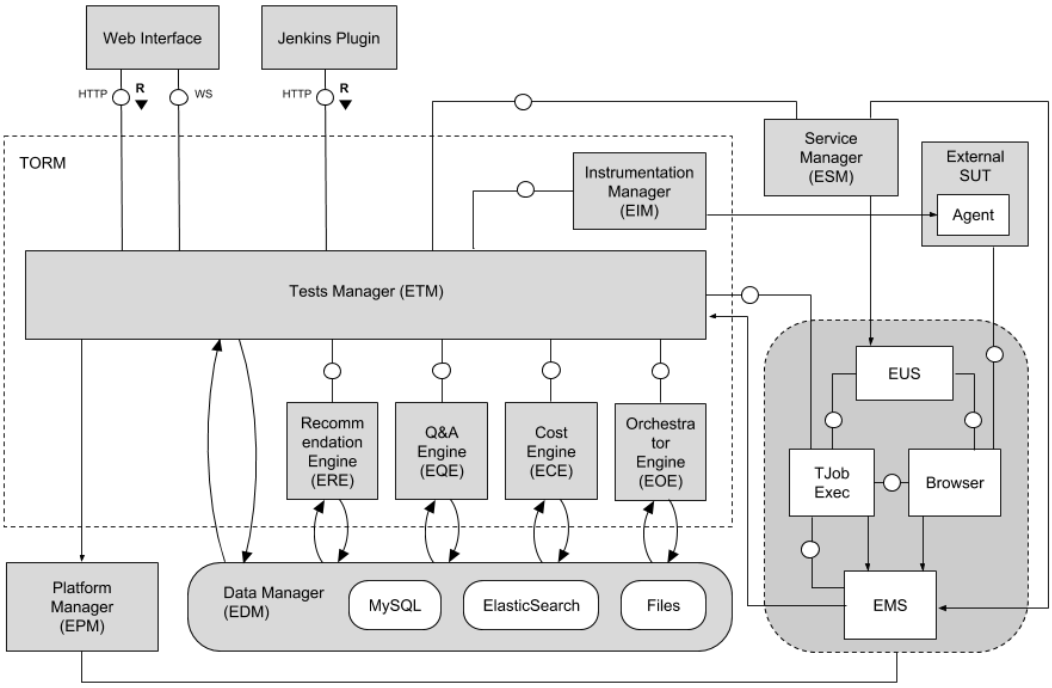

Figure 5. ETM FMC Diagram.

ETM remote API is used by external tools (like ElasTest Jenkins plugin) and also by other ElasTest components (for example, the ElasTest Cost Engine). Figure 6 shows the data model managed by ETM. A SuT definition is an instance of SuT entity. A SuT execution is an instance of SuTExecution entity. As a SuT can be executed multiple times, there is a one to many relation between SuT and SuTExecution. A TJob definition is an instance of TJob which can also be executed several times. Finally, the same SuT can be associated to multiple TJobs.

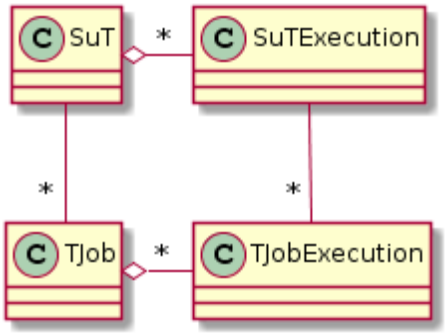

Figure 6. Main data model managed by ETM

As ETM is the main entry point of ElasTest features, it requires all other ElasTest core components to work properly. That is, this component requires all the following components:

- **Elastest Platform Manager (EPM):** The component that provides the ability to instantiate execution entities (like docker containers or virtual machines).
- **Elastest Service Manager (ESM):** The component that provides the ability to

instantiate Test Support Services (TSSs) and Test Engines (TEs). EMS uses the same underlying EPM to manage low level execution entities.

- **Elastest Data Manager (EDM):** The component that provides specialized persistence service to the rest of the platform.
- **Elastest Instrumentation Manager (EIM):** The component that allows ElasTest to instrumentalize already deployed SuT when tests are executed against it.

ElasTest architecture is inspired in a microservices architecture, in which every component is implemented as an independent process communicated with other components using remote protocols or APIs. Specifically, the following protocols are used:

- **EPM:** ETM interacts with EPM using a REST API. The API is defined for ElasTest project.
- **ESM:** ETM interacts with ESM using a REST API. The REST API is based on OSBA standard with extensions required by ElasTest.
- **EDM:** ETM interacts with persistence service provided by EDM using several protocols. Specifically, it uses native MySQL protocol to communicate with MySQL. It uses the ElasticSearch REST API to communicate with it. Also, the Logstash component, used by ETM, is connected with ElasticSearch by means of the native protocol between ElasticSearch and Logstash.
- **EIM:** ETM interacts with EIM using a REST API. The API is defined for ElasTest project.

The remote API published by ETM is divided in two parts: A REST API to manipulate the resources managed by ETM (TJobs, SuTs, executions…) and a WebSocket STOMP API to allow clients to receive realtime events from ElasTest.

Use cases

The main use case for ETM is to define a TJob and execute it as shown in Figure 7:

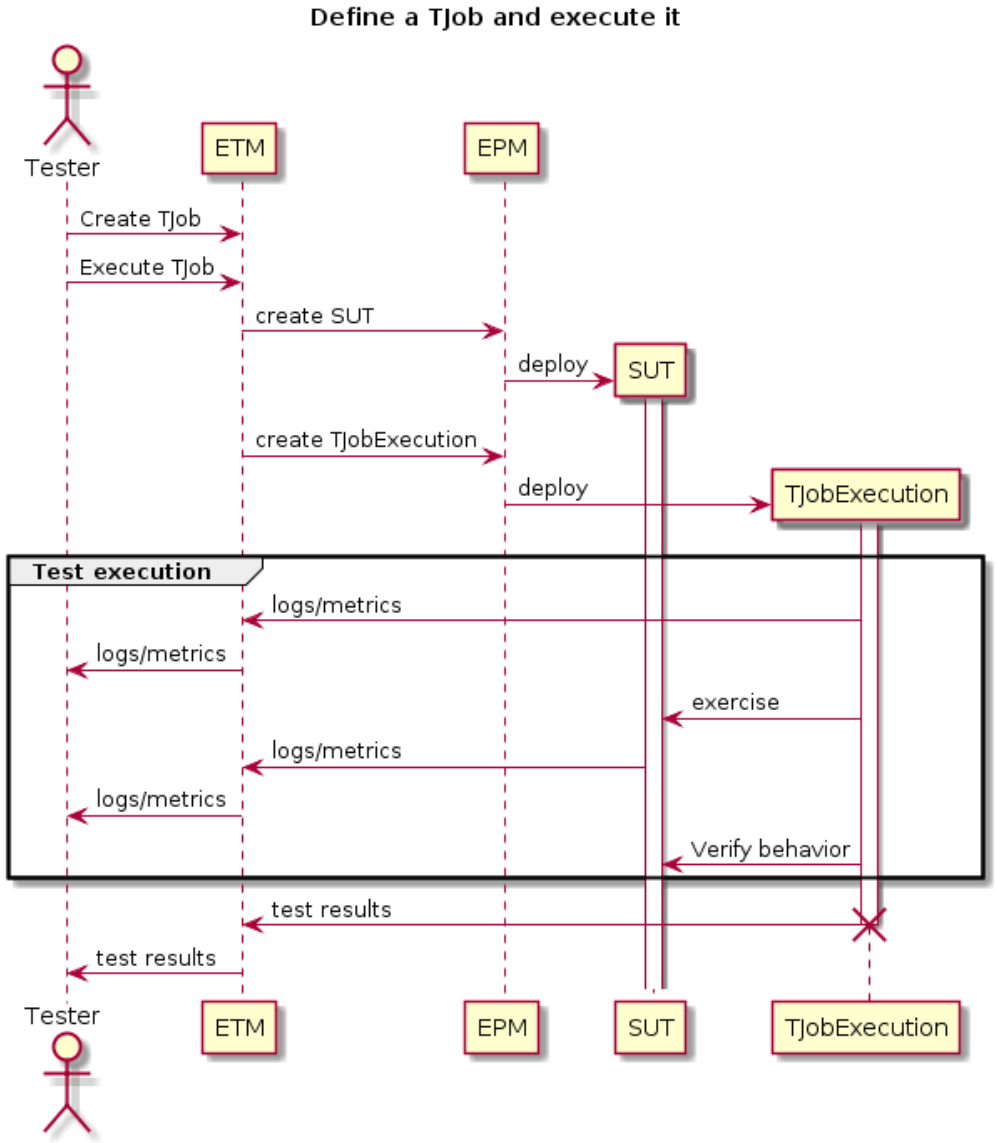

Figure 7. ETM use case - Define a TJob and execute it.

And also, using a TSS in the TJob as shown in Figure 8:

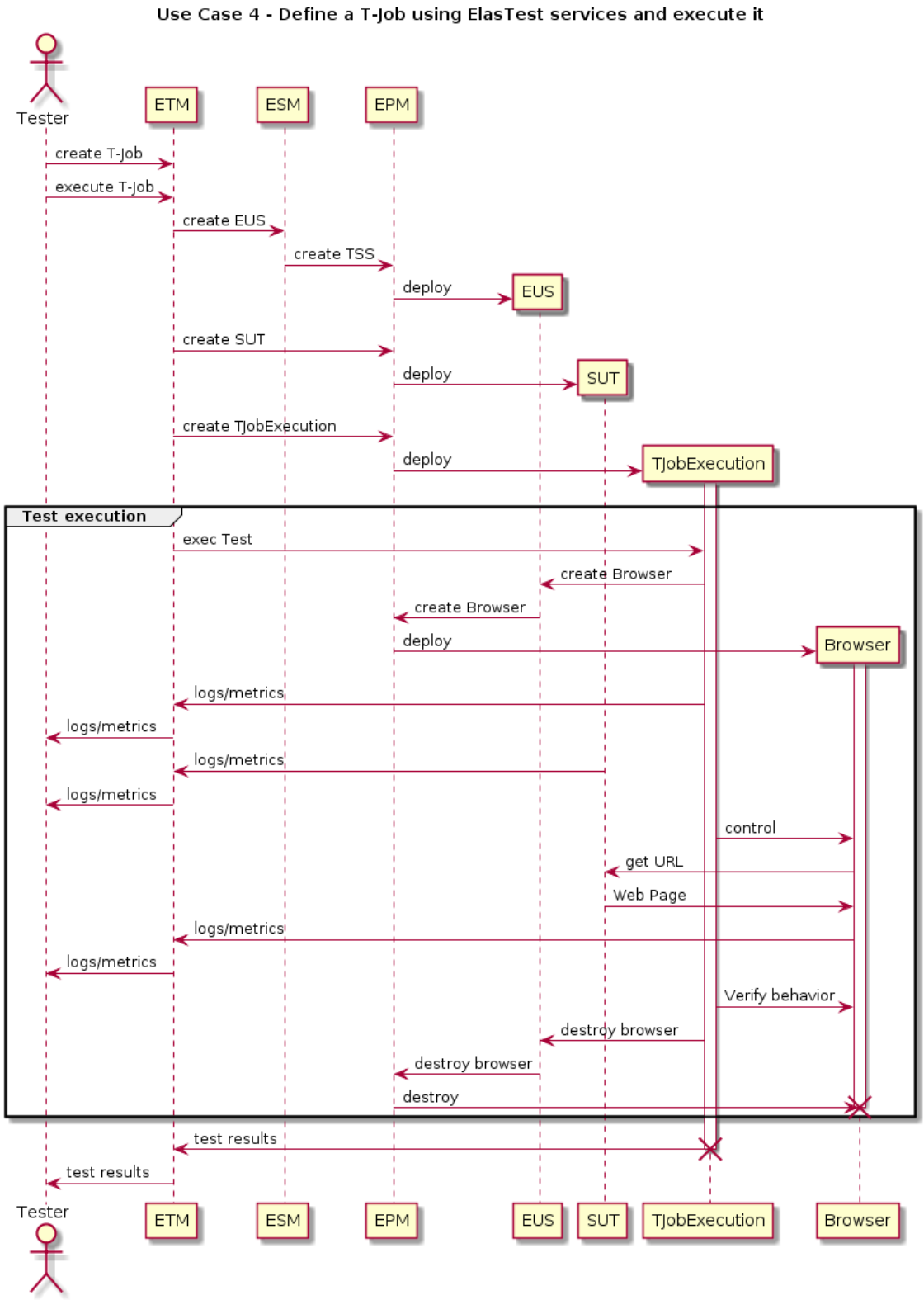

Figure 8. ETM use case - Define a TJob using the ElasTest services and execute it.

5.1.4 Interactions

Table 6. Input to ETM.

Input	Provided by	Remarks
TJob description	Tester	Initialization
SuT description	Tester	Initialization
Execution metrics	Agents, EPM, ESM, Jenkins	During execution
Logs	Agents, EPM, ESM, Jenkins	During execution
Test results	TJobExecution	Finalization
TJobExecution ID	Platform Manager	Deployment

Table 7. Output from ETM.

Output	Provided to
TJob description	Platform Manager
SuT description	Platform Manager
Metrics / Logs	Data Manager, Web Interface
Test Results	Data Manager, Web Interface

5.2 ElasTest Platform Manager (EPM)

The ElasTest Platform Manager is the interface between ElasTest components (e.g. TORM, Test Support Services, etc.) and the cloud infrastructure where ElasTest is deployed. Hence, this Platform Manager must abstract the cloud services so that ElasTest becomes fully agnostic to them and provide this abstraction via Software Development Toolkits (SDK) or REST APIs to the northbound consumers (i.e. the TORM). The ElasTest Platform Manager enabling ElasTest to be deployed and to execute seamlessly in the target cloud infrastructure that the consortium considers as appropriate (e.g. OpenStack, CloudStack, Mantl, AWS, Docker, etc.).

5.2.1 Objectives

The ElasTest Platform Manager shall support the following functionalities:

- Provide the appropriate mechanisms enabling other ElasTest components to deploy and provision the required virtual resources in target cloud environments.
- Abstraction of underlying cloud infrastructure technologies via the Northbound APIs of the EPM to make other ElasTest components to be agnostic to various technologies.

5.2.2 System Prerequisites and Technical Requirements Specification

As the other ElasTest core components, the EPM is delivered as a Docker container. Hence, it is required to have the Docker engine up and running with access to Docker Hub in order to download EPM's Docker image. There are no dependencies to other

ElasTest components besides the EMP and EMS which are required in case monitoring of virtual resources or log forwarding is desired.

5.2.3 Component Design

This section gives a High level description of the ElasTest Platform Manager. The architectural overview (see Figure 9) follows a black box approach but basically the EPM exposes an RESTful API at the northbound interface in order to allow the consumer (e.g. TORM, ESM) to manage virtual resources in a target cloud environment. It allows to allocate, terminate, update virtual resources (e.g. compute, network) and request information of those as well, execute runtime operations, and register and configure new workers. For maintaining state and to allow the user to retrieve state and information of the allocated virtual ressources, the EPM maintains data in a repository. The EPM follows a 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 does not need to take care about the type of the target cloud 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 those information by the EPM component either defined by the Consumer itself or the default configuration.

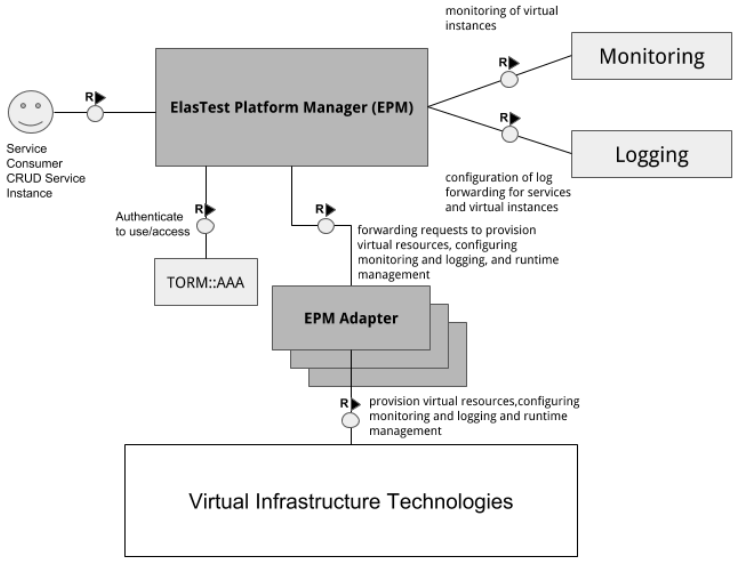

Figure 9. Architectural overview of EPM.

The EPM is intermediate component between other ElasTest components and the underlying virtual infrastructure cloud technologies. The EPM remote API is consumed by other ElasTest components (e.g. ETM, ESM) in order to provision and manage virtual resources in the target cloud infrastructure. Therefore, the consumer can define the virtual resource requirements in two ways by either using the information model exposed by the EPM or technology-specific templates directly (e.g docker-compose or Ansible templates).

The EPM and its remote API are consumed by the following ElasTest components:

- **ETM**: The ETM uses EPM in order to provision virtual resources and manage runtime operations (e.g. TJobs, SuTs, executions).
- **ESM**: The ESM uses EPM in order to provision TSS as virtual resources.
- ...

The EPM itself is not dependent directly on any other ElasTest components but uses the following indirectly:

- **EMS**: As part of the provisioning of virtual resource the EPM configures those to forward logs and to get monitored by the ESM.
- **EMP**: If the EPM is in charge of deploying other ElasTest components or Workers, it configures those to get monitored by the EMP.
- ...

5.2.4 Interactions

The Inbound Interfaces are exposed using a RESTful API and its usage is simplified by java and python clients. These clients provide a programmatic integration with the EPM and are meant to be used by the TORM and the other ElasTest components.

Table 8. Input to EPM.

Input	Provided by
Virtual resource description	ETM, ESM, other ElasTest components
Runtime management operation	ETM, other ElasTest components
Virtual resource lifecycle operations	ETM, other ElasTest components

Table 9. Output from EPM.

Output	Provided to
Details	ETM, ESM, other ElasTest components
Runtime management operation	ETM, other ElasTest components

The Outbound Interfaces are provided by the EPM Adapters discussed above. They connect to the different Cloud Environments and thus providing the options for handling the required resource management functionalities.

5.3 ElasTest Monitoring Platform (EMP)

ElasTest platform monitoring (EMP) is a service that monitors the core modules of ElasTest platform itself. The service is designed to be provisioned on demand, but will be utilized as long lived service.

5.3.1 Objectives

The principal objectives are -

- Ability to track system metrics of ElasTest core modules
- Sensible data visualization allowing comparative analysis amongst components in terms of resource consumption
- Health status panel showing at a glance state of every core component
- Sufficiently rich query interface allowing failure tracing and bottleneck identification amongst ElasTest core components
- Alerting capability allowing proactive notification to control modules when a critical component under observation becomes unhealthy.

5.3.2 System Prerequisites and Technical Requirements Specification

EMP makes use of agent processes and/or properly instrumented code for collection of relevant data/metric streams for visualization and analysis needs. EMP agents and core itself have been containerized, thus requires hosts' capability of executing containers. It is necessary to deploy EMP agents in all physical/virtual nodes where ElasTest core components are to be executed.

Table 10. EMP requirements.

Requirement	Description
EMP-REQ1	EMP framework must be able to allow log ingestion as well as metric stream injection.
EMP-REQ2	Framework must to be able to enforce data retention policy set by admin or users of EMP
EMP-REQ3	Administration interface (RESTful and/or GUI) must exist for users to create / modify monitoring spaces and series in EMP.
EMP-REQ4	EMP must have alerting capability which allows creation of rule based alerts based on health status of critical components under observation.
EMP-REQ5	EMP should at a glance show liveness status of group of

	components that make up TORM.
EMP-REQ6	EMP must allow advance query spanning multiple spaces, and series (belonging to the same user) allowing correlated analysis of data trends, or facilitating bug identification.
EMP-REQ7	API security and proper access control capabilities should exist safeguarding one user's data from another.

5.3.3 Component Design

Figure 10 shows the high level architecture of EMP.

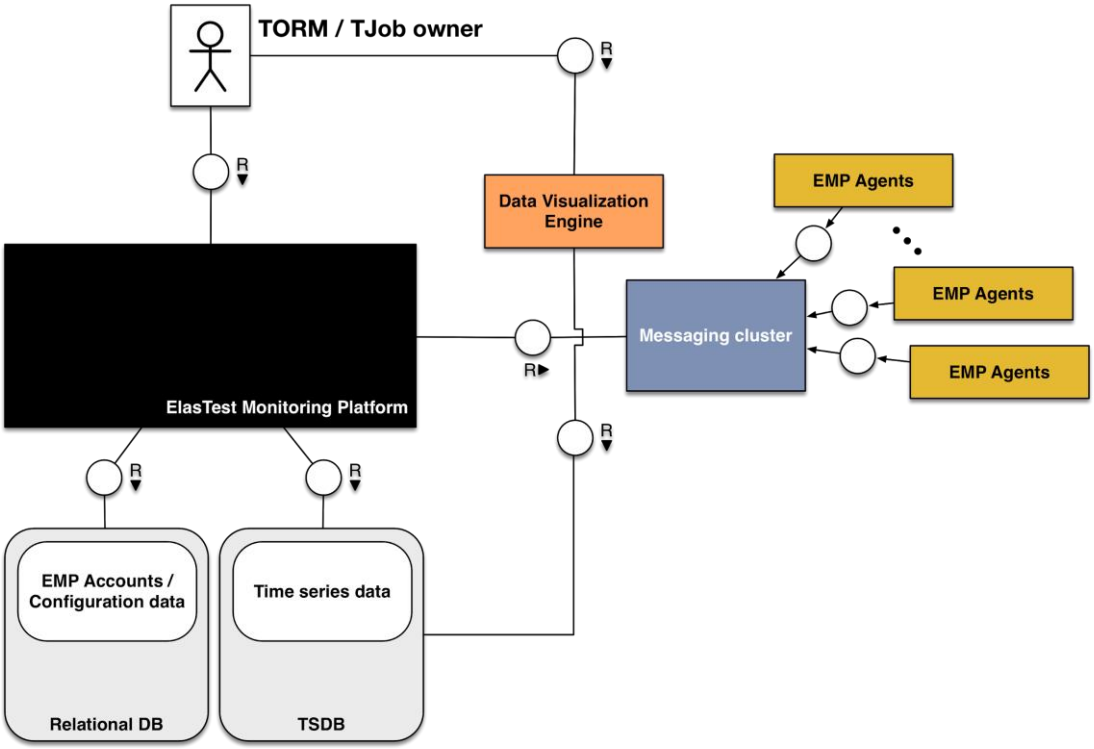

Figure 10. Architecture diagram of EMP.

Key components shown in the Figure 10 above are -

- EMP: provides the RESTful API as well as a graphical web UI for the users of EMP.
- Messaging cluster: messaging framework via which all EMP agents send relevant metric stream to EMP
- EMP agents: client side processes that gather useful data from the target operating environment for sending to the EMP framework
- Relational DB: SQL data store that keeps user account data as well as configured space and series information
- TSDB: time series data store that is used to optimally store the timestamped metric stream sent to EMP from agent processes

Sequence Diagrams showing interactions among various actors are shown next. In order to fully comprehend EMP workflows, it is important to clarify few terminologies.

- EMP space: it is a container where all relevant data streams for a project, application, group of services, etc. are marked as being part of.
- EMP series: each space can contain multiple series, each series stores data streams coming from one instance of EMP agent. Logically, series data belong to a module or component of larger service, application.

To clarify the concept with an example, consider a web service called Thimble_store. In order to service clients of **Thimble_store**, there are multiple microservices behind the scenes, such as *authentication service*, *db-stores*, etc. The provider of Thimble_store can create an EMP space called - *Thimble store Space*. And inside this space, he could create series such as *Thimble store Space-series for-Auth data* and *Thimble store Space-series for-DB access logs*. Let's keep the above anology in mind which reading the sequence diagrams presented below.

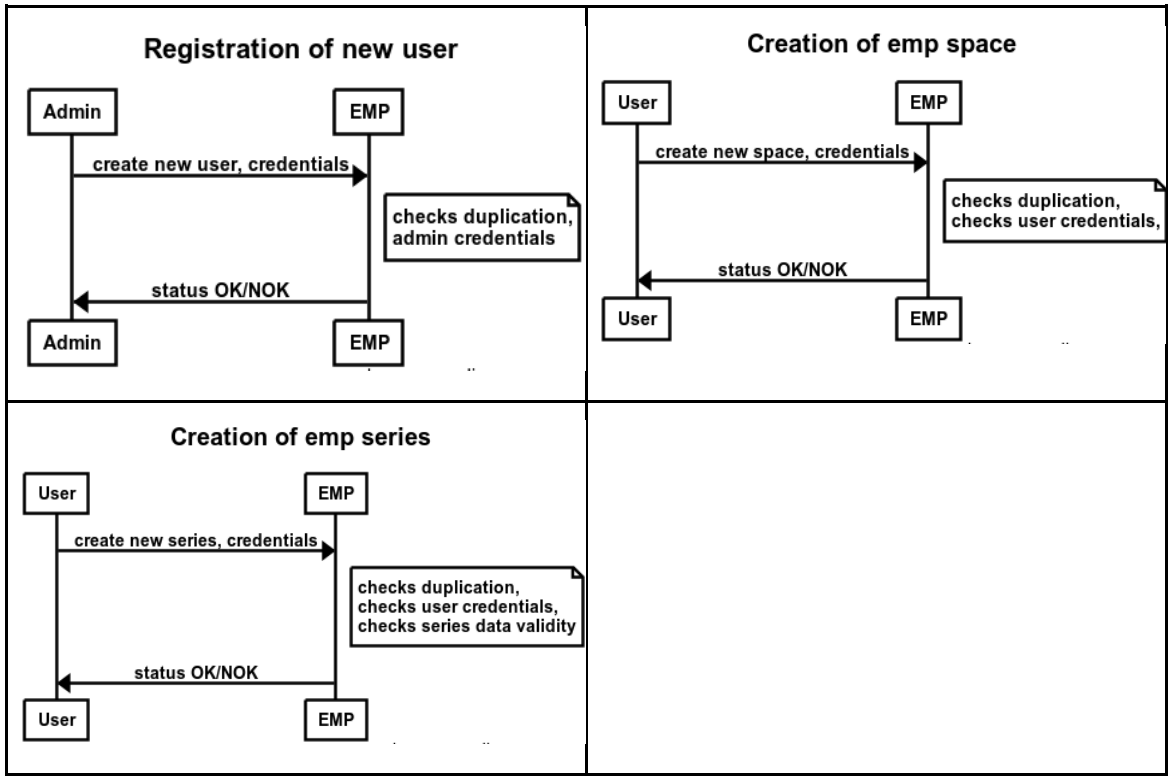

Figure 11. Registration and EMP setup phase interactions.

