D6.2	
Version	1.0
Author	NAEVATEC, URJC
Dissemination	PU
Date	29-06-2018
Status	FINAL

D6.2 ElasTest platform toolbox and integrations

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	WP6
WP leader	Guiomar Tuñón de Hita
Deliverable nature	Other
Lead editor	URJC
Planned delivery date	30-06-2018
Actual delivery date	29-06-2018
Keywords	Open source software, cloud computing, software engineering, operating systems, computer languages, software design & development

Funded by the European Union

License

Contributors

Name	Affiliation
Guiomar Tuñón	NAEVATEC
Francisco Ramón Díaz	URJC
Eduardo Jiménez	URJC
Pablo Fuente Pérez	URJC

Version history

Version	Date	Author(s)	Description of changes
0.1	22/01/2018	Guiomar Tuñón	Initial version (DRAFT)
0.2	31/05/2018	Francisco R. Díaz Eduardo Jiménez	Added Jenkins Plugin, Toolbox, TestLink and how it's work deploy on AWS.
0.3	07/06/2018	Pablo Fuente Pérez	Complete sections
1.0	29/06/2018	Pablo Fuente Pérez	Final revision

Table of contents

List of figures

List of tables

Glossary of acronyms

Acronym	Definition
API	Application Programming Interface
AWS	Amazon Web