Once the user accounts have been properly setup, and the users have registered EMP spaces and series within spaces, the framework is ready to accept data streams from various agents via the messaging cluster. Users in ElasTest context are TORM / TJob owners.

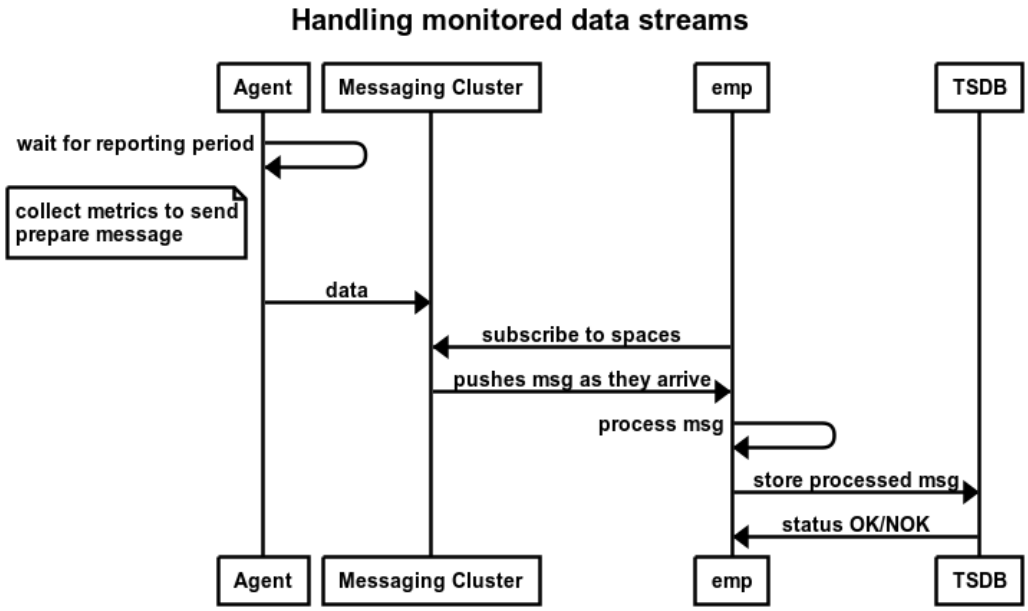

Figure 12. Data processing workflow in EMP

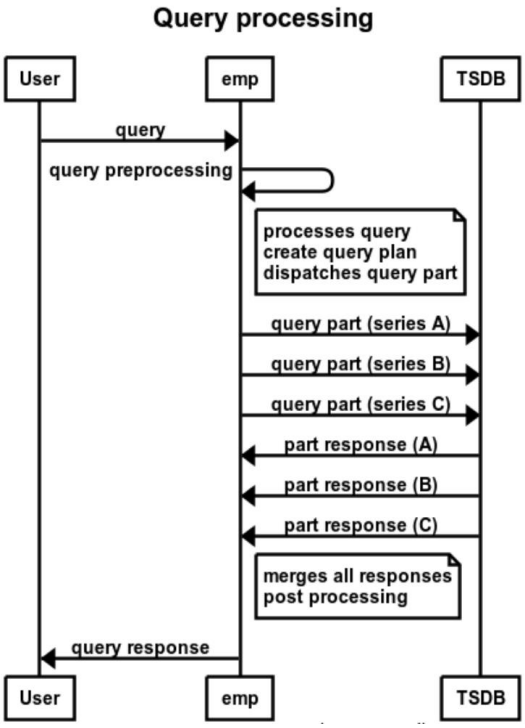

Figure 13. Query processing in EMP.

The query processing workflow described above also facilitates data visualization and feed data stream into the visualization engine. The visualization engine is configured to periodically query EMP and update the charts accordingly.

5.3.4 Interactions

Table 11. Input to EMP.

Input	Provided by
Account data	User or service proxy who wishes to use EMP
Monitored space and series configuration	User or service who is using EMP
Metrics data stream	EMP agents after proper configuration and upon activation
Log stream	EMP agents after proper configuration and activation
Alert definition	User or process using EMP
Query parameters	User or process using EMP

Table 12. Output from EMP.

Output	Provided to
Agent-configuration parameters	End user / process to enable proper configuration of EMP-agents
Query responses	User / process / grafana / web interface
Alert triggers	Registered alert endpoints / callback hook

5.4 ElasTest Service Manager (ESM)

The function and purpose of the ElasTest Service Manager (ESM) can be seen from two main perspectives: the service consumer (the requester of a service instance) and the service provider (the entity that offers their software as a service to service consumers).

From the perspective of the service consumer, the overarching goal of the ESM is to deliver, on request/demand, service instances of particular service types that are required to support the execution of TJobs in the ElasTest platform. Once the service instance is created the ESM has the responsibility to manage the lifecycle of that service instance. This means that testers and/or developers will not need to worry about how to deploy, provision or scale them and the service instances are delivered using the as-a-Service model.

From the service provider perspective, the initial focus of the ESM will be easy on-boarding of software to be offered as a service. It will be important not to lock the service provider into any specific resource management platform (e.g. OpenStack, Docker Swarm, Kubernetes) and as such will take avail of the services of the ElasTest Platform Manager (EPM) as one option to remove this potential lock-in threat.

5.4.1 Objectives

The objectives from a service owner's perspective follows. A service owner has created software that they wish to offer as a service. The service/application is composed of a number of components and optionally other support services. By agreement between all ElasTest partners, and hence by default, the services components are packaged as docker containers. The service/application requires resources to execute, which should be acquired from the EPM. The service owner know approximately how much resources you require and can describe this in a manifest that is specific to the resource provider (the EPM by default).

To provide a means to register your service implementation with the ESM so that end users (incl. the TORM) can request that a service instance can be created on-demand. The service owner will provide/make available the service implementation to the ESM packaged as docker containers. The service owner will also provide resource specifications that are required to run the service implementation. The ESM will take the resource requirements for the service implementation and create these every time a unique new request for a service instance is received. Once the resources are created, the ESM will configure the service instance and ready it for operation. The ESM will then begin to monitor the service instance from the external perspective. Optionally, the ESM could also monitor the software components of the service implementation. Once configured and monitored the ESM will make the instance available to the end-user.

5.4.2 System Prerequisites and Technical Requirement Specification

The ESM's internal technical prerequisites are as follows:

- Docker. As agreed project-wide. This will be used as the base of describing a service, its components and optionally the amount of resources needed per component
- Internal datastore: initially mongodb was selected, however reuse of existing ElasTest platform components motivated the use of MySQL as is present in the ElasTest Data Manager (EDM)

The ElasTest-specific prerequisites are as follows:

- **ElasTest Platform Manager**: used in order to acquire the resources necessary to execute service workloads (the components comprising a service)

The optional but recommended prerequisites for the ESM are as follows

- **ElasTest Monitoring Platform (EMP)**: If no monitoring service is provided then consequently no monitoring of service instances can be carried out.
- **AAA service**. If no AAA service then no access, authorisation or accounting can be carried out. This has the effect of identifying tenants (in a multitenant environment) to their service instances difficult. In order to provide this

functionality, OpenStack Keystone[3] has been proposed internally and accepted.

- **Billing service**. If no billing service is provided then charging for service instances cannot be carried out. This is of lesser priority from a fundamental perspective, however in order to bind service cost models with the TORM (both in terms of static and also dynamic pricing estimation) it needs to be provided. To date Cyclops[4] has been offered as a suitable solution.

In the Table 13 below the detailed requirements that should be satisfied by the ESM are described.

Table 13. ESM requirements.

Requirement	Description
List available service types	Supplies a list of services registered with the ESM that can be instantiated
Create a service instance of a specific service type	Creates the requested service instance. Will install service's software components upon resources provided by the EPM. Service consumers can receive notification of task completion through the "poll service instance status" requirement.
Poll service instance status	Allows a service consumer query the current state of their service instance
Access service instance	Provides service instance access details to the service consumer for example, user-name and password information to access a service's API (e.g. JDBC URL)
Configure service instance	Provides configuration parameters to the service instance to enable completion of service instance creation.
Get service instance details	Gives a description of the service instance.
Get service instance metrics	Supplies summary metrics of the service instance and also optionally provides a URL to the monitoring service where further service instance metrics can be retrieved
Update service instance	This changes the service plan that a service instance is currently operating with.
Remove access to a service instance	Disables access information of a service instance that was previously associated with it
Delete a service instance	Completely deletes (uninstalls) the service instance. This means that all configuration, data of the service instance is removed and then all resources (EPM) that were executing the instance are removed. Service consumers can receive notification of task completion through the "poll service

[3] openstack, https://docs.openstack.org/keystone/latest/

[4] Cyclops, http://icclab.github.io/cyclops/

instance status" requirement.

Other aspects that will be considered in an architectural revision include:

- Service composition, creating pre-req services for the target service. this is the case in EBS and EPM (EBS requires EPM)
- Starting and stopping a service but not destroying it

5.4.3 Component Design

In this section the details of the ESM architecture will be shown.

Context Diagram

Figure 14 the Fundamental Modeling Concept (FMC) diagram of the ElasTest service manager. It shows the key components of the ESM and relations to external ElasTest entities.

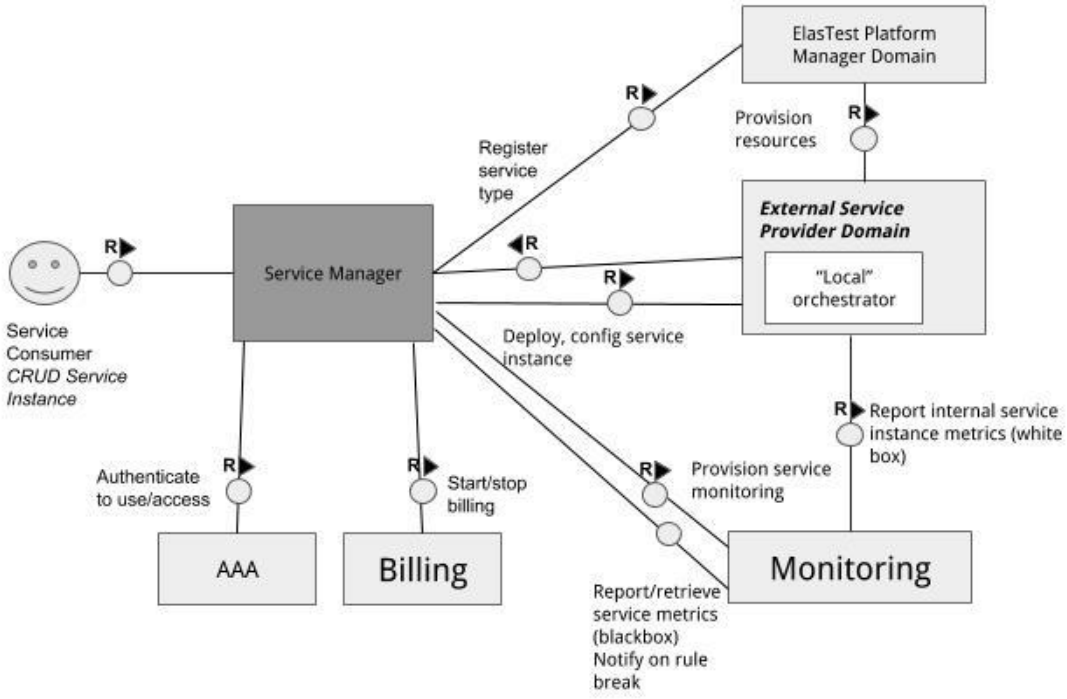

Figure 14. Architecture diagram of ESM

There are a number of components to the ESM and here we provide a brief explanation of each.

- **AAA**: not provided by ElasTest. Currently uses OpenStack Keystone.
- **Billing**: not provided by ElasTest. Currently provided by Cyclops.
- **Monitoring**: Provided by the EMP. monitors service from external perspective e.g. response time, latency, Round Trip Time (RTT) etc.
- **ElasTest Platform Manager (EPM)**: If service provider uses own resources then this is optional. For the case of ElasTest it is mandatory to use the EPM.
- **"Local" Orchestrator** provided by service owner. Orchestration of services and

dependencies by SM. This abstraction allows the service provider either use its own system to provide the service's resources or to use those by the EPM

Further details of the internal components of the ESM can be found in D3.1.1 [3].

Use Case Diagrams

There are two core actors used to model the use cases of the ElasTest service manager. The *ServiceConsumer* has specific *specialised* actors.

- *ServiceConsumer*: this is the entity that requests the service instance and/or uses that instance
  - TORM: The ElasTest Test Orchestration Manager
  - *ServiceProvider*: see below
  - *ServiceManagerUI*: this is the user interface that may be presented to reflect the model of the ESM.
- *ServiceProvider*: this is the entity that provides software on an on-demand basis as requested by a *ServiceConsumer.*

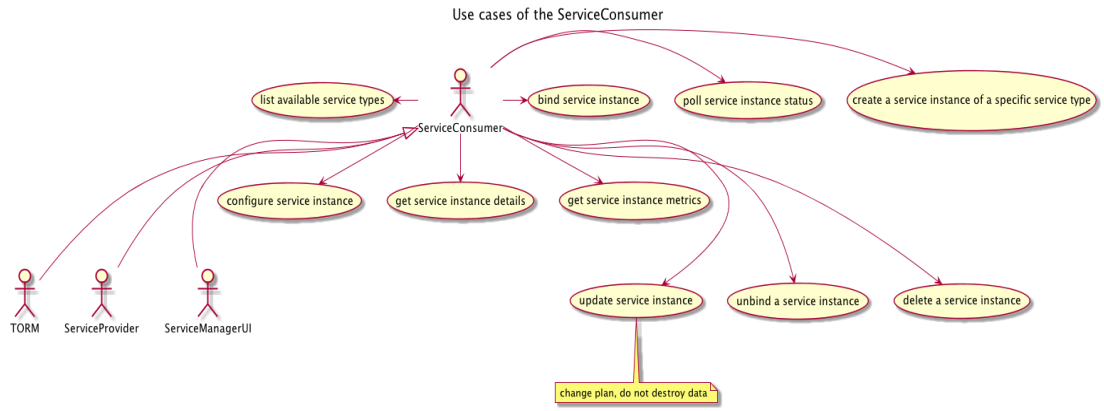

Figure 15. Use cases of ServiceConsumer.

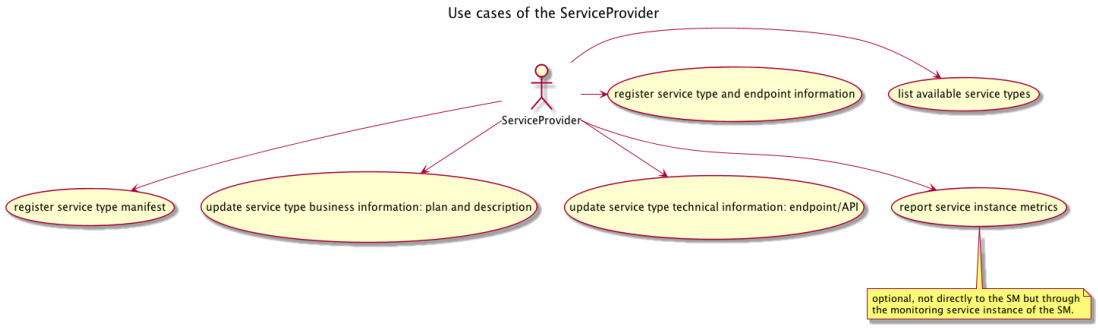

Figure 16. Use cases of ServiceProvider.

UML Sequence Diagrams

Based on the needs of each actor, shown in the use case diagrams, and the FMC architecture, the following UML sequence diagrams (Figure 17 and Figure 18) were created in order to illustrate the interactions between components within ElasTest of the ESM. The internal details are not shown but can be found in D3.1.1 [3].

Service Consumer Interactions

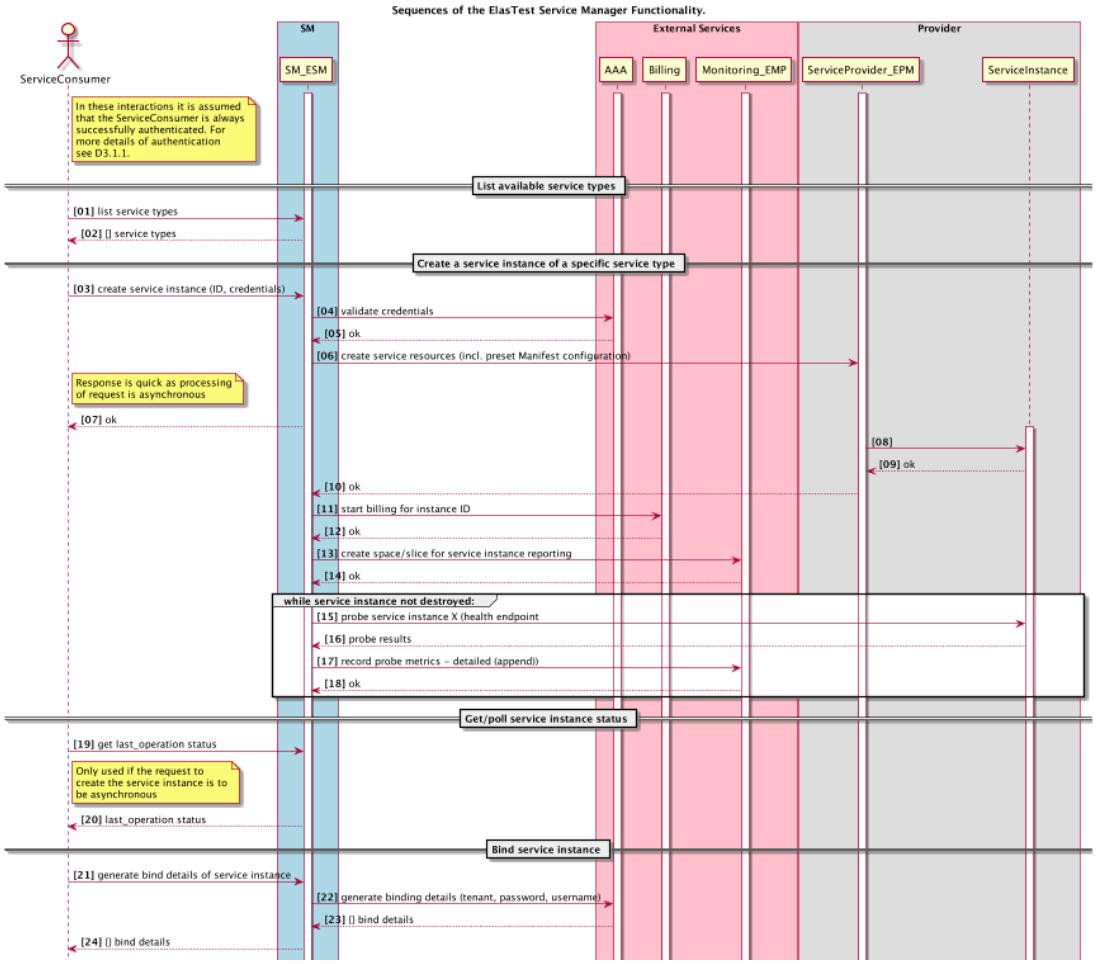

Figure 17. Sequences of ESM functionality - Part 1 of figure.

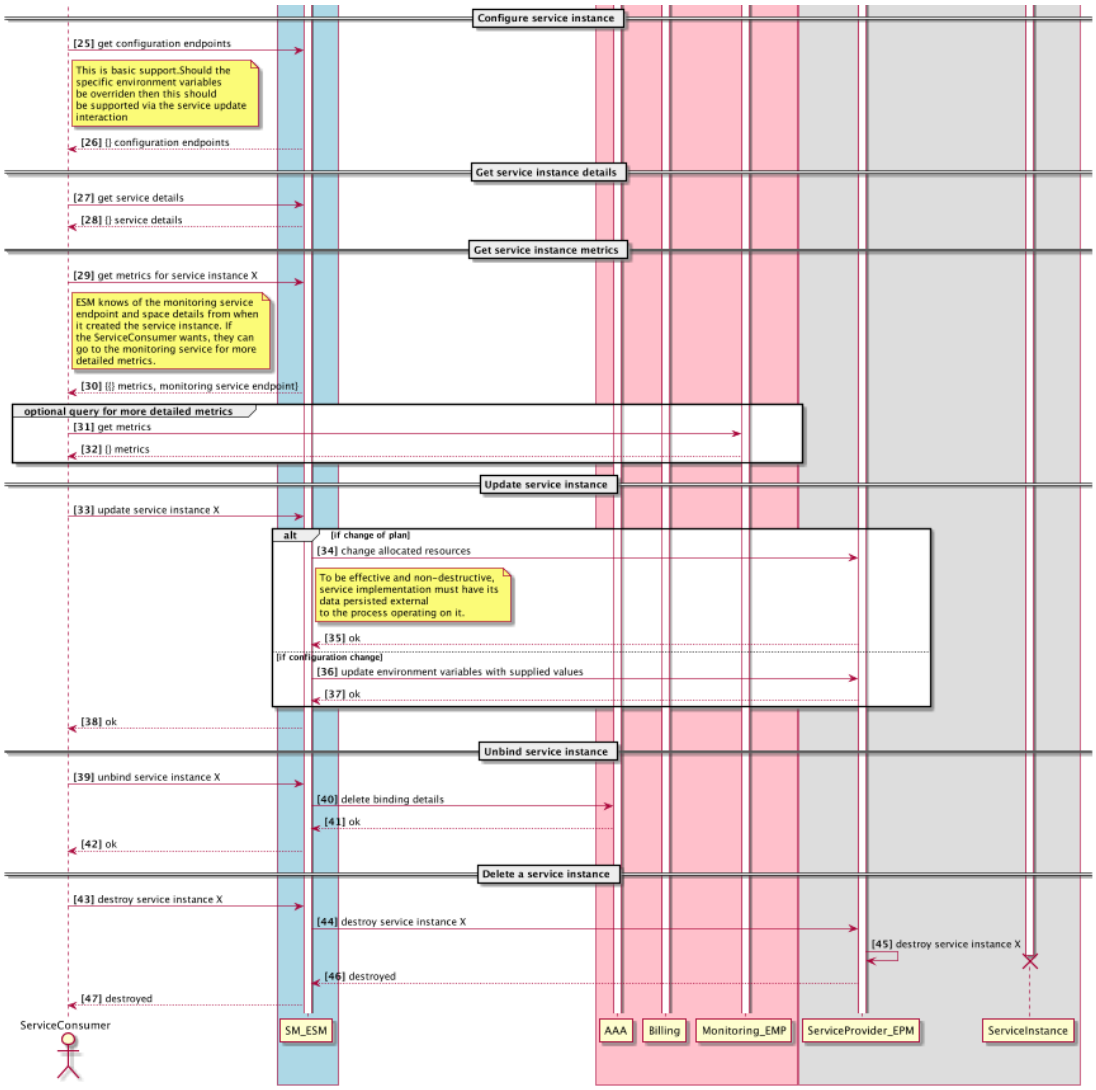

Figure 18. Sequences of ESM functionality - Part 2 of figure.

Service Provider Interactions

The Figure 19 shows the service provider interactions in ESM.

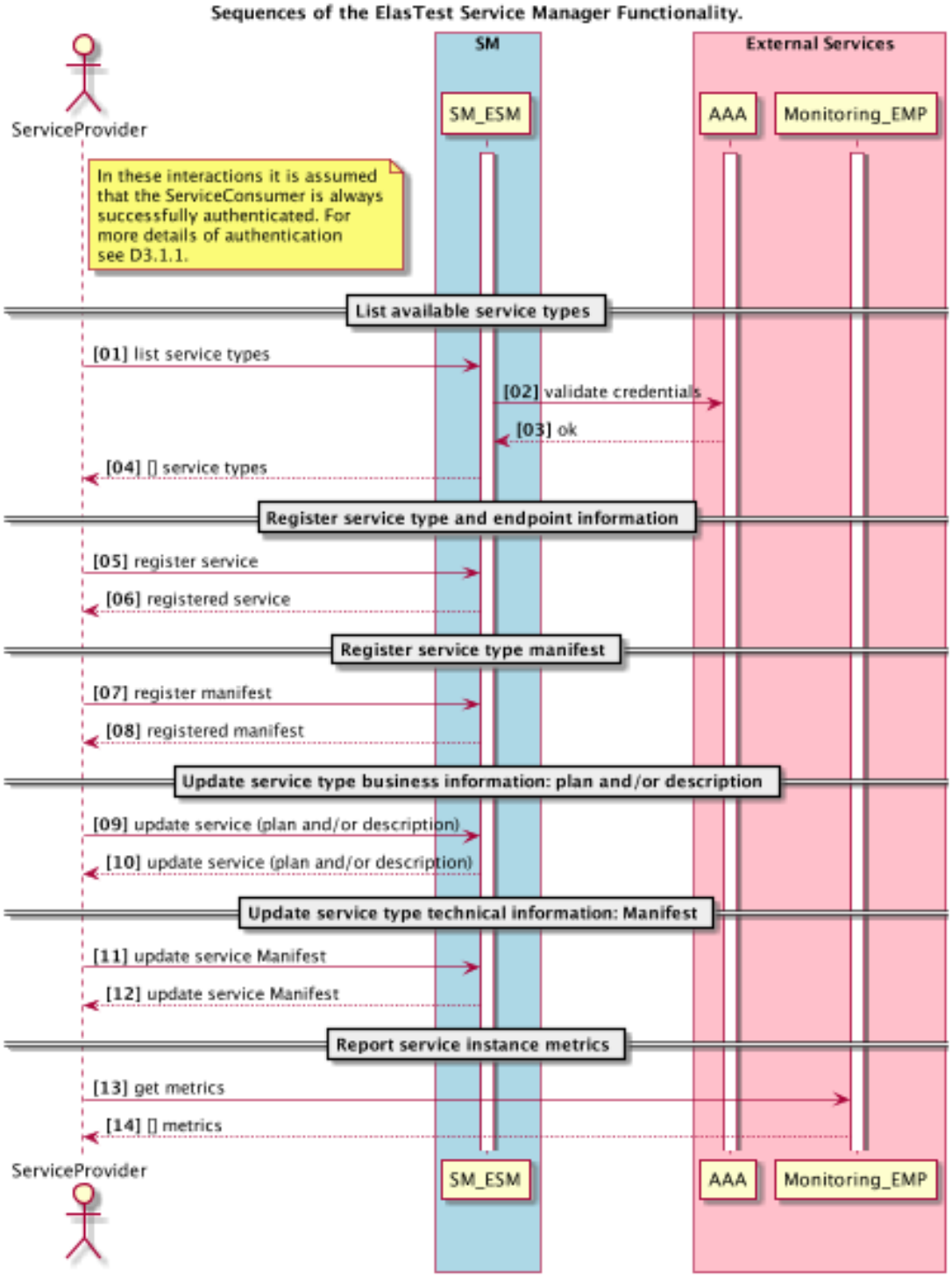

Figure 19. Service provider interactions in ESM.

Data Model

This data model shown in Figure 20 and Figure 21 is the model that is exposed out of the ElasTest Service Manager OpenAPI[5]. The PlanCost model as part of the PlanMetadata is work that is defined in WP4 and is here for completeness. What is not described in this model is the structure of requests that are sent over HTTP. The description of these can be found in the ESM's API OpenAPI definition[3].

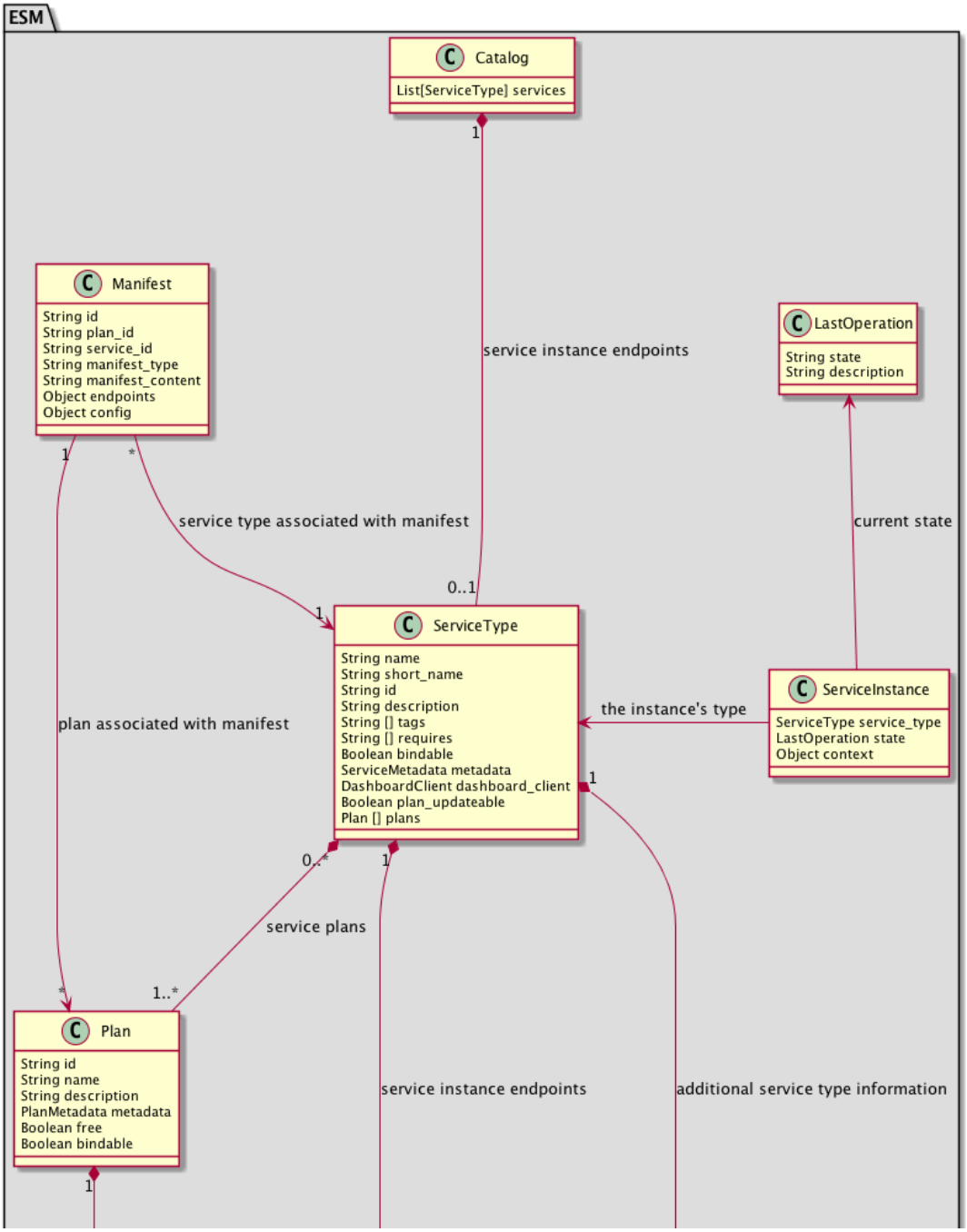

Figure 20. Data model of ESM - Part 1 of figure.

[5] ESM OpenAPI definition, https://github.com/elastest/elastest-service-manager/blob/master/api.yaml

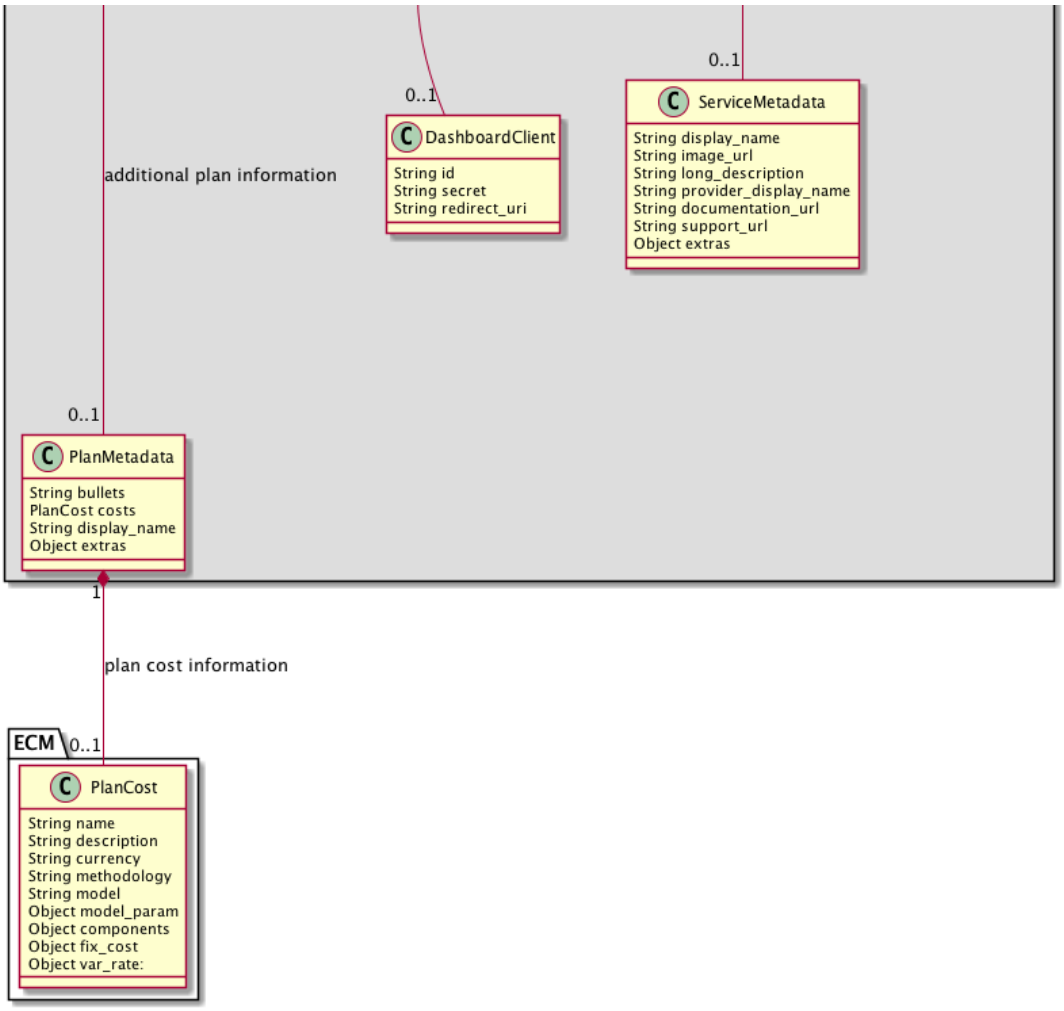

Figure 21. Data model of ESM - Part 2 of figure.

5.4.4 Interactions

Table 14. Input to ESM.

Input	Provided by	Provided to
Service instance request	Service Consumer[6]	ESM
Resource request	ESM	EPM
Service registration info	Service Provider	ESM
Service manifest	Service Provider	ESM
Principal	Service Consumer	AAA via ESM

[6] See Use Case Diagrams

Principal	Service Provider	AAA via ESM

Table 15. Output from ESM.

Output	Provided to	Provided by
Service Metrics / Logs	EMS	ESM
Service creation/deletion	Billing	ESM
Service Instance Info	TORM	ESM
Resource instances info	ESM	EPM

5.5 ElasTest Data Manager (EDM)

The ElasTest Data Manager (EDM) components provides the persistence layer services for all components of ElasTest, along with some extra capabilities on top. It consists of databases (MySQL, ElasticSearch), distributed filesystem (HDFS), Kibana for visual representation of ElasticSearch logs and a web file browser for easy management of HDFS files. Additionally, EDM provides a REST API to manage (e.g. backup/restore) the stored data.

5.5.1 Objectives

The objective of EDM is to provide all the required persistence services to all other ElasTest components, as well as the means to manage these components from a single entry point. Management of the services in question contains the notion of backing up/restoring the data, as well as on-demand scaling and monitoring the services. A list of the requirements is given below:

- To unify all storage requirements of ElasTest under the same umbrella.
- Load data into a log analytics stack for query parsing, search indexing, and trend visualization
- To provide an API for managing all the gathered data, per the given requirements.
- To provide a scalable storage layer for the ElasTest platform.

5.5.2 System Prerequisites and Technical Requirements Specification

EDM is provided as a set of Docker containers. The requirements below describe what is needed to execute EDM standalone, without taking into account any data stored in the system or scaled up services (for processing capacity). The user is strongly encouraged to scale up according to their requirements.

This component is a core part of ElasTest and is required to execute the platform, hence no specific requirements of EDM on other components exist.

A detailed explanation of the requirements satisfied by EDM in Table 16.

Table 16. EDM requirements.

Requirement	Description
Provide an RDBMS	An RDBMS system must be provided for all components that require such functionality.
Provide a flat filesystem for data storage.	Flat file storage must be provided. The capability to perform distributed data processing on top of the data storage layer is required.
Provide a way to integrate with different underlying infrastructures.	All services provided by EDM must operate regardless of the underlying hardware and management technologies.
Provide a log storage and processing engine	Stores SuT logs and processes the contents in the context of test analysis.
Provide a management UI for the various components	Provides a way to scale, backup/restore, and perform administrative tasks to all the components under EDM umbrella.
Provide health status for all services.	The management API performs systematic health checks and provides both an abstract and a detailed status of the component services to the platform.

5.5.3 Component Design

EDM is a bundled package of services, created to support all persistence requirements of the ElasTest platform. The component's architecture is described in this section.

Figure 22 shows the Fundamental Modeling Concept (FMC) diagram of the ElasTest data manager. The key components of EDM as well as the interactions and relations to other ElasTest entities is depicted here.

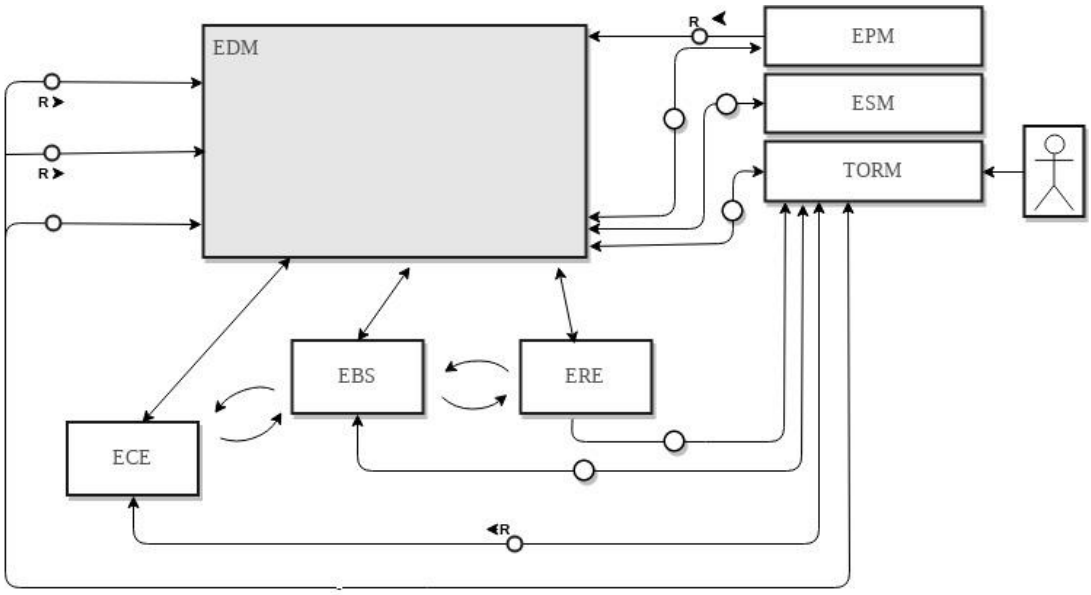

Figure 22. FMC diagram of EDM.

As seen in the diagram above, EDM provides persistence to many different components. In addition to that, EDM provides management and visualization interfaces for some components, as required. As an example, Kibana, Cerebro and CloudCmd are web interfaces provided by EDM forwarded via TORM to the platform user for utilization. These interfaces are described in the “interactions” section below.

5.5.4 Interactions

The entries below describe the various ways EDM interacts with external processes / flows.

Table 17. Input to EDM.

Input	Provided by
Logs	TJobs/SuTs
Relational data	ESM, EPM, TORM
Unstructured data	EBS, ERE
Backup requests	TORM
Restore requests	TORM
Health status	TORM, EPM

Table 18. Output from EDM.

Output	Provided to
Logs	TORM (UI/API)
Logs	UI/API Response
Files	UI (cloudcmd), EBS (Spark API)

Health Status	TORM, EPM
Relational data	ESM, EPM, TORM

5.6 ElasTest Instrumentation Manager (EIM)

The ElasTest Instrumentation Manager (EIM) component controls and orchestrates the Instrumentation Agents that are deployed in ElasTest platform. These agents instrument the operating system kernel of the SuT (Software under Test) host instances. Thanks to it, the agent is capable of exposing two types of capabilities:

1. Controllability, through which the agent can force custom behaviours on the host's network, CPU utilization, memory consumption, process lifecycle management or system shutdown, etc.
2. Observability, through which the Agent collects all information relevant for testing or monitoring purposes (e.g. energy consumption, resources utilization, etc.). The EIM receives through a REST API requests from the SuT Manager within the Test Orchestration and Recommendation Manager (TORM).

5.6.1 Objectives

- To design, specify and implement, at the northbound of the Instrumentation Manager, a set of interfaces suitable for controlling and monitoring ElasTest instrumentation capabilities. This interface shall be consumed by the TORM in order to generate custom operational conditions for the tests.
- To design, specify and implement at the southbound of the Instrumentation Manager, a set of interfaces and APIs suitable for enabling the control and management of the different instrumentation capabilities. This interface must enable the different types of agents to register into the Manager and to be controlled by it. It also must enable the manager to receive, through subscriptions, agents' status information.
- To design and develop the appropriate mechanisms and technologies enabling the Instrumentation Manager to work in a robust and scalable way in all types of cloud infrastructures required by the project.
- To analyze and design the different architectural possibilities for the agent in relation to the testing infrastructure (e.g. one agent for the whole infrastructure, one agent per SuT, etc.) and to generate the appropriate recommendations and technologies enabling its optimal use.
- To provide a toolbox of capabilities enabling the installation, configuration and provisioning of the Manager in all cloud infrastructures of interest for the project.

5.6.2 System Prerequisites and Technical Requirements Specification

Requirements Specification

EIM will be hosted as part of the ElasTest platform, this component is in charge of the lifecycle management of the remote instrumentation agents deployed on top of the

software that we want to evaluate.

Table 19. EIM requirements.

Requirement	Description
Non-intrusive	The agents should be as less intrusive as possible in order to have a low overhead of the instrumentation.
Lightweight	The agents should be lightweight enough as it may need to be deployed within the SuT.
Deploy and configure	The ElasTest Platform should be able to deploy the Instrumentation Agents in the target cloud environments or provide clear guidelines to allow developers to install and configure the agents within the SuT.
Pre-bundle	The SuT may require that the instrumentation agents can be pre-bundled.
Interoperability across OS distributions and version	The agents shall work, at least, on the Linux kernel. The agents should be designed to consume well established operating system interfaces to guarantee interoperability across distributions and versions.

System Prerequisites

- **EIM internal prerequisites:**
  - Docker. As agreed project-side.
  - Web server (tested with jetty and apache tomcat) where the EIM web service will be deployed.
  - The EIM uses a Configuration Management System to install and configure the instrumentation agents on the remote software under test. The Conf. Mgnt. System have been dockerized as part of the EIM.
  - EIM has been tested under Ubuntu 14.04 distribution but other OS systems and versions should be supported as well.

- **ElasTest prerequisites:**
  - TORM (SuT Manager) will invoke the EIM trough a REST API.
  - Monitoring service – EMS. If no monitoring service is provided the monitoring collectors will not be able to propagate monitoring information to other ElasTest modules.

  The EIM need to know the target resource to monitor as well as the end point of the ElasTest Monitoring Service module in order to properly configure the agents.

5.6.3 Component Design

Instrumentation is a collective term used for measuring instruments, which is the activity of obtaining and comparing established standard objects and events (used as

units), the process of measurement gives a number relating the item under study and the referenced unit of measurement.

The Instrumentation Manager will act as a web service that encapsulates the working of the several Instrumentation Agents deployed.

Instrumentation within ElasTest refers to extending the interface exposed by a software system for achieving enhanced controllability (i.e. the ability to modify behaviour and runtime status) and observability (i.e. the ability to infer information about the runtime internal state of the system).

The Instrumentation Agent consists of a software agent that instruments the operating system of the computing nodes (i.e. virtual machines, containers, etc.) where the SuT is deployed. This agent makes it possible to customize all the resources under the control of the node's operating system kernel (e.g. network stack behaviour, CPU utilization, node shutdown, etc.) In addition, the agent collects information on node behaviour including metrics for performance, resource consumption, energy, etc. This is compatible with all types of cloud technologies as it only requires installing and launching the agents on the nodes where the SuT is deployed.

Context Diagram:

In the Figure 23 as appears in the FMC diagram the EIM communicates with the Instrumentation Agents deployed over SuT and with the TORM Manager using a REST API. Fundamental Model Concept (FMC) has been selected for a comprehensive description of the ElasTest software systems. The notation reference is available for download[7].

Figure 23 shows a graphical notation optimized for human comprehension. The same notation has been used to design other ElasTest SW modules; therefore you may find part of the interactions described in the aforementioned figure duplicated across other ElasTest FMC diagrams. A brief description of the components is provided bellow:

- Instrumentation Manager: Control and orchestrates the Instrumentation agents.
- Instrumentation Manager: Control and orchestrates the Instrumentation agents.
- SuT Manager: Invoke the appropriate actions on behalf of the TORM. It uses a REST interface to interact with the EIM web service interface.
- SuT: It refers to the software that we want to evaluate. It could be locally deployed where ElasTest is hosted or may need to be accessed remotely.
- Instrumentation Agent: It is a software agent that instruments the target environment. The agents will be able to perform two different roles over the SuT, observability role allowing the monitoring of the system and controllability role allowing the invocations of actions over the system.
- Monitoring as a Service: This system is in charge of filtering the monitoring data collected by the instrumentation agents as well as distribute this information

[7] http://www.fmc-modeling.org/download/notation_reference/FMC-Notation_Reference.pdf

across the ElasTest modules that wants to receive this information.

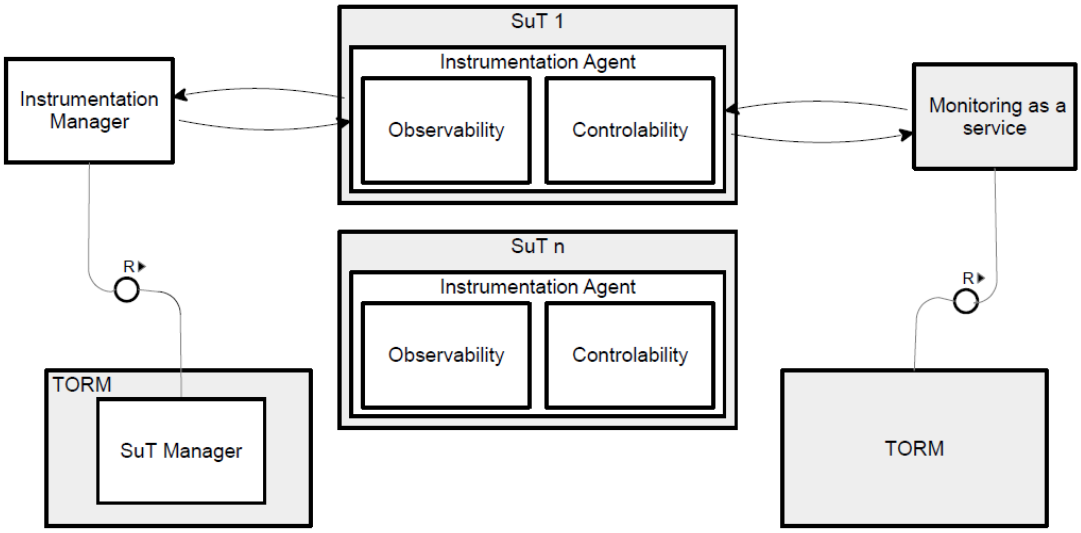

Figure 23. EIM FMC diagram.

Use cases:

The main use cases for the ElasTest Instrumentation Manager (EIM) are described in this section. In addition to the description a sequence diagram divided per use case shows how EIM interacts with other ElasTest components.

Agent deployment: SuT Manager launch the order of deployment of a new agent. This order arrives to EIM and it deploys the new agent over the SuT. This use case is depicted in Figure 24.

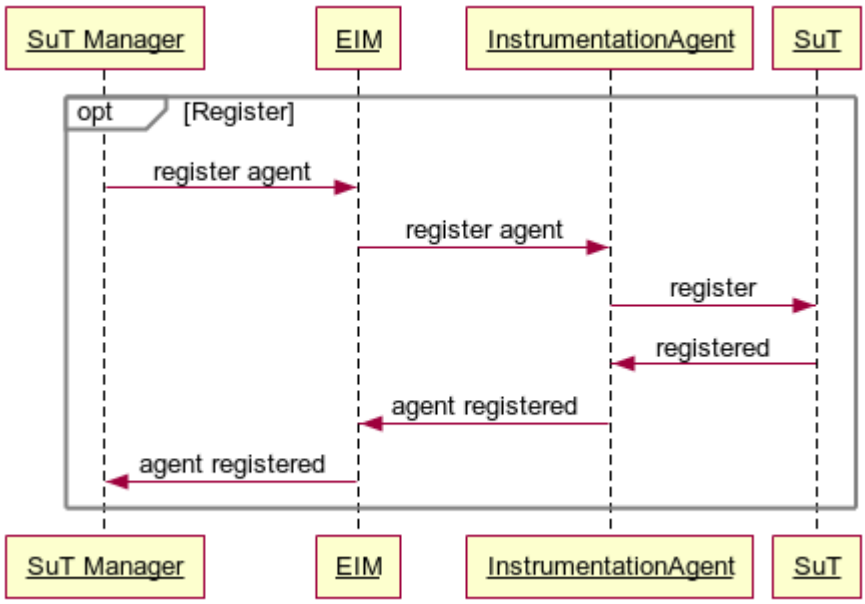

Figure 24. EIM use case - Agent deployment.

Observability operation: SuT Manager launches an observability operation to EIM, the agent collects data from SuT and this data is sent to the Indexer that filter the data and the new filtered data is sent to MaaS (Monitoring as a service) component. This use case is shown in Figure 25.

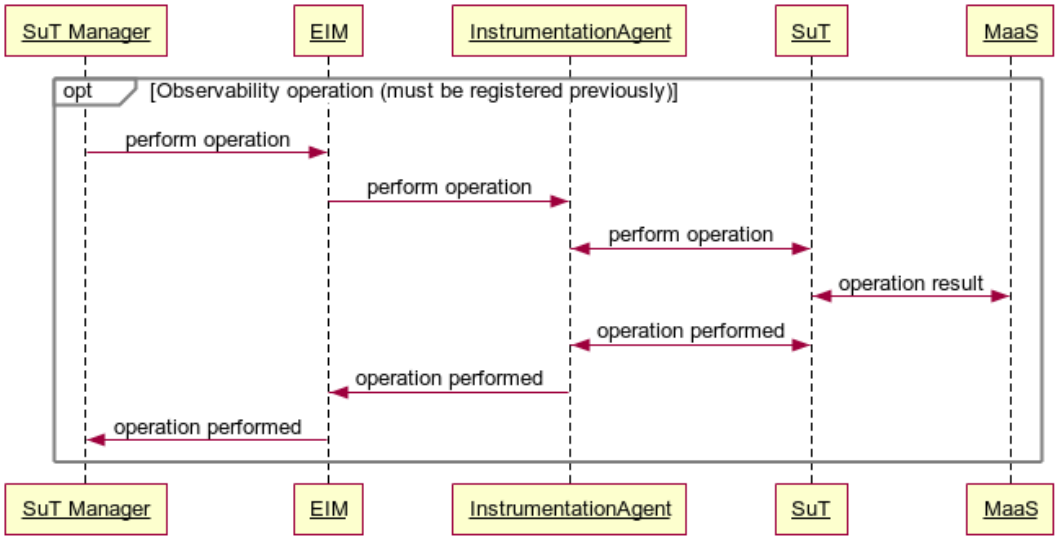

Figure 25. EIM use case - Observability operation.

Controllability operation: SuT Manager launch an controllability operation to EIM, the agent receives the operation and perform it over the SuT. This use case is depicted in Figure 26.

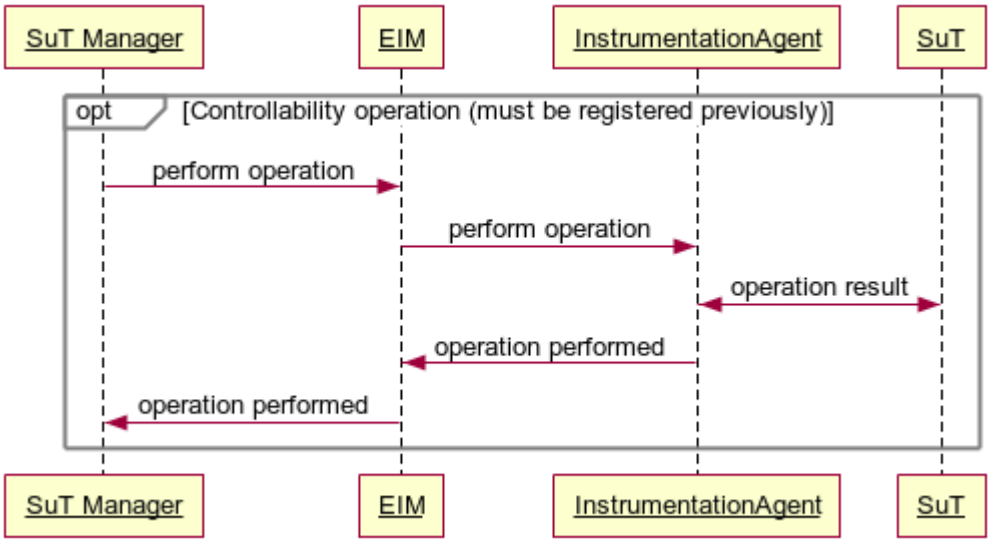

Figure 26. EIM use case - Controllability operation.

Agent undeployment: SuT Manager launch the order of undeployment of an existing agent. This order arrives to EIM and it undeploys the existing agente over the SuT. This use case is shown in Figure 27.

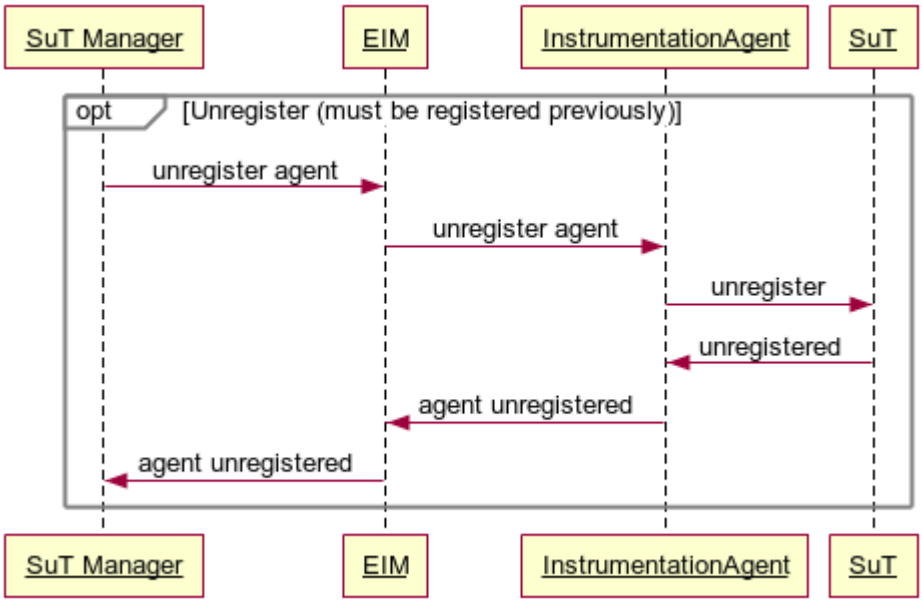

Figure 27. EIM use case - Agent undeployment.

5.6.4 Interactions

Table 20. Input to EIM

Input	Provided by
Agents operations (add, delete, get, update)	EIM
Monitoring and Instrumentation operations	EIM
Operations over agents	EIM

Table 21. Output from EIM

Output	Provided to
Agents operations (add, delete, get, update)	SuT Manager
Monitoring and Instrumentation operations result	SuT Manager, MaaS
Operations over agents	SuT Manager

6 ElasTest Test Support Services

The Test Support Services (TSS) are the services that are offered by ElasTest which can be used to write TJobs and SuTs. These components are ephemeral in nature, such that, the component is alive during the execution of the TJob.

More information can be derived from Table 1 in Section 4. For an in depth understanding of the TSS, the reader is referred to D5.1 [6] of work package 5.

6.1 ElasTest User Impersonation Service (EUS)

The ElasTest User Impersonation Service (EUS) is devoted to provide the appropriate mechanism for emulating final users in end-to-end tests. This shall be achieved by automating GUIs (Graphical User Interfaces) controlled in automated fashion. In addition, EUS enables to measure the end-user's perceived quality so that testing through the validation of the subjective perceived quality becomes possible. For this, the following metrics shall be used:

- Indirect QoE metrics based on QoS metrics. These include traffic metrics such as network latency, network packet loss, network jitter, retransmissions and consumed/estimated bandwidth.
- Direct QoE metrics. We will analyze the multimedia QoE for audio and video using existing algorithms, such as Perceptual Evaluation of Speech Quality (PESQ) for audio or Structural SIMilarity (SSIM) for video to name a few.

6.1.1 Objectives

EUS component is devoted to provide user impersonation for two types of GUIs:

- Web browsers GUIs. For this, EUS provides a Browser as a Service (BaaS) capability suitable for exposing browser GUIs through an API in a universal way. This service have been built on top of the open source web testing framework Selenium.
- Mobile GUIs. For this, EUS provides a Mobile as a service (MaaS). We concentrate on the two main platforms in the market for prototyping an Android device as a Service and an iOS device as a service. This service will be built on top of the popular open source automation for mobile applications Appium.

These services are available for ElasTest users following a SaaS model. Therefore, final users do not need to take into consideration problems related to computing resources scheduling, software provisioning or system scaling since the ElasTest platform is able to elastically provide the required resources to web and mobile automation.

In order to drive browsers and mobile devices in an automated fashion, EUS has been conceived as an extension of the **W3C WebDriver recommendation** [8]. This recommendation was based in the so-called JSON Wire Protocol, first developed by the Selenium team. This protocol defines a REST API instrumented by means of JSON messages over HTTP. Nowadays, this protocol is being standardized in the WebDriver API by W3C.

Therefore, the EUS component provides full compatibility with external browser drivers (e.g. Selenium Grid applications), but enhanced with new capabilities to allow automated for different kind of GUI applications (including browsers and mobiles) while allowing advance quality assessment (including QoE and QoS metrics).

6.1.2 System Prerequisites and Technical Requirements Specification

On the one hand, the EUS's system prerequisites are the following:

- Java. The EUS component has been implemented as a Spring-Boot application, exposing its capabilities by means of a REST API using JSON over HTTP. All in all, this application requires at least a JRE (Java Runtime Environment) installed in the machine hosting ElasTest.
- Docker. As usual, the EUS component is part of at the ElasTest microservices architecture based in Docker. Therefore, Docker engine is required to execute EUS as a Docker container.

On the other hand, the ElasTest-specific prerequisites of EUS are the following:

- ElasTest Service Manager (ESM). According to the overall ElasTest architecture, EUS is one the Test Support Services (TSS). For that reason, EUS instances are started by ESM. To that aim, a Docker Compose file is provided by EUS as part of the description file (elastest-service.json).

[8] W3C Webdriver recommendation, https://www.w3.org/TR/webdriver/

- ElasTest Tests Manager (ETM). The EUS has a GUI devoted to trace browsers executions in ETM. For that reason, in order to use this GUI, ETM should be available.
- ElasTest Platform Manager (EPM): In order to acquire the resources necessary to web browsers and/or mobile devices, EUS will act as client of EPM.

6.1.3 Component Design

In Figure 28, a high-level description of EUS and its relationships with the rest of the ElasTest component is depicted using a FMC diagram. As introduced before, EUS is one TSS, and therefore, is made available in ElasTest by EPM. In the picture we can see a direct relationship with ESM, which is in charge of providing EUS instances within ElasTest. Once EUS is up and running, it is able to create the required browsers for end-to-end web tests. These browsers interact with the System under Test (SuT). Typically, EUS is controlled by TJobs, in which end-to-end tests using Selenium bindings require different types of browsers, created by EUS and controlled by tests. Both TJobs and browsers export log and metrics information, which is gathered in EMS and then it is available in ETM.

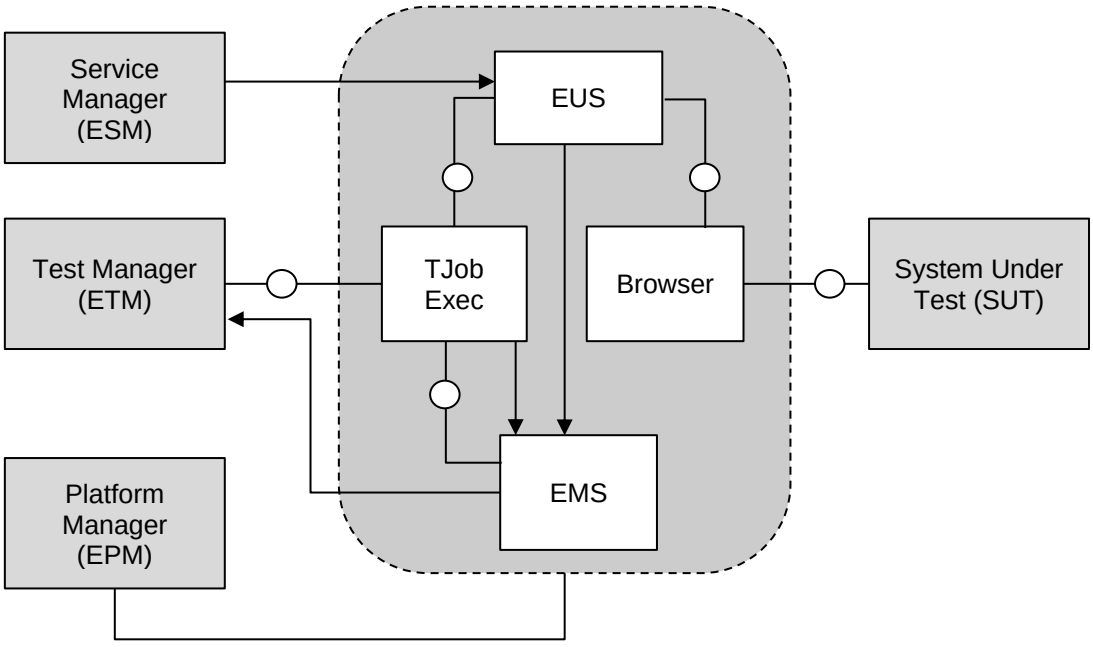

Figure 28. EUS FMC diagram.

6.1.4 Interactions

The following tables summarizes the relationships in terms of input and output from/to (respectively) the different ElasTest components to EUS.

Table 22. Input to EUS.

Input	Provided by	Provided to
WebDriver request messages	TJobExecution	EUS
Health-check requests	ESM	EUS

Table 23. Output from EUS.

Output	Provided to	Provided bt
WebDriver response messages	TJobExecution	EUS
Metrics and logs	EMS	EUS
Video recordings	ETM	EUS
Live sessions	ETM	EUS

6.2 ElasTest Device Emulator Service (EDS)

ElasTest Device Emulator Service (EDS) is a microservice developed in ElasTest as a Test Support Service (TSS), to emulate devices used in Internet of Things (IoT) applications. EDS facilitates in rapid prototyping and testing of IoT applications. The emulated devices include sensors and actuators which form the basis of IoT applications. Furthermore, EDS can be used to build and test interactive IoT applications.

6.2.1 Objectives

The core concept of EDS is centered around providing emulated devices such as sensors and actuators, such that a user can build an IoT application. A typical IoT application, comprises of:

- One or more sensors which provide data to the application
- An IoT application, which consumes the sensor data to apply a logic and come to a decision
- One or more actuators, which can be triggered based on the decision take by the IoT application

For example in a temperature sensing application, we would like to flag an alarm if temperature goes above 50 degree centigrade. For this a temperature sensor is provided which feeds data in periodic intervals say 1 second to the logic. The logic decides if an actuation is needed by checking the temperature provided by sensor is greater than 50 degrees. If greater than 50 degrees an actuating signal is sent to actuator which may be an alarm.

In reality, to build such an IoT application, a prerequisite is the possession of the right sensor and actuator hardware, which are able to communicate via defined interface to a computing machine where the IoT application is hosted.

However, with the facility of emulation, real device hardware is not needed. The software can be built to emulate the behavior of devices. The main objectives of EDS are:

- To provide the feature where user can request and obtain emulated devices from the component.
- Use such obtained devices can be programmatically interconnected to realize meaningful IoT applications.

6.2.2 System Prerequisites and Technical Requirements Specification

The system prerequisites for running EDS are mentioned in the following table:

Table 24. EDS prerequisites.

Pre-requisite	Description
Docker	The technology used in ElasTest to run mircoservice EDS as a docker container
ESM	ESM is a prerequisite to deploy EDS
ETM	ETM is used as an entrypoint, therefore a prerequisite.
OpenMTC	OpenMTC is a reference implementation of oneM2M, M2M communication standard. Interaction with EDS is possible using OpenMTC.

6.2.3 Component Design

Figure 29 shows a simple FMC diagram of EDS. The ETM acts an entrypoint by triggering the execution of a TJob which depends on EDS. EDS is being a TSS, is deployed by ESM. EDS is built using a middleware called OpenMTC[9]. OpenMTC is a reference implementation of oneM2M[10] standard. oneM2m is a standard for Machine to Machine (M2M) communication. The System Under test (SuT), is an IoT application upon which the TJob conducts tests using OpenMTC. The IoT application in the SuT, explicitly requests emulated devices from EDS and connects them in the application.

The core of EDS comprises of the OpenMTC gateway, the EDS Orchestrator and the device emulator. The functions of each component is detailed in the technical deliverable D5.1 [6].

[9]OpenMTC, www.openmtc.org
[10] OneM2M, www.onem2m.org

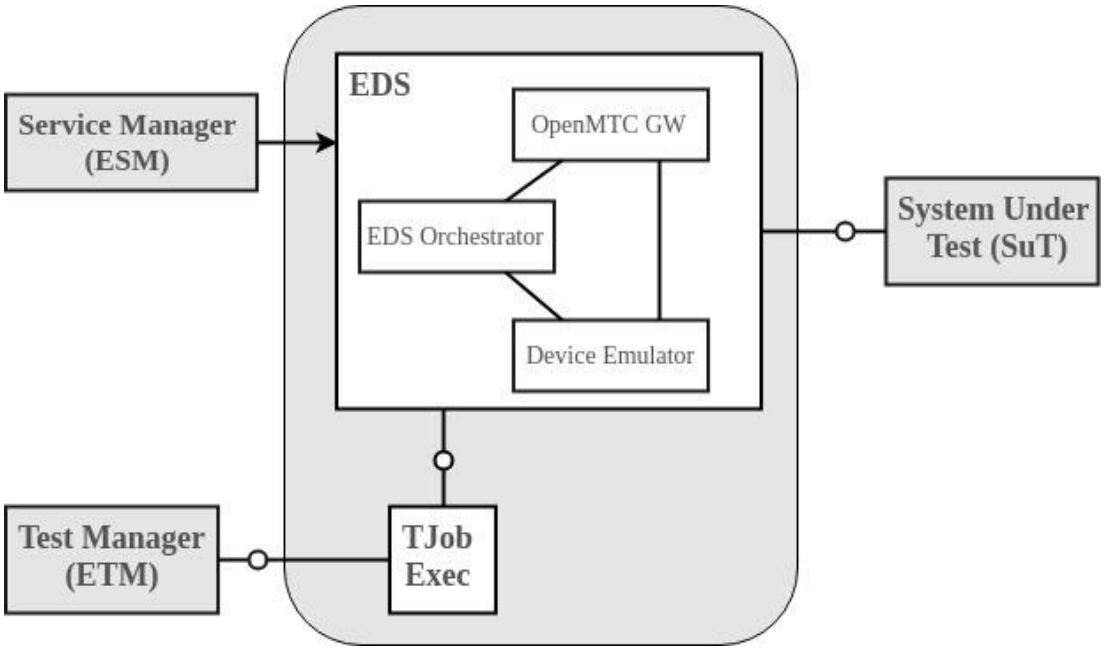

Figure 29. EDS FMC diagram.

Figure 30 shows a sequence diagram, which shows a typical interaction between the user written applications and EDS. Use case, where a user is able to define a SuT application and call upon ETM to execute a TJob on the SuT.

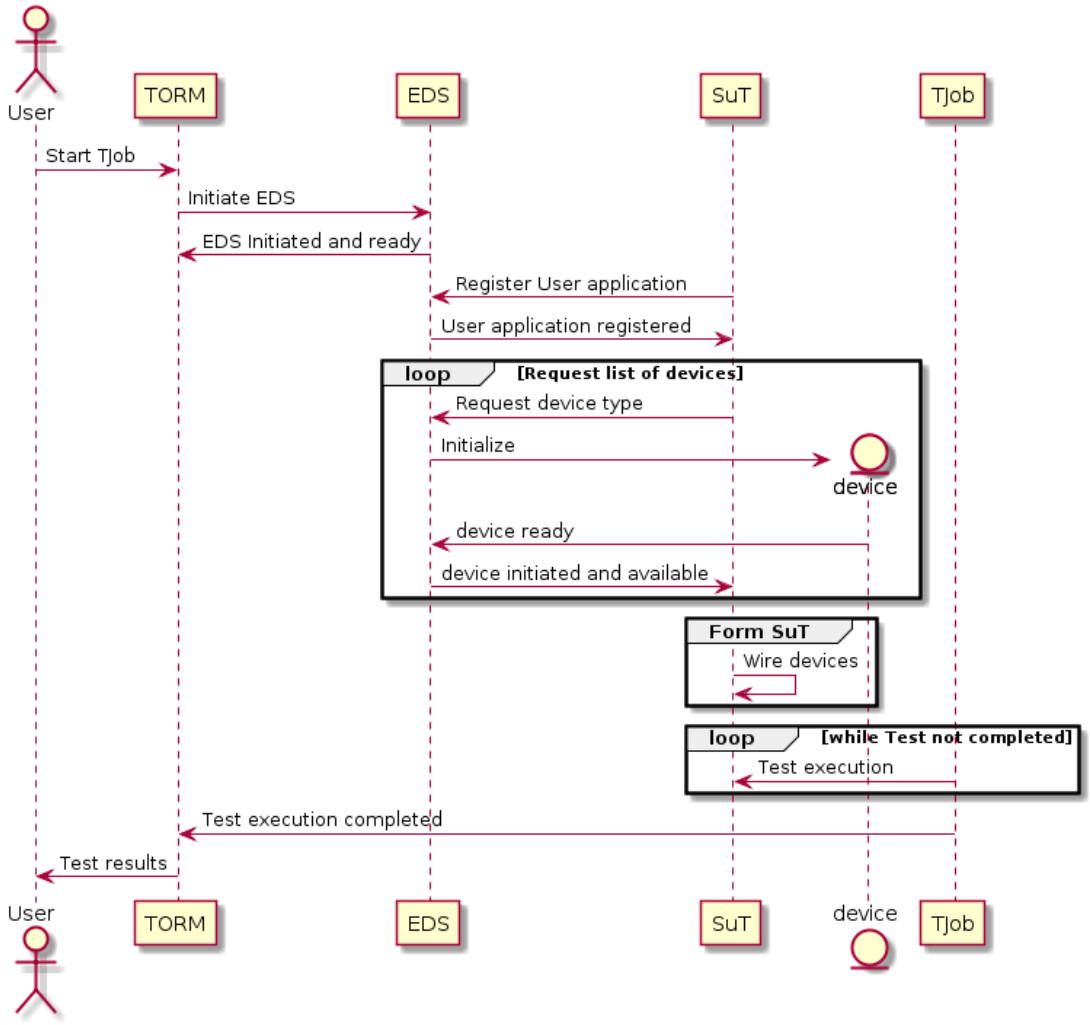

Figure 30. EDS use case sequence diagram.

6.2.4 Interactions

The following tables summarizes the relationships in terms of input and output from/to (respectively) the different ElasTest components to EDS.

Table 25. Input to EDS.

Input	Provided by
Application provision request	SuT/User
Device provision request	SuT/User
Health-check requests	ESM
Device modify request	SuT/User
REST API	SuT/TJob/User

Table 26. Output form EDS.

Output	Provided to
Application provision response	SuT/User
Device provision response	SuT/User
Device modify response	SuT/User
Rest API	SuT/TJob/User

6.3 ElasTest Security Service (ESS)

The ElasTest Security Service (abbreviated as ESS) is a test support service in Elastest that facilitates the security testing of the System under Test (SuT).

6.3.1 Objectives

The main objectives of the ESS are listed below.

- Analyze the HTTP traffic generated by tjob execution and detect security vulnerabilities
- Probe the SuT by sending HTTP requests that mimics attacker behavior and analyze the corresponding HTTP responses to detect security vulnerabilities
- Provide a detailed security test report to the tester

6.3.2 System Prerequisites and Technical Requirements Specification

ESS is a component within the TORM and it is provided as a docker container.

Table 27. ESS requirements.

Requirement	Description
REQ1	Requires **the tester** to write tjobs that makes HTTP connections via the man-in-the-middle proxy of the ESS
REQ2	Requires **the TORM** to provide an environment variable that can be used by the tester to refer to the man-in-the-middle proxy of the ESS
REQ3	Requires **the TORM API** to execute the tjob created by the tester. The execution of the tjob creates HTTP traffic that can be analyzed by the ESS for detecting security vulnerabilities
REQ4	Requires **the TORM web GUI** to display the interface of ESS

6.3.3 Component Design

The FMC Diagram of ESS is shown in Figure 31 below. The tester can interact directly with

1. the ESS (via the ESS API or the Web-GUI),

2. TORM (via the TORM API or the Web-GUI) and
3. the SuT.

ESS needs to interact with the TORM API for executing Tjobs. During the Tjob execution, the HTTP traffic connections are made directly to a man-in-the-middle proxy service running within the ESS container and the proxy intern interacts with the SuT on behalf of the Tjob.

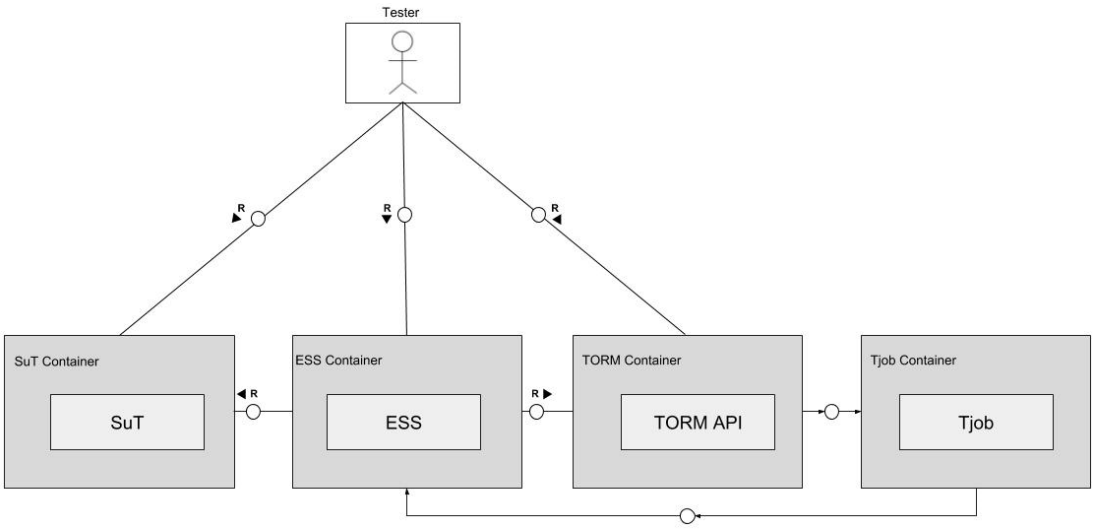

Figure 31. ESS FMC diagram.

The sequence diagram showing the details of the functioning of the ESS is shown in Figure 31. The following are the details:

Steps 1-2: The tester configures a Tjob that makes HTTP connections via the man in the middle proxy of ESS

Steps 3-5: The tester provides the details (mainly the TjobId) of the Tjob that must be analyzed via ESS (Step 3). The ESS uses the TORM API to execute the Tjob (Steps 4 and 5).

Steps 6-9: When the Tjob is executing, the HTTP connections will be made via the ESS's man in the middle proxy.

Steps 10-12: After the Tjob finishes execution, the ESS generates the HTTP requests mimicking attacker behavior (shown a s Security Test's HTTP Requests in Step 10 Figure 2) and the HTTP responses are analyzed to identify vulnerabilities (Steps 11). In the end, the details of all the tests run, including the vulnerabilities found are report to the tester (Step 12).

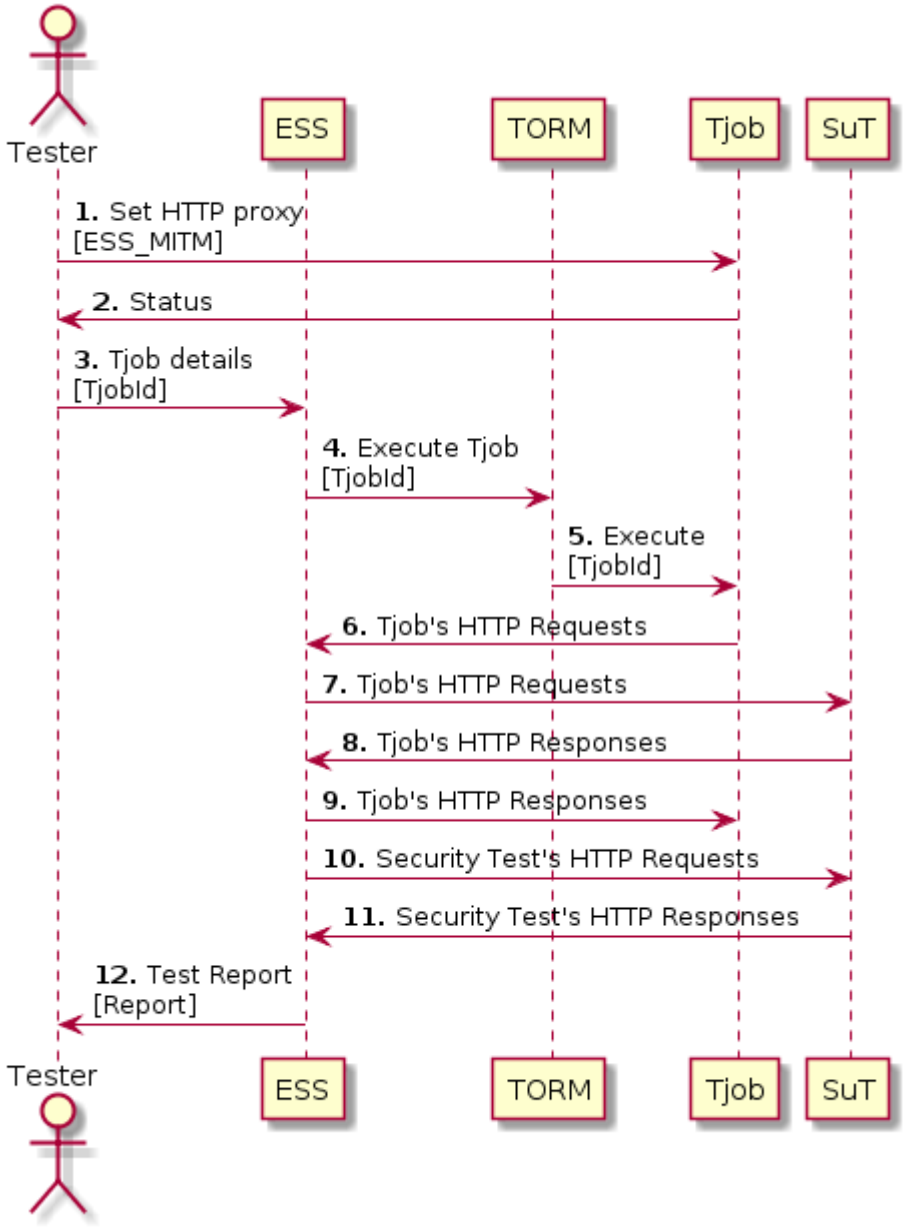

Figure 32. ESS use case sequence diagram.

6.3.4 Interactions

Table 28. Input to ESS

Input	Provided by
HTTP traffic generated during a TJob execution	TORM
Report of the security test in HTML format	TORM
Details of the TJob to be executed	TORM

Table 29. Output from ESS.

Output	Provided to
Report of the security test in HTML format	TORM
Details of the TJob to be executed	TORM

6.3.5 Additional information

It is important to note that while using ESS the tester do not have to write any security tests by himself/herself. All the security tests are automatically generated by ESS.

6.4 ElasTest Big-Data Service (EBS)

The ElasTest Big-Data Service (EBS) is an ElasTest Test Support Service (TSS) that provides a computing engine based on Apache Spark to be utilized by tests inside ElasTest. To achieve that, tests utilizing EBS are built inside a specialized container which is launched as part of the whole Big Data stack (both EBS and EDM parts). For usage and developer documentation, check the component documentation and ElasTest Documentation.

6.4.1 Objectives

- To provide a scalable compute engine for components that should need it.
- To provide a controllable Spark stack to test Big-Data applications.

6.4.2 System Prerequisites and technical Requirements Specification

EBS is provided as a set of Docker containers. The requirements below describe what is needed to execute EBS standalone, in its minimal form (one spark master and one spark worker). In order to scale up, the user is strongly encouraged to dimension accordingly. As EBS does not hold any persistent data, dimensioning is required only on RAM and CPU.

6.4.3 Component Design

EBS is a Spark service as a set of docker containers, designed to be horizontally scalable on demand. In Figure 33, a high level description of EBS and its relationships with other ElasTest components is depicted. As a TSS, EBS lifecycle is managed by EPM, which makes it available to the rest of ElasTest. Also, ERE (and potentially other services as well) is using EBS as a calculation engine, to manage large datasets available in EDM. EBS is capable of utilizing all internal components of EDM as data sources and as data sinks. After calculations are completed, EBS is capable of either returning results to TORM, ERE or any other requesting component, or leaving the results stored in EDM for the other components to access.

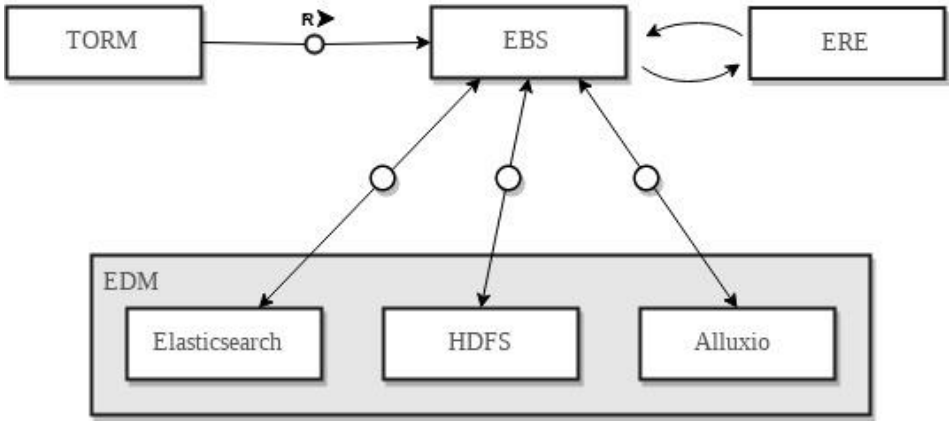

Figure 33. EBS FMC diagram.

Sequence diagrams:

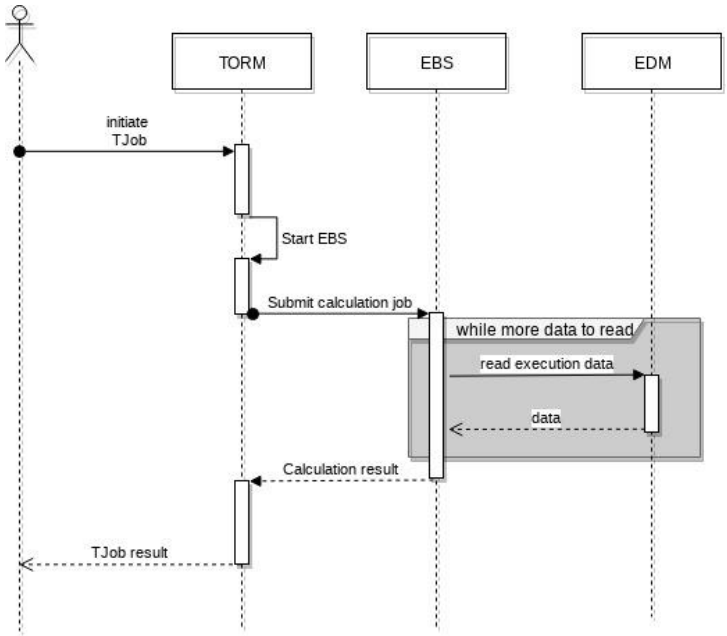

Figure 34. EBS sequence diagram: Use from a TJob.

In this Figure 34, a standard ElasTest workflow is described. The example describes a user-initiated TJob which requires some data processing on SuT produced data (e.g. large logs). The specific TJob initiates an EBS instance, which then retrieves all the required data from EDM components (e.g from Elasticsearch, HDFS, MySQL) and processes the results. The total calculation result is then returned to TORM, where the TJob will use it to generate the actual TJob return status that will be provided to the user.

6.4.4 Interactions

Table 30. Input to EBS.

Input	Provided by
Spark submitted jars	TORM

Output	Provided to
Text data	TORM (UI/API)
Saved files	EDS

6.5 ElasTest Monitoring Service (EMS)

The goal of this component is to provide a monitoring infrastructure suitable for inspecting executions of a SuT and the ElasTest platform itself online.
This service will allow the user and the platform to deploy machines able to process events in real time and generate complex, higher level events from the incoming stream of events. This functionality can help to better understand what's happening in the execution of the test, detect anomalies, correlate issues, and even stress the tests automatically; all of which aims to maximize the chances of uncover bugs and their causes.

6.5.1 Obejctives

- Facilitate the publication of events through an extensible entry point capable of handling a wide range of formats and protocols, thus making it easier to develop agents, and finally collecting as much information as possible.
- Offer a simple yet powerful Domain Specific Language to discover interesting complex events based on the flow of incoming events.
- Handle large amounts of events efficiently in terms of time and memory.
- Provide an extensible subscription endpoint that allows many subscribers to receive events in different formats and protocols.

6.5.2 System Prerequisites and Technical Requirements Specification

1. **TORM/ESM:** EMS requires syntactically valid Monitoring Machines and Announcements to be deployed.
2. **Event publishers:** EMS requires the publishers to properly send well-formed events.
3. **Event subscribers:** EMS requires the subscribers to specify an endpoint and a protocol in which they can properly handle the reception of processed events.

6.5.2.1 System Prerequisites

The EMS will start Logstash instances for incoming and outgoing events, with a few input and output plugins, so all these need to be installed. The Monitoring Machines engine will be developed using the Go programming language, so a supported OS with the necessary libraries is required. Nevertheless, it is planned to package all ElasTest

services into Docker containers so, in that sense, only a Docker engine will be needed to execute it.

Table 31. EMS requirements.

Requirement	Description
REQ1	**From TORM/ESM:** EMS requires syntactically valid Monitoring Machines and Announcements to be deployed.
REQ2	**From Event publishers:** EMS requires the publishers to properly send well-formed events.
REQ3	**From Event subscribers:** EMS requires the subscribers to specify an endpoint and a protocol in which they can properly handle the reception of events.
REQ4	**From platform:** a Docker engine.

6.5.3 Component Design

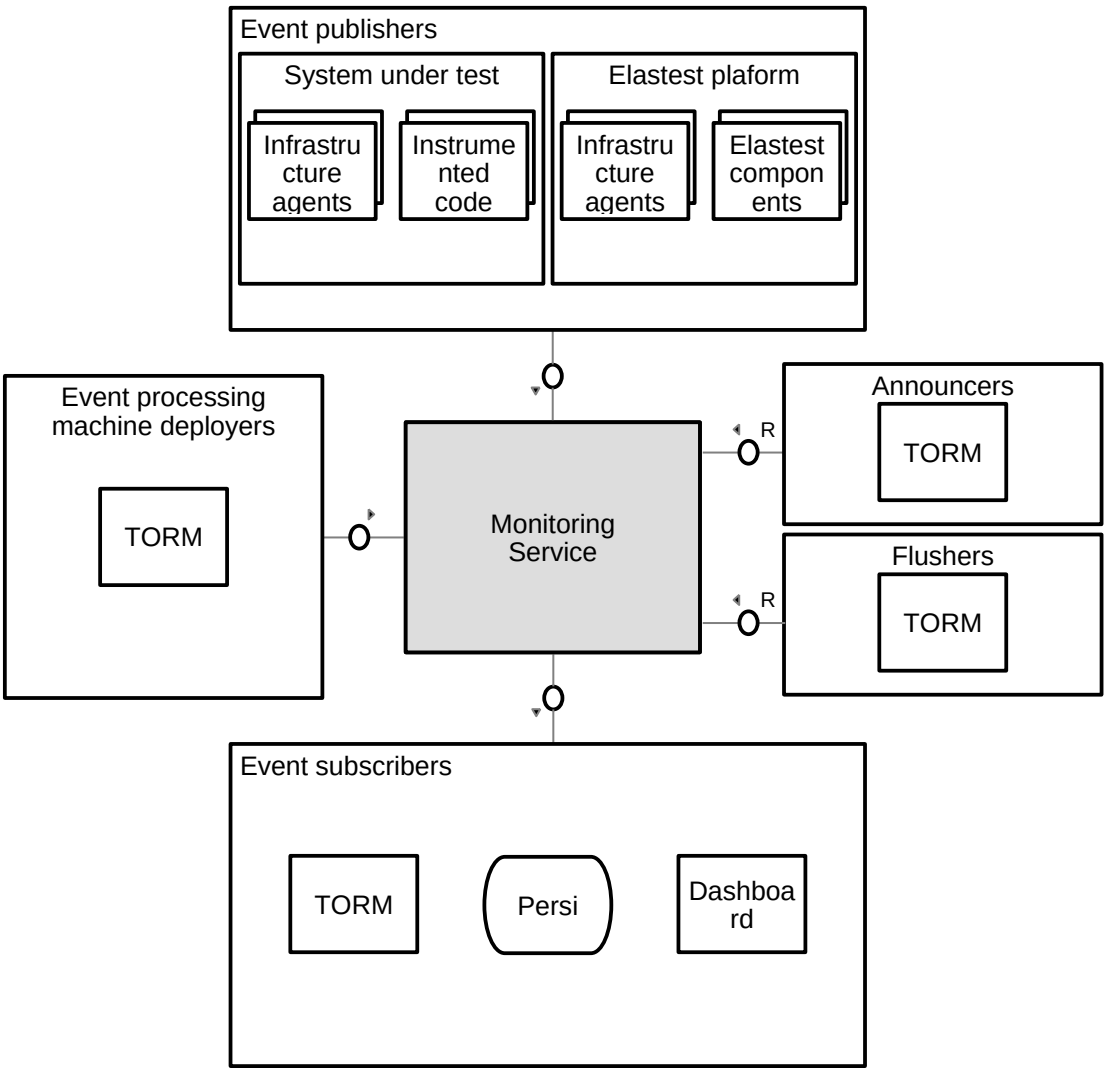

Figure 35. EMS FMC diagram.

The Monitoring Service in the system (external architecture diagram) is presented in Figure 35.

The components present in the previous diagram and their relationship with the EMS are the following:

- Event publishers: external components, such as the instrumented SuT, the infrastructure agents and the ElasTest components in general, including the TORM executing a TJob, who report events to the Monitoring Service.
- Announcers: external components who will provide the rules to infer the channels of incoming events.
- Monitoring Machines deployers: external components who will deploy and undeploy the Monitoring Machines, for example, TORM.
- Event subscribers: external components who are willing to receive events sent over the channels to which they subscribe.
- Flusher: external components who may reset the EMS to its initial state in order to reuse it.

Please note that publishers can write directly to any channel, subscribers can read from any of channel, and the Monitoring Machines can do both (read and write) to any channel, provided that there are no cycles in the induced digraph. The presented topology was chosen for the sake of readability. A channel can be an input and an output channel at the same time, or neither as well.

The system is expected to be spawned and configured by the ESM and the EPM; to get subscription petitions and feed events to many destinations including, but not limited to, the dashboard, the logger, the big data analysis engine and the TORM; and to receive events from many sources including, but not limited to, the agents supervising the infrastructure in which the SuT is deployed, the agents supervising the infrastructure in which ElasTest is deployed, the instrumented code of the SuT, the code of the ElasTest components.

The internal structure of the EMS is the presented in Figure 36:

- Events broker: receives the incoming published events and routes every of them through the corresponding channel.
- Monitoring Machines Manager: is the component in charge of deploying and removing the Monitoring Machines into and from the engine.
- Monitoring Machines engine: holds the list of currently deployed Monitoring Machines, and evaluates each machine for every event received from the broker, sending the resulting events to the event dispatcher.
- Events dispatcher: the component in charge of feeding the output events generated by the engine to the corresponding output channels, sent to the subscribed external components.
- Flush performer: resets the EMS to its initial state upon external request.

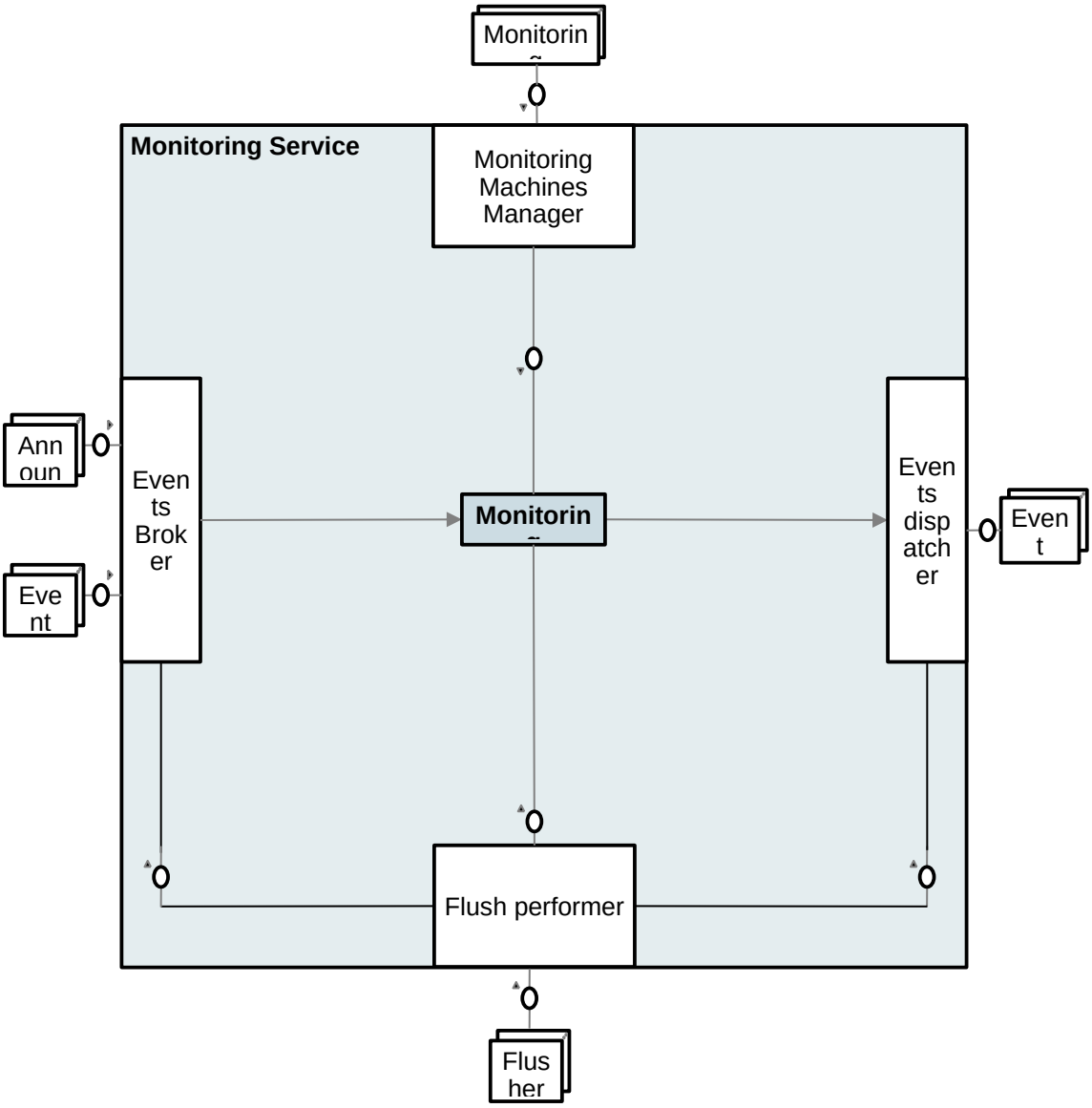

Figure 36. EMS internal components.

Sequence diagrams

Following figures show sequence diagrams depicting the interaction with each component.

1. Management of the Monitoring Machines (formerly, Event Processing Machines, therefore the acronym EPM) shown in Figure 37.
2. Subscription to channels shown in Figure 38.
3. Event publishing shown in Figure 39.

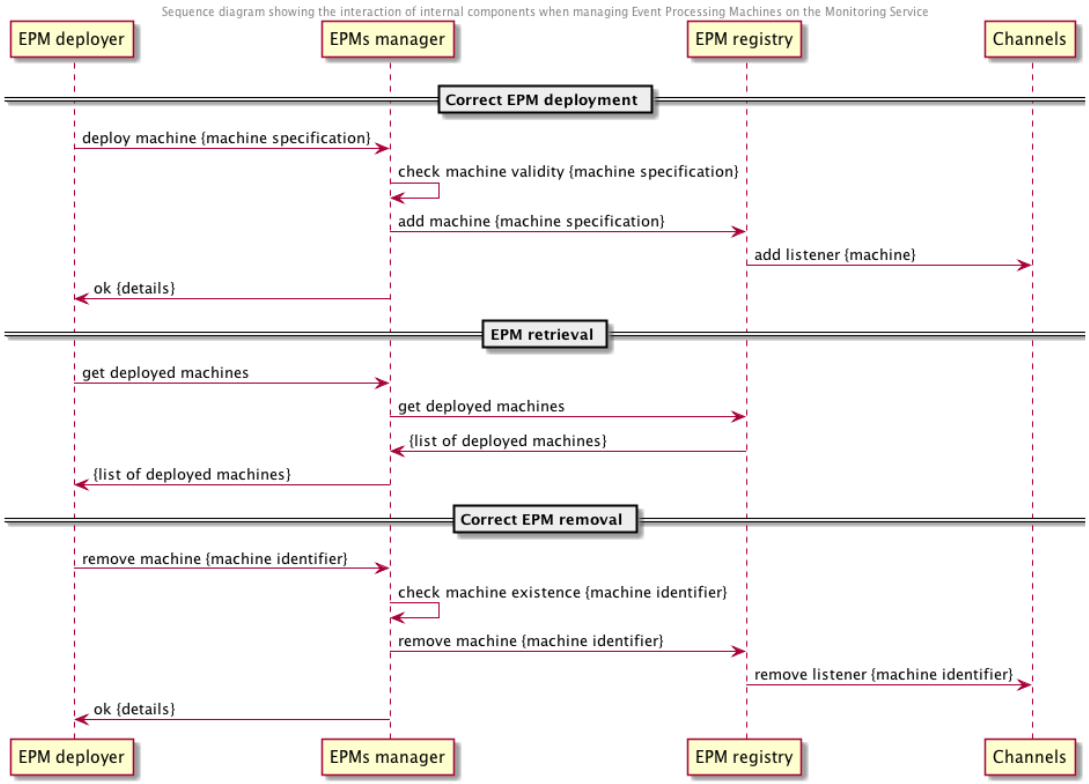

Figure 37. Sequence diagram showing Management of monitoring machines.

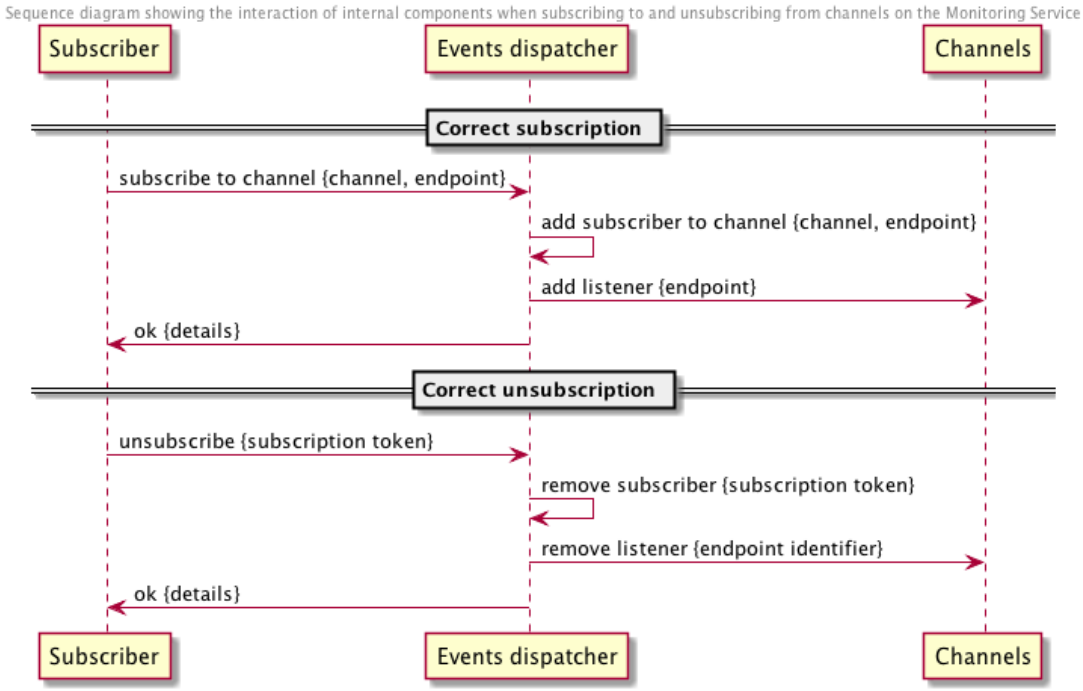

Figure 38. Sequence diagram for subscription of channels.

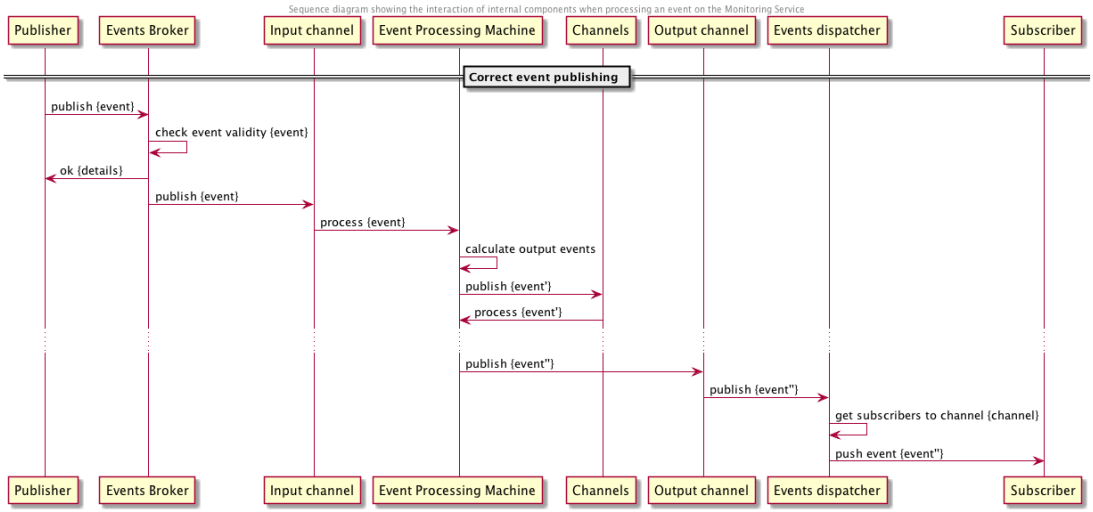

Figure 39. Sequence diagram for event publishing.

Sequence diagram for main use cases:

- ### Executions of a test

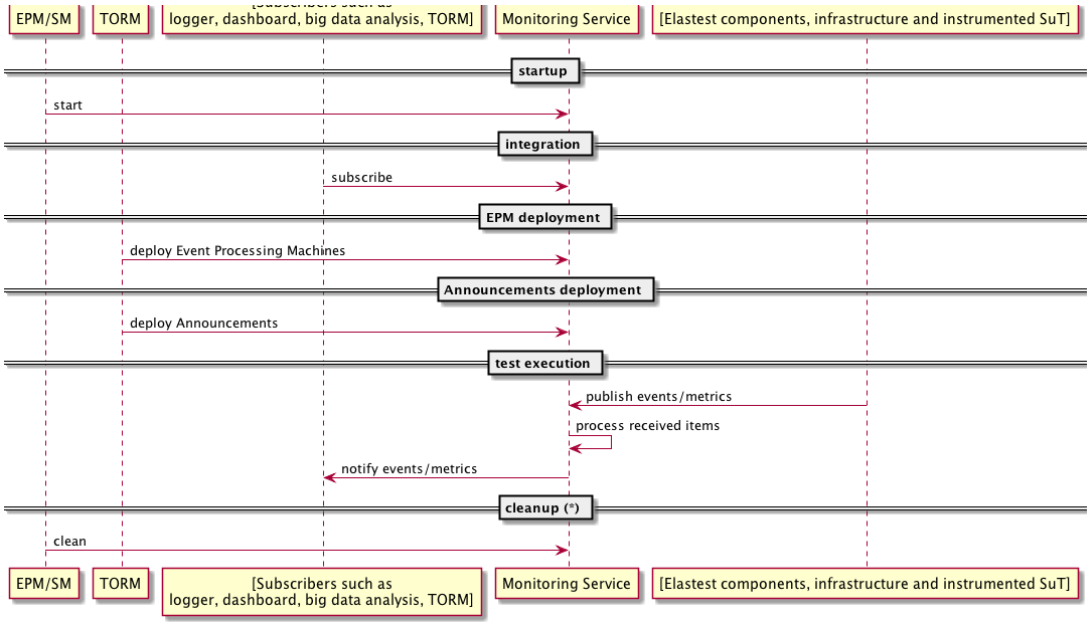

Figure 40. EMS use case sequence diagram - execution of a test.

- ### Debugging of the ElasTest platform

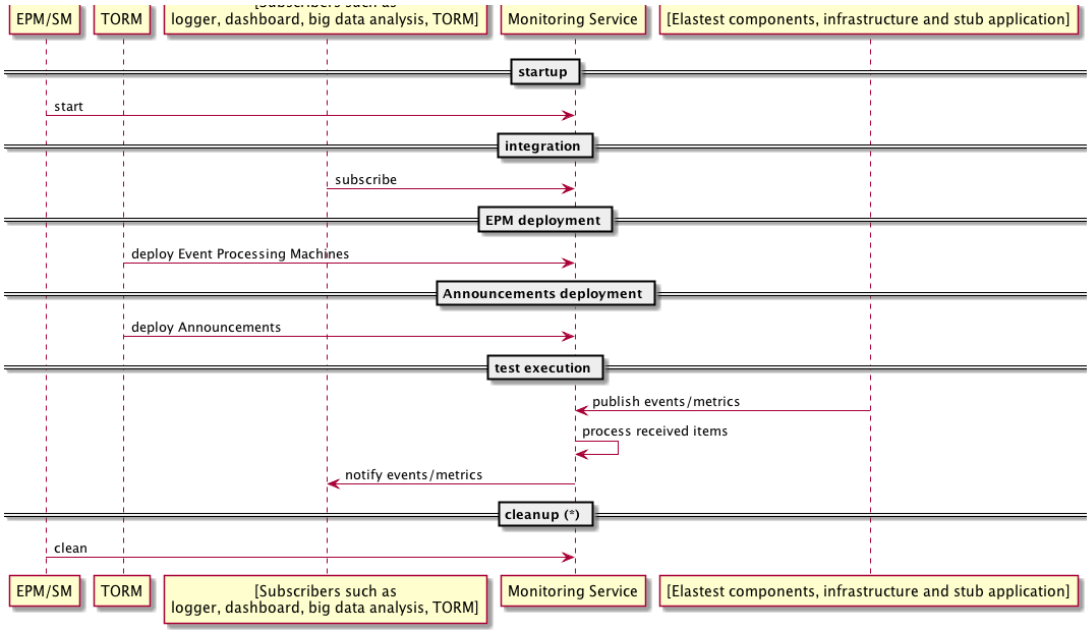

Figure 41. EMS use case sequence diagram - debugging ElasTest platform.

In all these diagrams, the EPM/ESM starts the EMS. Then the subscribers indicate the channels they want to listen to, and where and how they should be sent. After that, TORM deploys the Monitoring Machines along with the Announcements to infer the channel of unstamped events. Later, the tests are executed and publishers start emitting events to the EMS, who in turn processes them using the deployed machines and announcements, and sends the outgoing ones to the subscribers. Finally, the EPM/ESM shall cleanup or flush the EMS in order to reset it to its initial state and reuse it for another test.

As a consequence of the indistinguishability between the SuT and the infrastructure in which it is deployed; and the ElasTest platform and the infrastructure in which it is deployed in terms of the Monitoring Service, it is feasible to debug ElasTest itself using the same tools as in the testing of third parties applications. The main difference lies in the deployed Monitoring Machines, who would focus more on events received from the ElasTest platform, probably ignoring most of those sent by the running “stub application”, which in turn would be specifically designed to stress the ElasTest platform aspects of interest.

Data Model

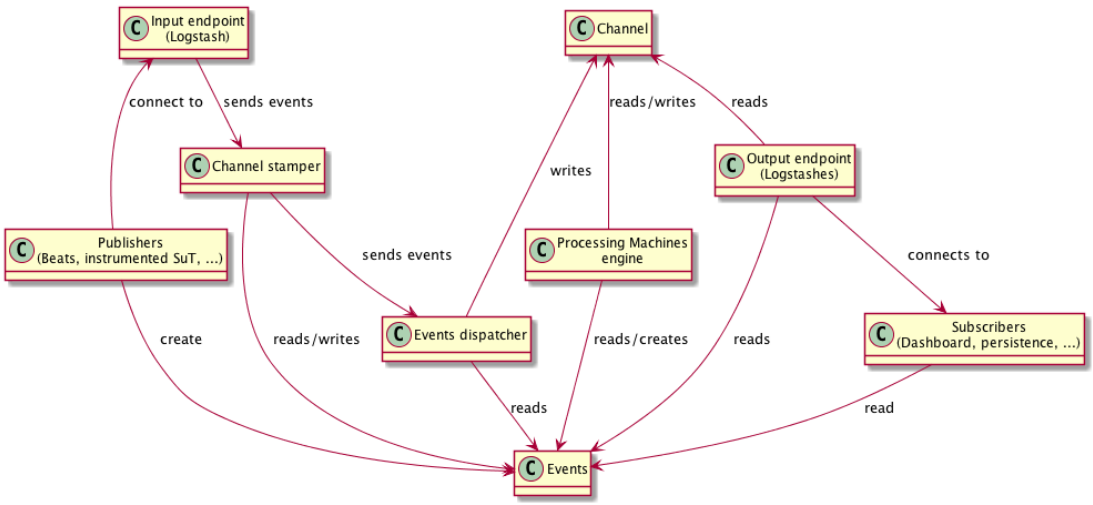

Figure 42. EMS data model

In the data model shown in Figure 42, we can see how the different components interact with each other.

Service Stakeholders:

- EPM: the ElasTest Platform Manager is the interface between the ElasTest testing components (e.g. TORM, Test Support Services, etc.) and the cloud infrastructure where ElasTest is deployed. It will be in charge of deploying the Monitoring Service.
- Dashboard: it is the ElasTest component in charge of visually presenting the data generated by the test to the user through the use of dashboards.
- Logger: This is the service in charge of persisting the data for its further offline analysis and consumption.
- Big Data analysis engine: this component is in charge of providing a distributed system for collecting log data of the ElasTest platform from many sources, aggregating it, and writing it to a Hadoop Distributed File System (HDFS) where it can be analyzed.
- TORM: the service in charge of orchestrating the tests, may tune, pause and stress them in function of the events flowing in the system.
- Infrastructure Agents: these agents will report the status of the infrastructure running code and low level events that take place in them.
- Instrumented code: the SuT will be instrumented in specific places to feed information to the testing environment.
- ElasTest components in general: all the ElasTest components (including the Monitoring Service) may report failures or other information of interest to the Monitoring Service for its processing and analysis.
- ESM: the ElasTest Service Manager will use the Monitoring service to record metrics gathered on service instances under its responsibility. It will retrieve

these metrics either directly or, optionally, provide service consumers (owner of a service instance) the URL to where service instance metrics can be retrieved. It will also use alerting capabilities to inform the ESM of an activated rule.

6.5.4 Interactions

Table 32. Input to EMS.

Input	Provided by
Monitoring Machines	TORM/ESM
Incoming events	Event publishers
Subscribers endpoints	Event subscribers

Table 33. Output from EMS.

Output	Provided to
Monitoring Machines DSL	TORM/ESM
Supported input protocols	Event publishers
Supported output protocols	Event subscribers

7 ElasTest Test Engines

The test engines are the components offered by ElasTest which complement the features offered by the core components. These components can be started by user of ElasTest on demand.

More information can be derived from Table 1 in Section 4. For an in depth understand of the test engines, the reader is referred to D4.1 [4] and D4.2 [5] of work package 4.

7.1 ElasTest Cost Engine (ECE)

ElasTest cost modeling engine task is two-fold, to estimate the cost of running a test on the ElasTest framework using cloud resources, as well as tracking true cost of a test execution run post completion. Cloud resources even when maintained internally in an organization has costs associated - energy, personnel, h/w and s/w costs. One of the goals of ElasTest is to make developers cost aware of running and testing large scale systems over public/private clouds.

7.1.1 Objectives

The main objectives are:

- Development of comprehensive cost model which is extensible, and which can accommodate direct/indirect, own/3rd-party pricing elements

- Cost estimation engine that can predict the cost of executing a test run over a time window based on the cost model specified and resource specification
- Accounting and billing engine which generates actual cost of running a test using actual monitored metrics

7.1.2 System Prerequisites and Technical Requirements Specification

ECE is available as docker image and hence any machine capable of executing docker containers can easily execute ECE. ECE works with a well-defined cost model, and every ElasTest support service must use the proposed model syntax when defining cost as part of their service plan definition. For real cost computation, relevant metrics from the test monitoring framework (EMS) should be accessible by ECE beyond the termination of the test job itself.

Table 34. ECE requirements.

Requirement	Description
ECE-REQ1	The ECE module should be able to get list of all TJobs, from ETM and TORM
ECE-REQ2	The ECE module must be able to interact with ESM to fetch the service plan definitions for support services
ECE-REQ3	ECE must predict the cost of execution of a test based on cost model parameters and execution parameters
ECE-REQ4	ECE must show the true cost of test execution based on collected execution metrics by TJob monitoring system
ECE-REQ5	Given a TJob ID, ECE should be able to locate and then query monitoring API endpoint for execution specific metrics such as start-time, end-time, resource consumption data either aggregated or time stamped list of sample points.

7.1.3 Component Design

The Figure 43 provides a High level description of the ECE.

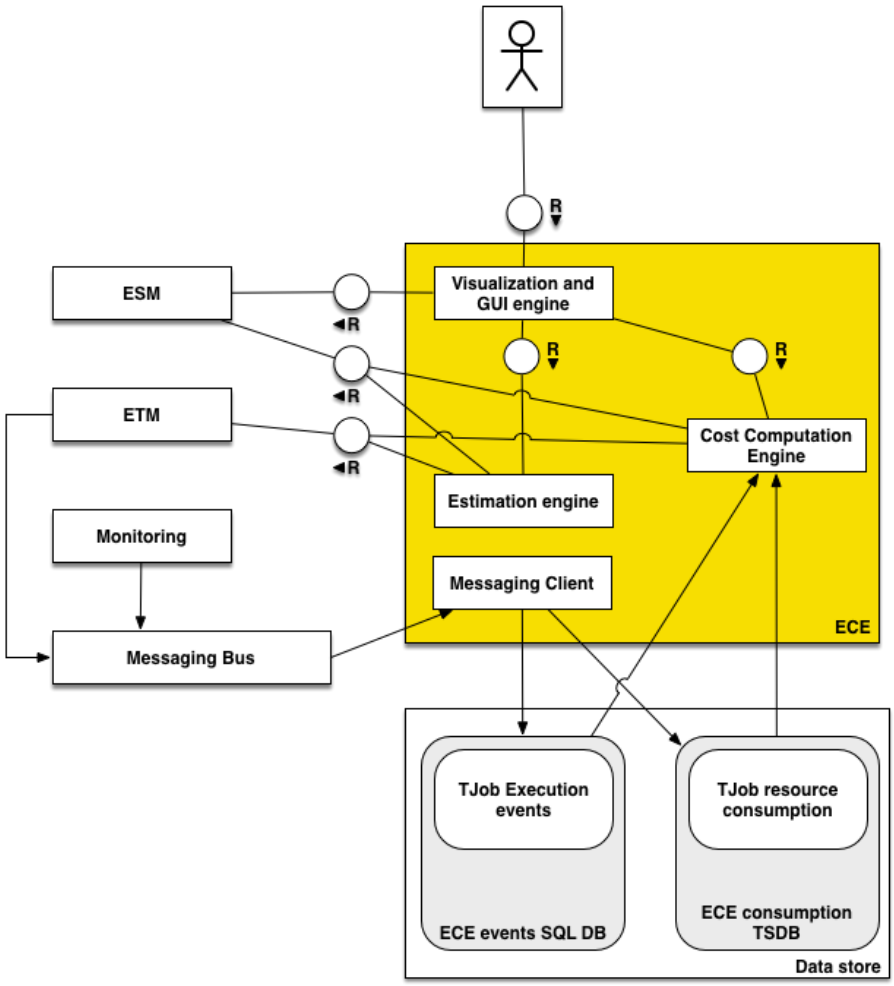

Figure 43. FMC diagram of ECE.

Main components shown in the diagram above are:

- Visualization and GUI engine: this component allows user interaction with the engine, it fetches the list of registered TJobs with ElasTest TORM and allows users to initiate estimate or calculation of actual cost analysis for the selected TJob
- Estimation engine: this module computes the estimated cost for running a TJob together with requested support services using the cost model defined by various services.
- Cost computation Engine: this module gets all execution run list of a particular TJob and using actual execution parameters, resource consumption metrics observed during execution, and defined cost models computes the true cost of running the test.
- Messaging Client: execution events (start/stop) and monitored metrics are sent to messaging bus, this module fetches the messages off the queues and persists them to relational DB or time-series data store based on the nature of the data.

The main interaction between involved actors and ECE is shown next via few sequence diagrams.

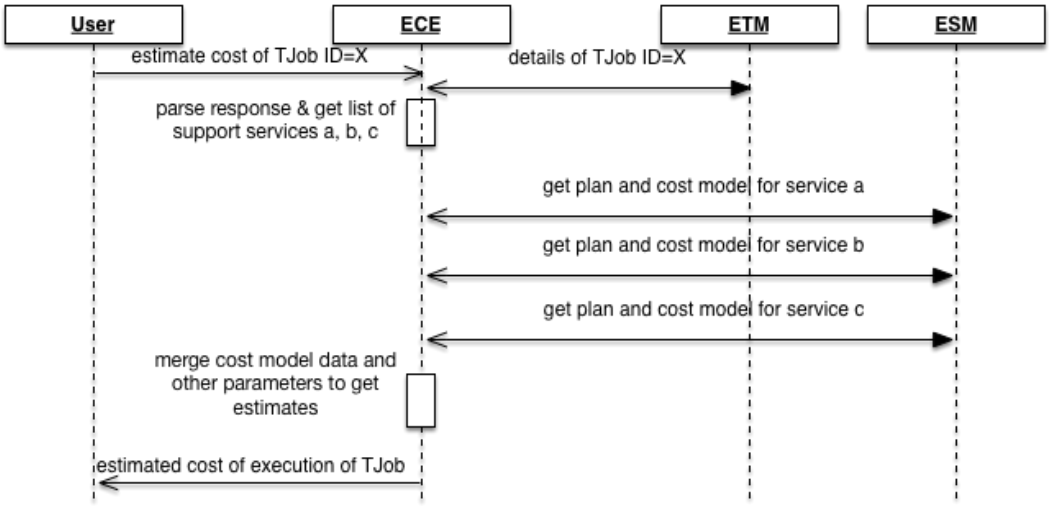

Figure 44. Sequence diagram showing steps in cost estimation process.

Figure 44 above shows the interactions involved in estimation execution cost for a selected TJob. Figure 45 shows the interaction between various ElasTest services and ECE for computation of actual costs for executing a TJob.

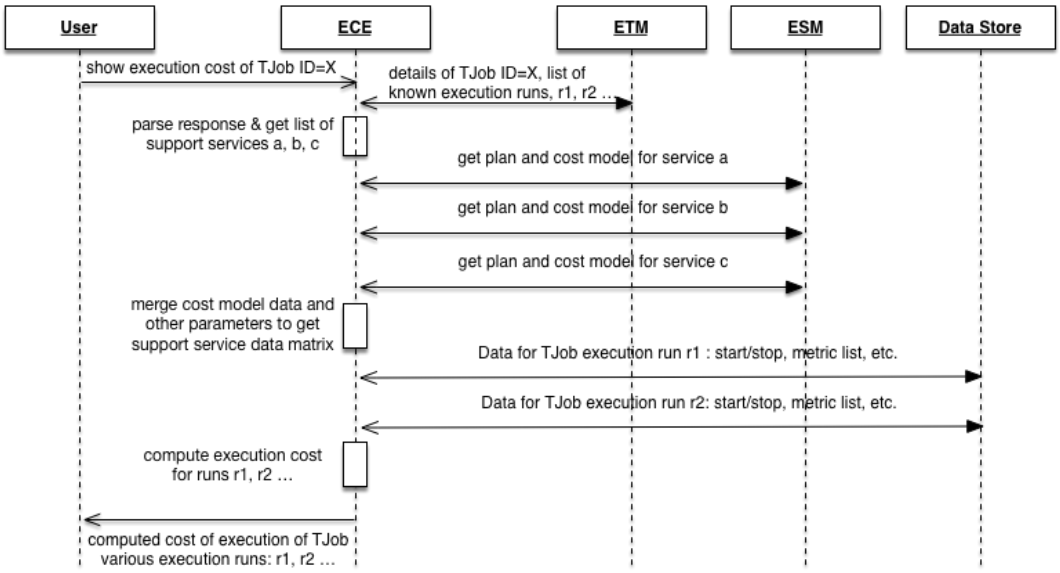

Figure 45. Sequence diagram showing steps for cost calculation of a TJob execution.

The cost model is described in Snippet 1. Example of ECE cost model. It forms the basis for estimation and computation by ECE for all TJob runs. The brief explanation of various elements is provided here (more detailed treatment can be found in WP4 deliverable):

- Components - list of external models this particular model depends on, currently it is empty
- Description - text describing the model
- Currency - ISO currency label, EUR for Euro, CHF for Swiss Francs, USD etc.
- Methodology - whether the cost is computed based on the time (duration) a resource was actively consumed, or if cost is computed on the true value of usage metric as observed by monitoring systems (usage).
- Model - whether the cost model is based on pay-as-you-go or on subscription style
- Model_param - if subscription model is chosen, this field then defines finer elements of subscription model. Currently it is undefined and is to be left empty.
- Fix_cost - this field captures in case there is a one time setup fee for configuring a service or not.
- Name - name of the cost model object
- Var_rate - this block captures the various physical resources and their cost-rates, the rates could be time linked, or it could be volume linked. Example of meters where rates on usage volume makes sense are: cpu_cycles, network_bytes_in, network_bytes_out, io_bytes_in, etc. Similarly, meter examples where time linked cost makes sense are: cpu-core, disk-space, ram-size, etc.

```
{
      "components": {},
      "description": "cost model for EMS support service",
      "currency": "EUR",
      "methodology": "duration",
      "model": "PAYG",
      "model_param": {},
      "fix cost": {
        "setup_cost":5
      },
      "name": "ems-model-A",
      "var_rate":{
        "cpu": 50,
        "cpu_unit": "core-hour"
        "disk": 1,
        "disk_unit": "gb-hour",
        "memory": 10,
        "memory_unit": "gb-hour"
      }
}
```

Snippet 1. Example of ECE cost model.

7.1.4 Interactions

The entries below captures the inputs to be received by ECE and the output from ECE to/from relevant actors.

Table 35. Input to ECE.

Input	Provided by
T-Jobs	TORM (ETM)
Cost Models	EPM, ESM
Monitoring Information for Status	EMS + TORM

Table 36. Output from ECE.

Output	Provided to
Cost Estimations	UI/API Response
Real Cost of running a T-Job	UI/API Response

7.2 ElasTest Recommendation Engine (ERE)

ElasTest Recommendation Engine (ERE) is a cognitive system designed to support software engineers developing automated test cases. It learns from the information gathered in software engineering repositories (historical test cases, test code features and code comments), and provides test code recommendations based on learned knowledge. The engine leverages machine learning algorithms to (1) generate Java code for new test cases from natural language descriptions provided by the user; and (2) recommend existing test cases suitable for reuse.

7.2.1 Objectives

The ultimate goal of ERE is to increase both the level of test coverage, and fault detection potential of the created test suites. This is achieved by:

- Recommending complete implementations of automated test cases based on natural language descriptions provided by users, so as to reduce time, effort and resources spent on test automation
- Allowing to directly control and guide the process of test automation, so as to avoid generating random test cases.
- Recommending a set of test cases suitable for reuse, so as to take most advantage of existing code resources.

7.2.2 System Prerequisites and Technical Requirements Specification

Requirements Specification

Table 37. ERE requirements.

Requirement	Description
REQ1: Generating test cases	Given a description of a test case in natural language, the system should generate new code for that test case

REQ2: Retrieving test cases for reuse	Given a description of a test case in natural language, the system should recommend a list of most relevant existing test cases for reuse
REQ3: Training custom models	User should be able to download own test data and train the machine learning model using that data
REQ4: GUI	The system should provide Admin UI (for loading data and training the model) and Tester UI (for accepting queries and displaying recommendations)

Prerequisites

In order to build custom machine learning models, Recommendation Test Engine requires training data that is domain- and/or system-specific, and hence must be provided by the user. The main source of data is project-specific software engineering repository.

7.2.3 Component Design

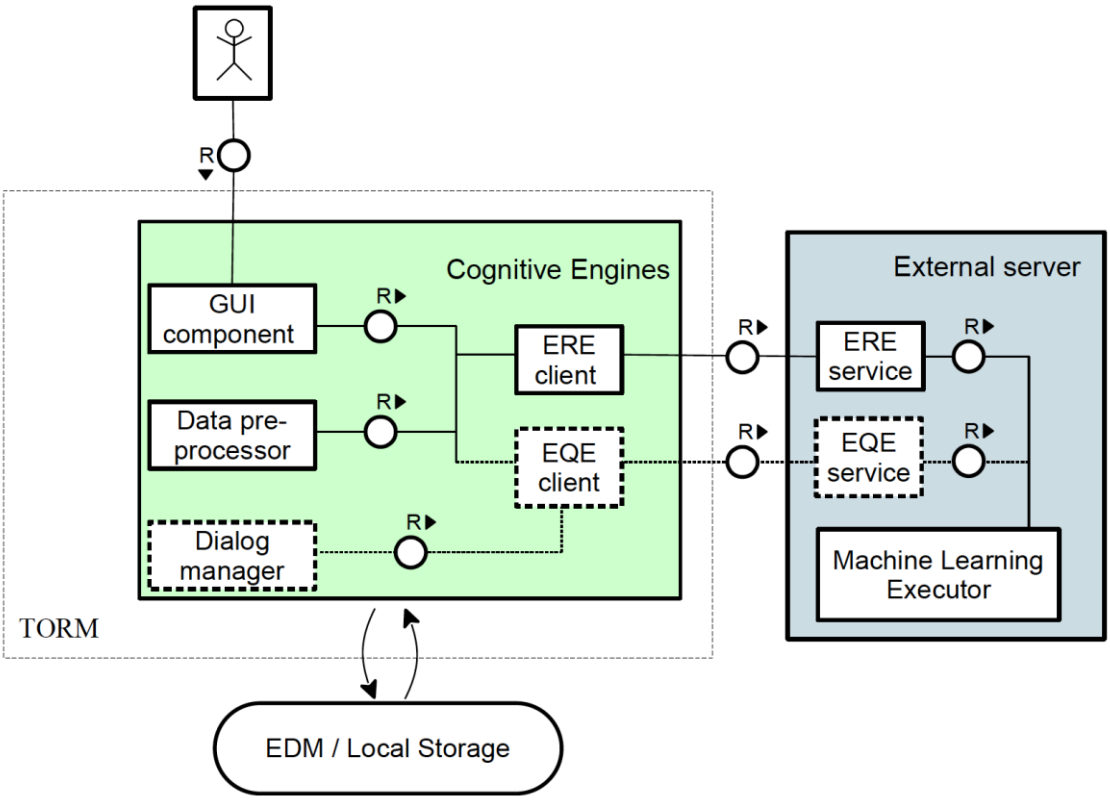

Figure 46. Cognitive Engines - FMC diagram.

Context Diagram

FMC diagram presented in Figure 46. shows the Recommendation Test Engine in the context of ElasTest components which it interacts with. The recommender system (ERE) is one of two Cognitive Engines that are hosted by the Test Orchestration and Recommendation Manager. The internal architecture of the recommender consists of:

- GUI component – embedded into TORM GUI. The interface comprises two separate parts: Recommender UI and Admin UI. Recommender UI is used by software testers and its main functions are: (1) to allow testers to input queries; (2) to display recommendations generated by the recommender system in response to user queries; (3) to select model to run queries against. Admin UI is used by system administrators and its main functions are to allow users (1) launch pre-processing of training data; (2) submit data for training (3) launch training of new machine learning models.
- Pre-processor – handles pre-processing of user data and saving them in format suitable for submitting for training.
- ERE client – manages communication with ERE service deployed externally.
- ERE services – to handle requests for machine learning tasks.
- Machine Learning Executor – the set of machine learning algorithms executed to (1) learn neural representations of source code tokens (2) to generate Java code from previously unseen natural language descriptions; (3) identify the set of related test cases suitable for reuse.

ERE use cases involve two human actors: a Tester and an Admin. Two main use cases for the Tester are:

- Receive recommendations on automating test cases
    - User selects adequate model from the available trained models.
    - User inputs a short description of the testing task.
    - The system generates recommended Java implementation of the test case.
- Receive recommendations on test cases to reuse.
    - User selects adequate model from the available trained models.
    - User inputs a short description of the testing task.
    - The system identifies a set of most relevant existing test cases for reuse, and present them to the user.

Main use cases involving Admin user are:

- Pre-process user data.
    - Admin user indicates the location of software engineering repository containing user data.
    - The system crawls the repository and extracts relevant data.
    - The system parses data to the format that can be consumed by machine learning algorithms.
- Train custom model:
    - Admin user selects pre-processed training data and submits them for training.
    - The system executes machine learning algorithms to train a new model.
    - The system reports the results of training.

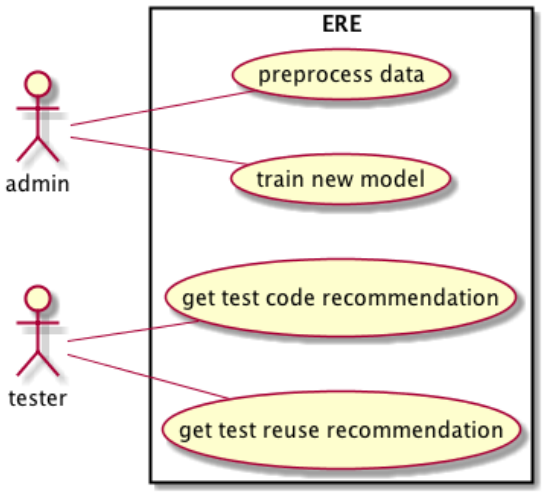

Figure 47. ERE use cases.

Figure 47 presents high level use-case diagram. Simplified sequence diagrams for Tester and Admin actions (assuming happy path – no errors) are presented in Figure 48 and Figure 49 respectively.

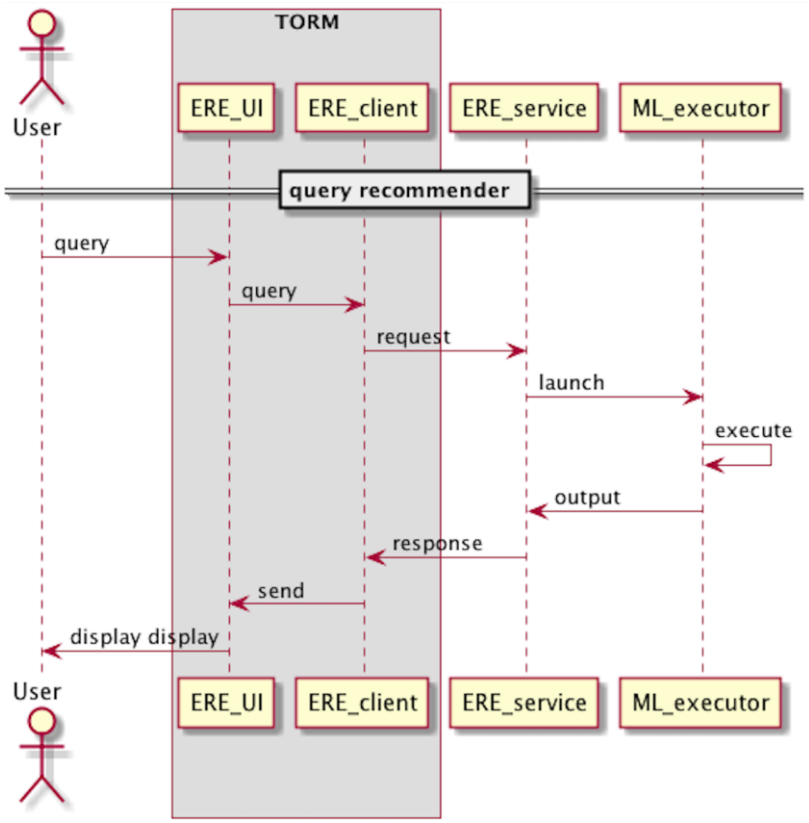

Figure 48. ERE sequence diagram (Tester actor).

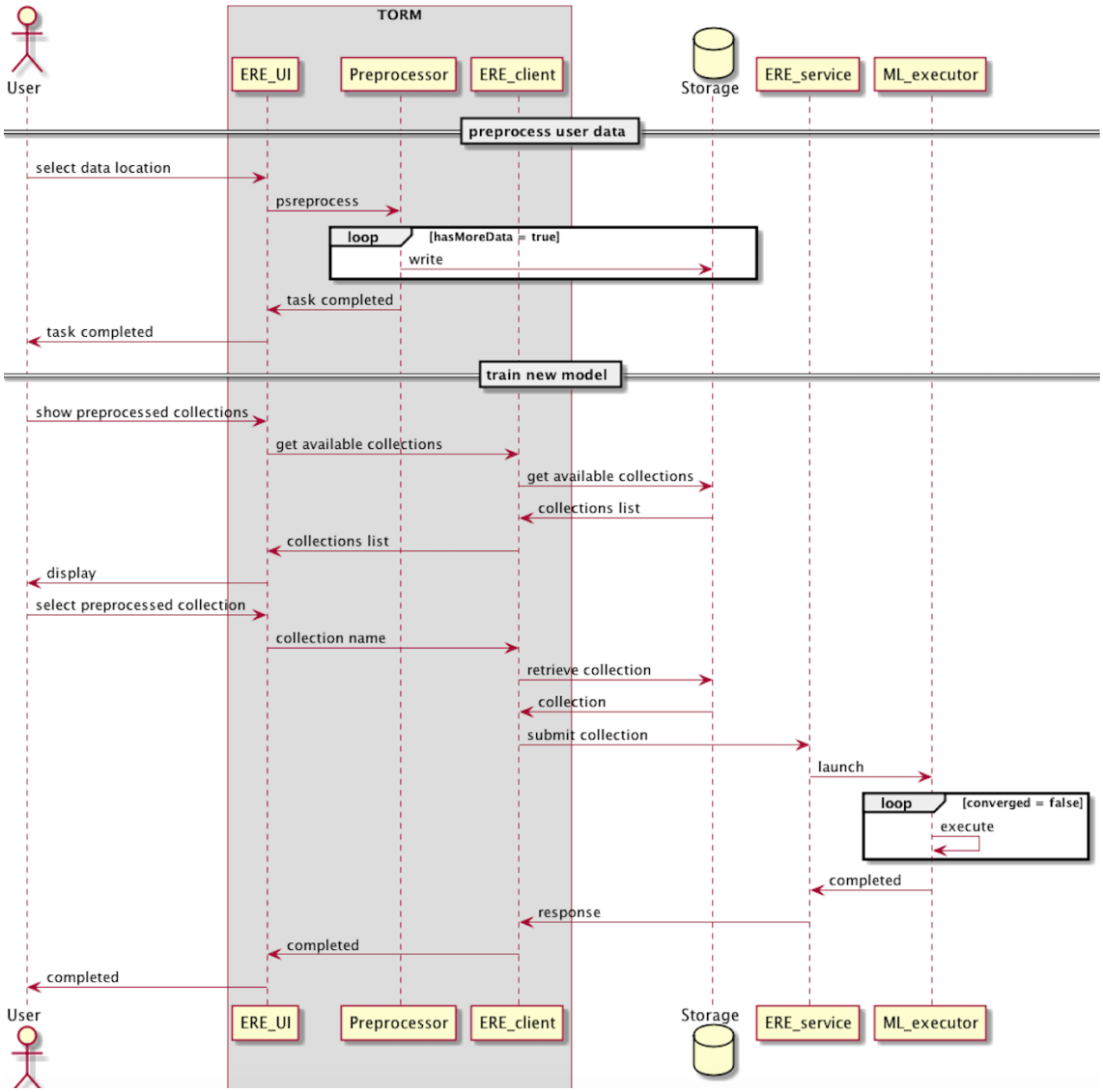

Figure 49. ERE sequence diagram (Admin actor).

7.2.4 Interactions

Table 38. Input to ERE.

Input	Provided by
Training data	User
Pre-processed data	EDM
Pre-process/Load/Train request	User (via ETM / ERE_UI)
Recommendation request	User (via ETM / ERE_UI)

Table 39. Output from ERE.

Output	Provided to
Pre-processed data	EDM

Test recommendations	User (via ETM / ERE_UI)

7.3 ElasTest Question & Answer Engine (EQE)

Cognitive Q&A Engine (EQE) allows testers and developers to ask questions in relation to testing. The Q&A Engine accepts questions asked in natural language and tries to identify user's intention or information need. This requires generating prompts for the user so that the user formulates the question more precisely or provides additional constraints. Having identified the intent, the system generates candidate answers, scored for best linguistic fit.

7.3.1 Objectives

ElasTest Q&A Engine builds on and extends ElasTest Recommendation Engine. Main objectives of EQE are as follows:

- To help user to take full advantage of ERE by advising on how to formulate efficient queries. Ambiguous queries to ERE may yield unsatisfactory results. The task for EQE is to prompt the user for additional information and suggest most adequate reformulation of queries.
- To leverage knowledge extracted from large open source repositories for providing natural language suggestions for new test cases required for the concrete SuT.

7.3.2 System Prerequisites and Technical Requirements Specification

Requirements specification

Table 40. EQE requirements.

Requirement	Description
REQ1: Advising on working with ERE	Assuming that user knows what feature they wish to test, help user to formulate efficient query for code recommendation
REQ2: Advising on new test cases to create	Assuming that user does not know which features/areas require testing, help user to identify most relevant test cases by proving natural language descriptions of new tests.
REQ3: GUI	Provide interactive UI interface for receiving user questions and displaying answers/prompts returned from the engine.

System prerequisites

EQE needs to adjust and match knowledge extracted from open source repositories for the needs of the specific SuT, and therefore it requires training data provided by the user. Access to project-specific software engineering repository (containing both test code and production code) is a prerequisite

7.3.3 Component Design

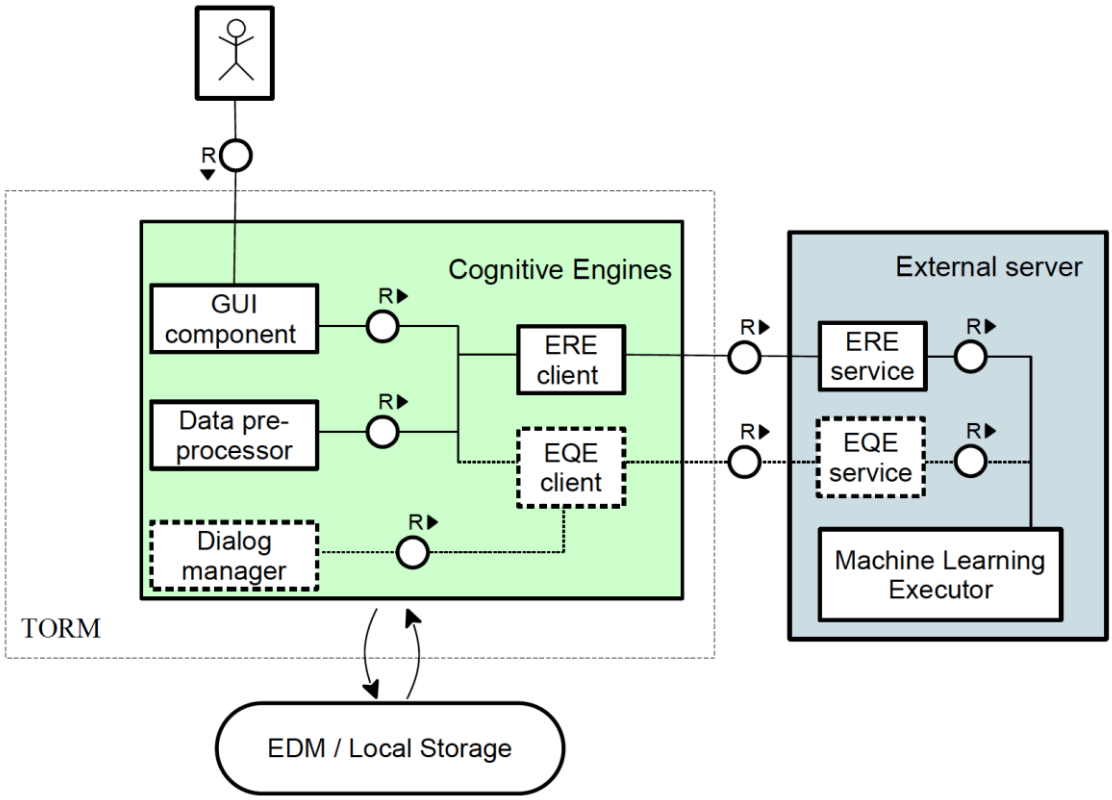

Figure 50. Cognitive Engines - FMC diagram.

Context Diagram

FMC diagram presented in Figure 50 shows EQE in context of ElasTest components which it interacts with. The Q&A system is one of two Cognitive Engines that are hosted by the Test Orchestration and Recommendation Manager. The internal architecture of the Q&A system consists of:

- GUI component – embedded into TORM GUI. The interface comprises two separate parts: Q&A UI and Admin UI. Q&A UI is used by software testers and its main functions is to allow user to interact with the system in a dialog-like manner. Admin UI is used by system administrators and its main functions are to allow users (1) launch pre-processing of training data; (2) submit data for training (3) launch training of new machine learning models.
- Pre-processor – shared with ERE; handles pre-processing of user data and saving them in format suitable for submitting for training.
- Dialog manager – enables a conversation between a user and the Q&A engine (a continuous exchange of messages that form a coherent flow).
- EQE client – manages communication with EQE service deployed externally.
- EQE services – handle requests for machine learning tasks.
- Machine Learning Executor – the set of machine learning algorithms executed to train and run a machine learning model capable of generating answers to user questions.

EQE use cases involve two human actors: a Tester and an Admin. Two main use cases for the Tester are:

- Ask for advices on working with ERE.
- Ask for advices on new test cases suitable for SuT.

Main use cases involving Admin user (shared with ERE) are as follows:

- Pre-process user data.
- Train custom model

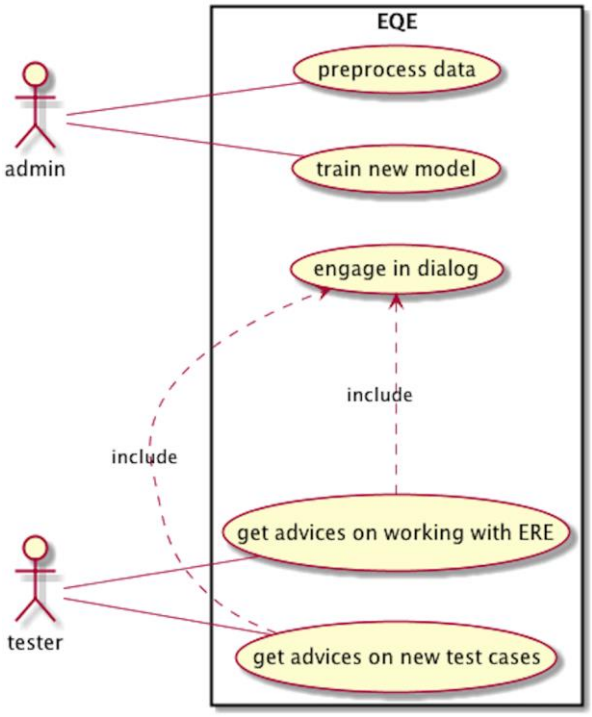

Figure 51. EQE use cases.

High level use-case diagram for EQE is shown Figure 51, whileTester and Admin actions are captured in sequence diagrams in Figure 52 and Figure 53.

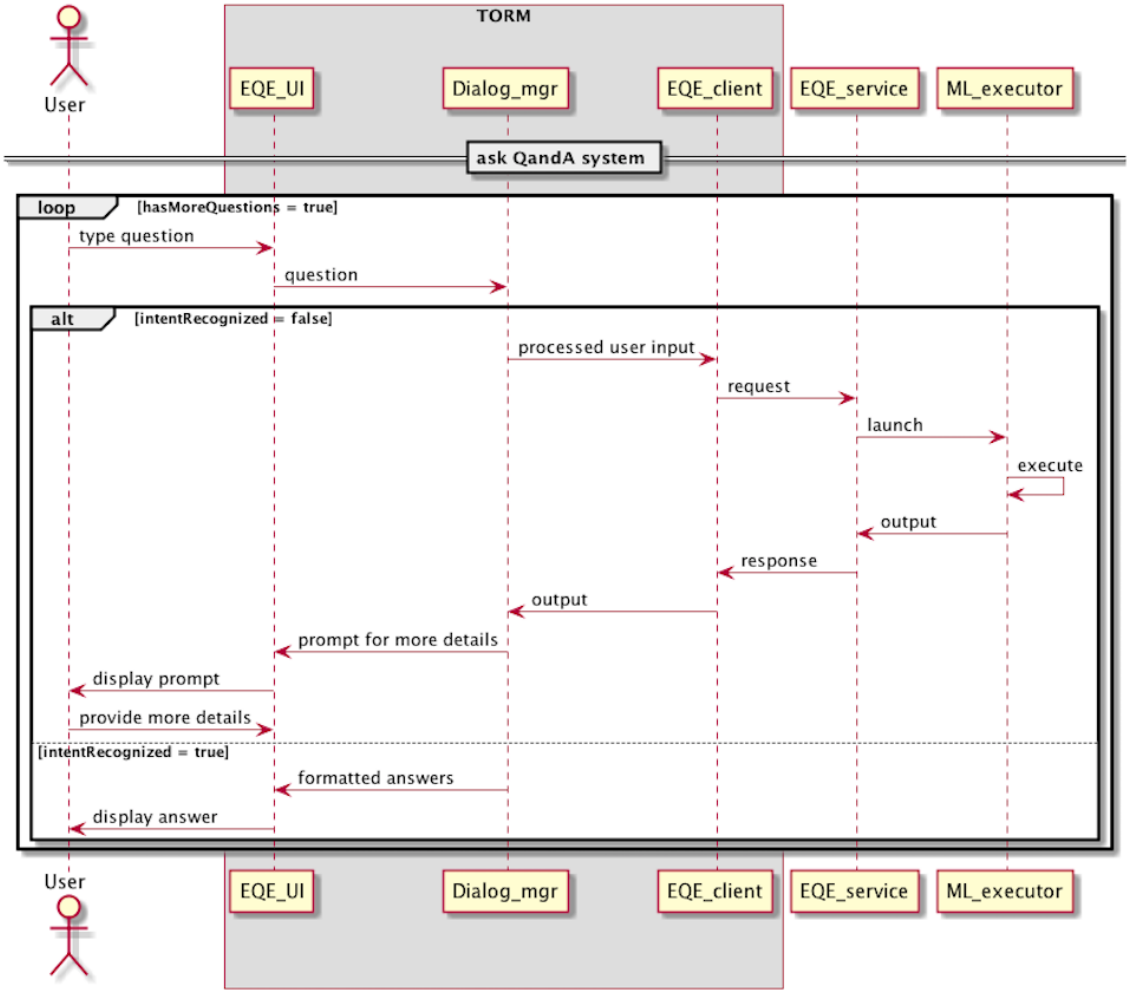

Figure 52. EQE sequence diagram (Tester actor).

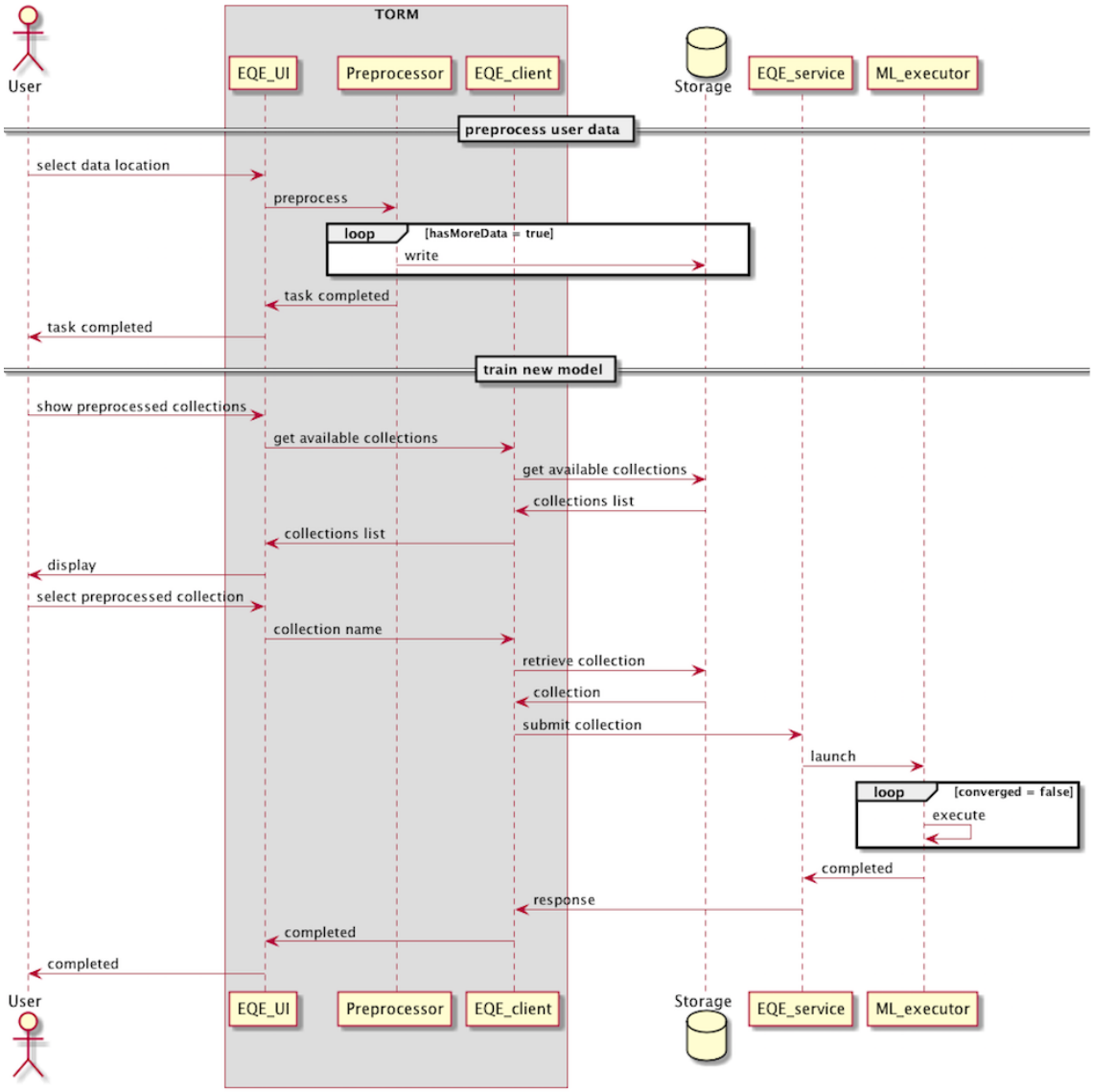

Figure 53. EQE sequence diagram (Admin actor).

7.3.4 Interactions

Table 41. Input to EQE.

Input	Provided by
Training data	User
Pre-trained knowledge representations (re-using outputs of recommender training)	ERE
Pre-process/Load/Train request	User (via ETM / EQE_UI)
Questions	User (via ETM / EQE_UI)

Table 42. Output from EQE.

Output	Provided to
Pre-processed data	EDM
Answers to questions	User (via ETM / EQE_UI)
Prompts for additional input	User (via ETM / EQE_UI)

7.4 ElasTest Orchestration Engine (EOE)

In ElasTest, the concept of *test orchestration* is understood as a novel way to select, order, and execute a group of TJobs. A TOJob (Test Orchestration Job) consists of a group of such TJobs, executing in coordination with the objective of validating high-level properties of the SUT. Hence, from a tester's perspective a TOJob can be seen as a "graph" of TJobs, where graph edges correspond with the execution of a TJob and graph nodes are checkpoints where synchronization and oracle verification for the composed TJobs can take place. The main objective of the test orchestration is to generate such graph, which involves composing opportunely the TJob inputs and inferring the overall outcome assertions.

The ElasTest core functionality is extended by means of the so called *engines*. The concept of test orchestration is implemented in the ElasTest component called ElasTest Orchestration Engine (EOE). The Test Orchestrator Engine (EOE) is the responsible to orchestrate individual TJobs to create TOJobs.

7.4.1 Objectives

The vision of test orchestration is to create richer test suites using the "divide and conquer" principle applied to testing, as hypothesized in the ElasTest DoA. To achieve this, two main mechanisms are proposed in ElasTest to implement test orchestration:

1. Topology generation. The objective is to combine intelligently TJobs for creating a more complete test suite, called TOJobs.
2. Test augmentation. This objective is to introduce additional TJobs to reproduce custom operational conditions of the SuT.

7.4.2 System Prerequisites and Technical Requirements Specification

On the one hand, the EOE's system prerequisites are the following:

- Java. The EOE component is going to be implemented as a Spring-Boot application, exposing its capabilities by means of a REST API. Therefore, EOE requires at least a JRE (Java Runtime Environment) installed in the machine hosting ElasTest.
- Docker. As usual, EOE is part of a the ElasTest microservices architecture based in Docker. Therefore, Docker engine is required to execute EOE as a Docker container.

7.4.3 Component Design

Figure 54, a high-level description of EOE and its relationships with the rest of the ElasTest component is depicted using a FMC diagram. These relationships are explained as follows:

- ElasTest Tests Manager (ETM). The EOE has a GUI part in ETM devoted to configure and execute TOJob. Therefore, ETM will be a client of the EOE API to expose its capabilities to ElasTest's users.
- ElasTest Data Manager (EDM). The topology of TOJobs needs to be stored somehow within ElasTest. For that reason, EOE needs to use the persistence layer of ElasTest provided by EDM.
- Test Support Services (TSS). EOE can behave as a proxy for TSS. The idea is that EOE intercepts calls from TJobs to TSSs to share sessions between all TJobs. For example, and supposing that the tests in the orchestration are using a browser provided by the ElasTest User impersonation Service (EUS), the browser is shared between all the tests within a TJob.

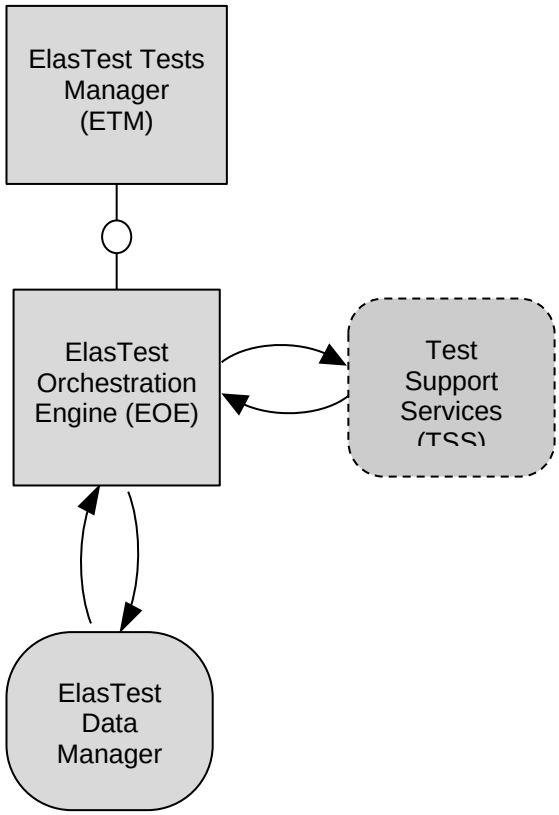

Figure 54. EOE FMC diagram.

7.4.4 Interactions

The following tables summarizes the relationships in terms of input and output from/to (respectively) the different ElasTest components to EOE.

Table 43. Input to EOE.

Input	Provided by
TOJob topology	ETM
TOJob topology	EDM
TJob input data	ETM
TSS data	TSS

Table 44. Output from EOE

Output	Provided to
TJob output data	ETM
TJob verdict	ETM
TOJob verdict	ETM

8 ElasTest Integrations with External Tools

These are the set of tools and plugins developed to facilitate the integration of ElasTest with external tools. For a deeper understanding of these facilities the reader is referred to D6.2 [8]. To understand the continuous integration libraries built to support these facilities, the reader is referred to D6.1 [7] of work package 6.

8.1 ElasTest Jenkins Plugin (EJ)

The ElasTest Jenkins Plugin (EJ) is developed to easy the use of the ElasTest features from of the most extended CI server open source. In that way the users with theirs Jobs configured on Jenkins will be able to use ElasTest effortlessly. To aim this, the plugin can be enable either in an standard Jenkins Job or in a pipeline Jenkins Job.

8.1.1 Objectives

The main objectives of EJ are listed below:

- The integration between Jenkins and ElasTest.
- Allow the use of ElasTest functionalities from a Jenkins Job.

8.1.2 System Prerequisites and Technical Requirements Specification

To run the EJ plugin as a main technical prerequisites is necessary:

- A Jenkins Server where the EJ will be installed.
- An ElasTest Platform to the EJ will be connecedt.

8.1.3 Component Design

This section describes the EJ component architecture and its interaction with ElasTest.

Context Diagram

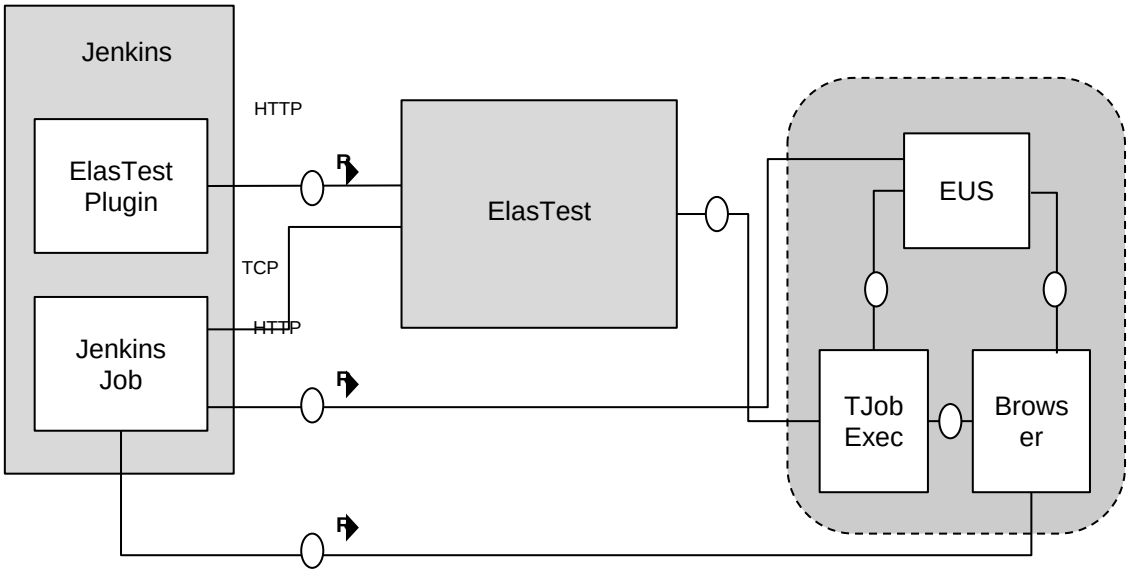

Figure 55. EJ FMC diagram.

In the diagram appears some components of the ElasTest Platform that are described below:

- **ElasTest Plugin:** Plugin developed to integrate Jenkins with ElasTest. Allows the Jobs executed on Jenkins send the execution logs to ElasTest and to request the necessary TSSs to ElasTest.
- **Jenkins Job:** A Job is a task that you create in Jenkins to do something, such as downloading your application's source code from a GitHub repository and compile it with maven. With the EJ, this Job will be replicate in ElasTest.
- **ElasTest:** Entity that represents the ElasTest Platform**.**
- **EUS:** TSS is responsible for providing browser instances ready for use by users, by TJobs stored in ElasTest or from a job run from Jenkins.
- **TJob Exec:** Specific execution of a TJob defined in ElasTest or in Jenkins. This entity will store the related data associated to that execution, such as logs, recordings,...
- **Browser:** Entity that represents the browser instances used in the tests defined in a TJob.

Use Cases

The following diagrams describe the main use cases of the EJ.

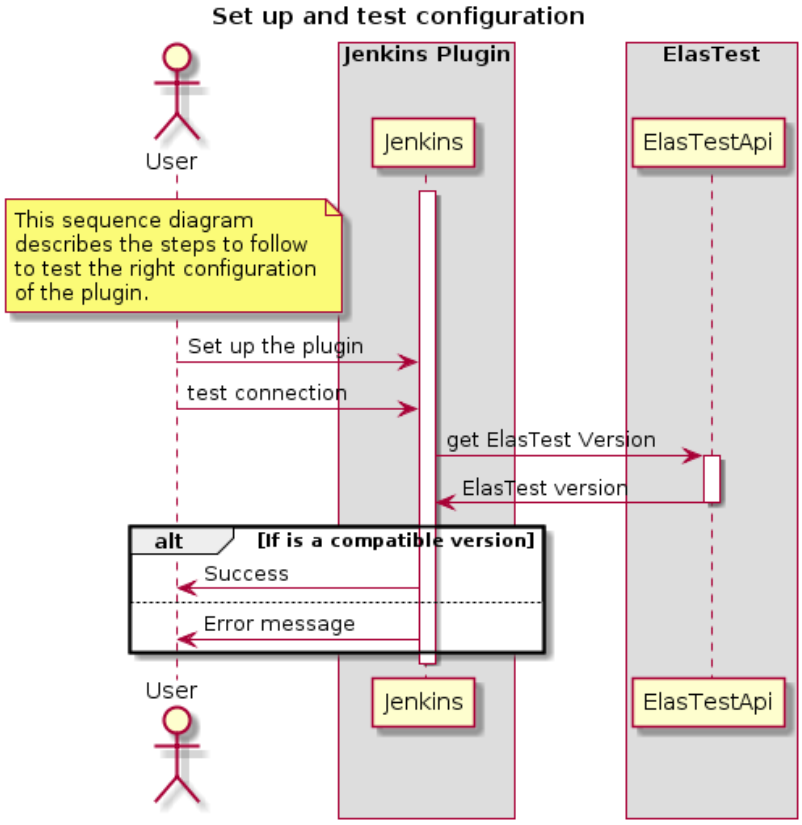

Figure 56. EJ use case sequence diagram - setup and test configuration.

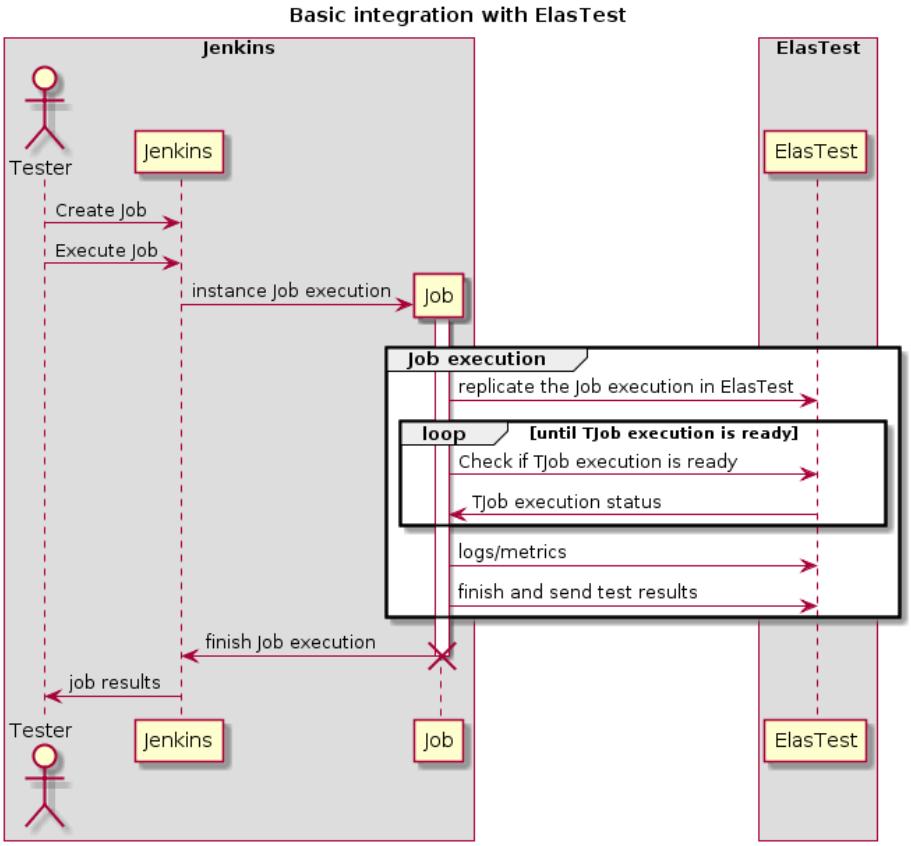

Figure 57. EJ use case sequence diagram - basic integration with ElasTest.

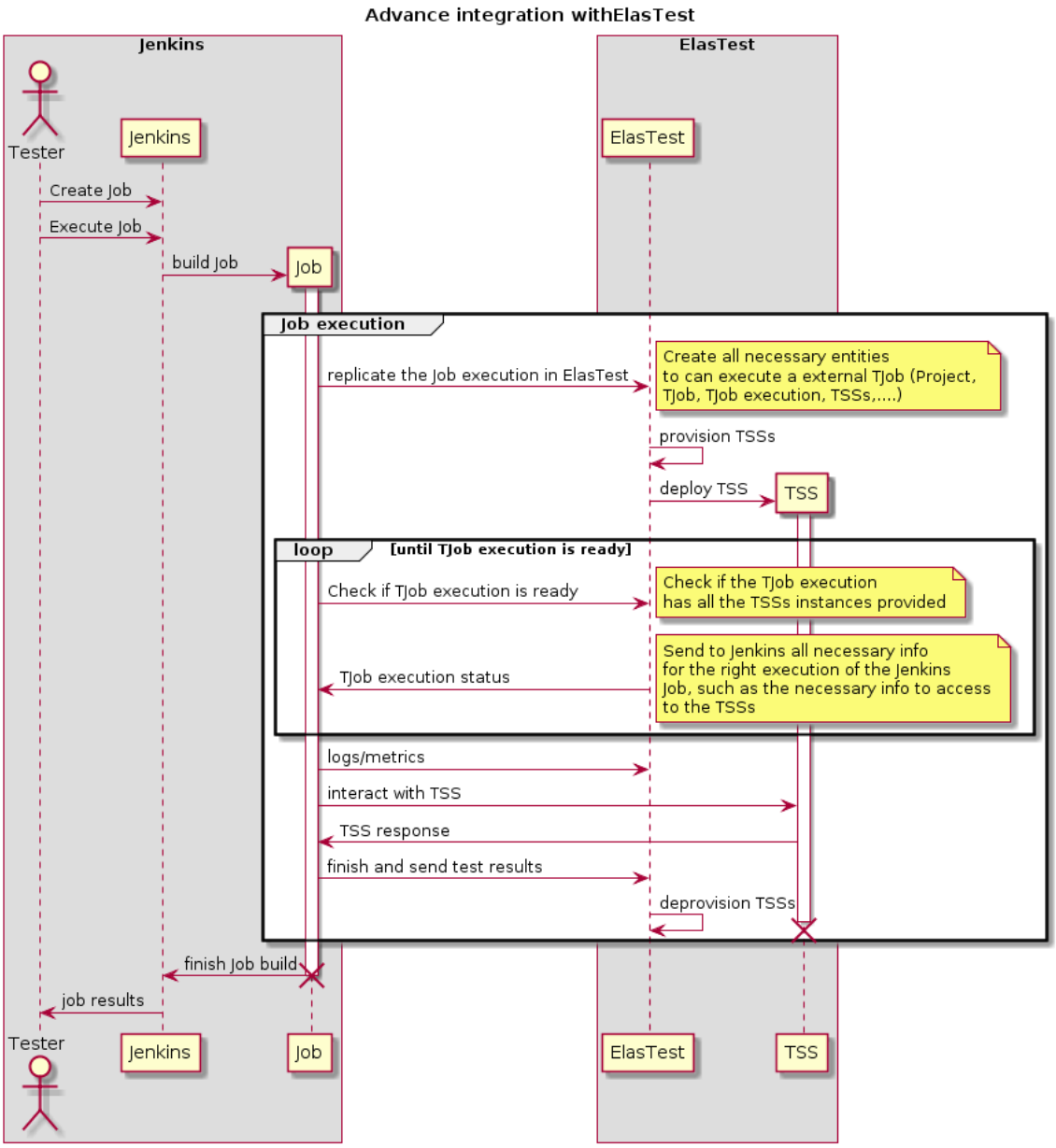

Figure 58. EJ use case sequence diagram - advance integration with ElasTest.

8.1.4 Interactions

Table 45. Input to EJ.

Output	Provided by	Provided to
Job definition	TestTer	EJ/Jenkins
TJob Execution data	ElasTest	EJ/Jenkins
TSS Data	ElasTest	Jenkins Job
Browser	EUS	Jenkins Job

Output	Provided by	Provided to
Job Logs	ElasTest	EJ
SuT Logs	ElasTest	Jenkins Job
SuT Metrics	ElasTest	Jenkins Job
Test results	ElasTest	EJ
TSS commands	TSS	Jenkins Job
Browser commands	Browser	Jenkins Job

8.2 ElasTest Toolbox (ET)

The ElasTest Toolbox (ET) is a dockerized application provided by ElasTest with a couple of administrative tools to manage ElasTest. The main functionalities that offer this application are the possibility of to start/stop ElasTest and update it to the next version in an easy way. In addition to these functionalities, ET offers another set of functionalities for internal use such as the script to prepare the release of a new stable version.

8.2.1 Objectives

The main objective of the ET is to provide tools to install and configure ElasTest in the easiest way possible.

8.2.2 System Prerequisites and Technical Requirements Specification

To run the ET you will need only to have docker installed on your machine with MAC OS, Windows or Linux OS. But on the other hand, remember that to run ElasTest without problems your machine need to comply with the following system specifications[11].

8.2.3 Component Design

This section describes the ET component architecture and its interaction with the rest of the services in ElasTest.

[11] https://elastest.io/docs/try-elastest/#system-specs

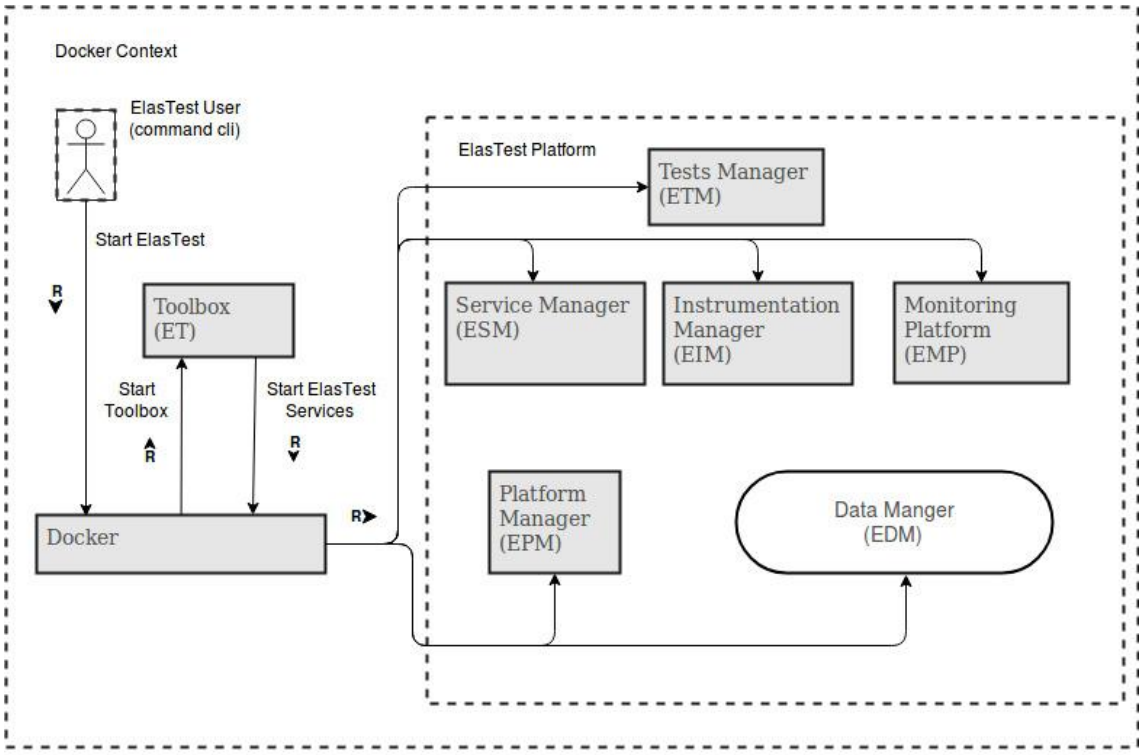

Figure 59. ET FMC diagram.

In Figure 59, appears the ET and the structural services started when ElasTest starts:

- **ElasTest Toolbox (ET):** The component used by the users to start and configure ElasTest in the easiest way possible.
- **Elastest Platform Manager (EPM):** The component that provides the ability to instantiate execution entities (like docker containers or virtual machines).
- **Elastest Service Manager (ESM):** The component that provides the ability to instantiate Test Support Services (TSSs) and Test Engines (TEs). EMS use the same underlying EPM to manage low level execute entities.
- **Elastest Data Manager (EDM):** The component that provides specialized persistence service to the rest of the platform.
- **Elastest Instrumentation Manager (EIM):** The component that allows ElasTest to instrumentalize already deployed SUT when tests are executed against it.

Use cases

Below you can see the main use cases of the Toolbox. Figure 60 describes what happens when the *start* command is executed.

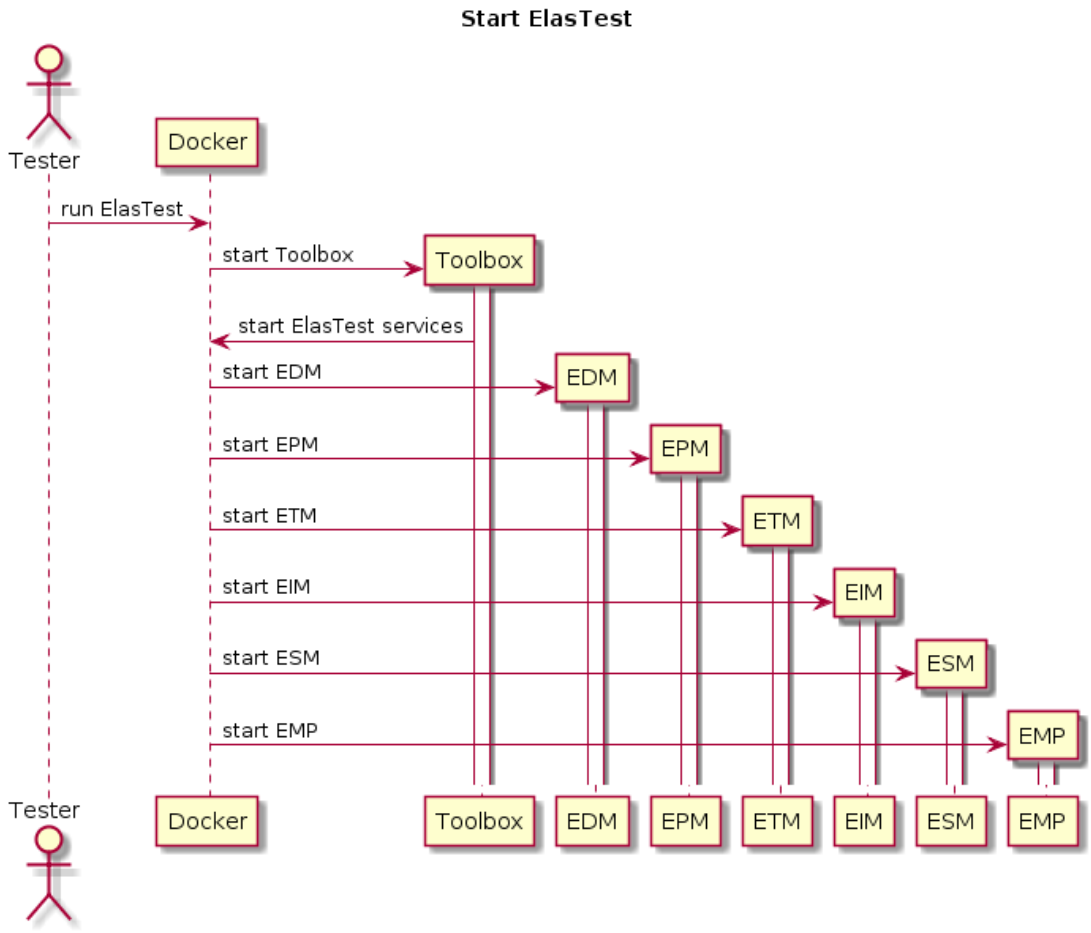

Figure 60. ET use case sequence diagram - start ElasTest.

In the Figure 61 that represents the execution of the stop command, you can see that the opposite happens than what is described in Figure 60.

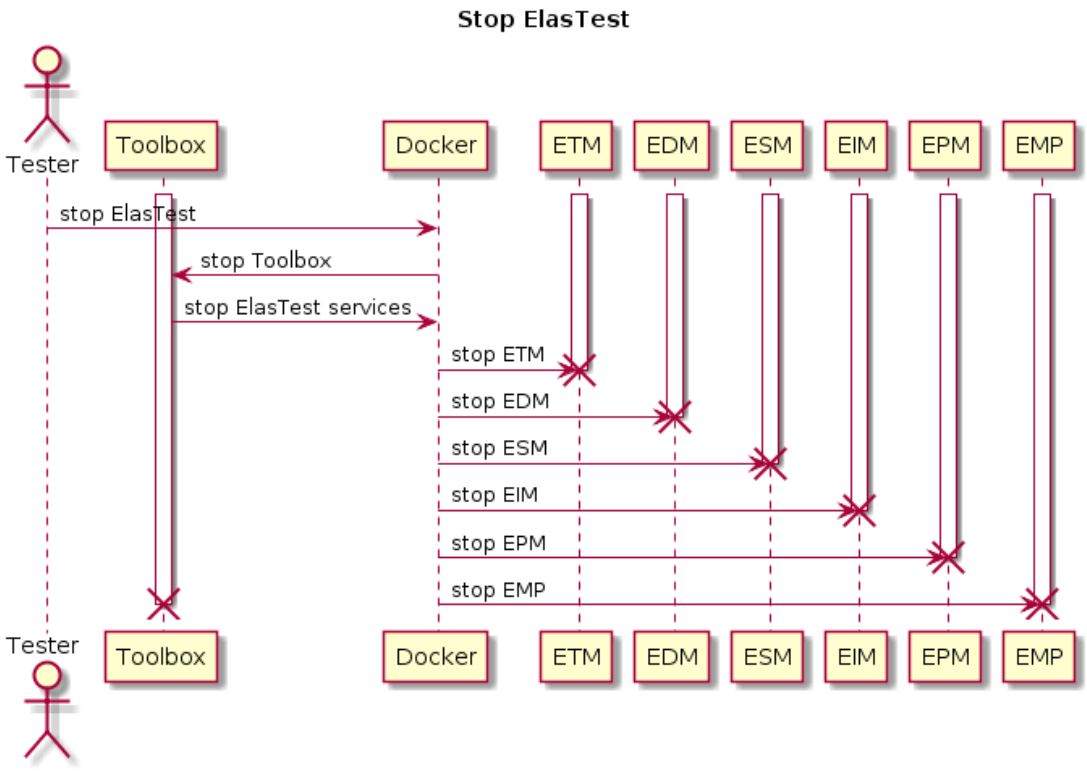

Figure 61. ET use case sequence diagram - stop ElasTest.

And in Figure 62 the update process is described.

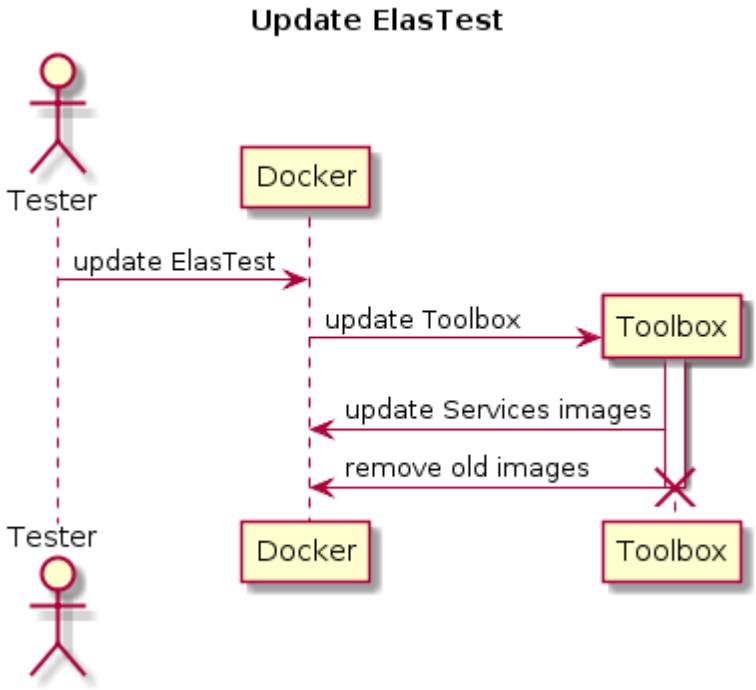

Figure 62. ET use case sequence diagram - update ElasTest.

8.2.4 Interactions

Table 46. Input to ET.

Input	Provided by
Start request	Command cli
Stop request	Command cli
Update	Command cli

Table 47. Output from ET.

Output	Provided to
Start ETM	ETM
Start EPM	EPM
Start EIM	EIM
Start EMP	EMP
Start EDM	EDM
Start	ESM

9 Conclusions and Future Work

This deliverable attempts to document the efforts of Task 2.2 and Task 2.3 of WP2. The focus of the deliverable, was to provide an overview of ElasTest. To this end, the deliverable has listed the user requirements and the use cases for the platform. Deriving from the use cases, an overall architecture was presented and associated technologies used to realize the platform. The classification of components as core, test support services and engines enabled for a better design of the platform.

The high level descriptions of the components were presented.

The tasks T2.2 and T2.3 will continue to focus on understanding and developing the platform which targets the end user. Collectively enhancing the requirements and thereby the architecture based on the needs of the demonstrator from WP7, the work intends to keep track of the development effort by using the build-measure-learn cycle.

10 References

[1] ElasTest project Description of Action (DoA) – part B. Amendment 1. Reference Ares(2017)343382. 23 January 2017.

[2] ElasTest Project. D2.2 SotA revision document v1. 30 June 2018.
[3] ElasTest Project. D3.1 ElasTest Platform cloud modules v1. 30 June 2018.
[4] ElasTest Project. D4.1 ElasTest Orchestration basic toolbox v1. 30 June 2018.

[5] ElasTest Project. D4.2 ElasTest recommendation engines v1. 30 June 2018.
[6] ElasTest Project. D5.1 ElasTest Test Support Services v1. 30 June 2018.
[7] ElasTest Project. D6.1 ElasTest Continuous Integration and Validation Systemv1. 30 June 2018.
[8] ElasTest Project. D6.2 ElasTest platform toolbox and integrations v1. 30 June 2018.
[9] Compositional Structures, Block diagrams – reference sheet, http://www.fmc-modeling.org/download/notation_reference/Reference_Sheet-Block_Diagram.pdf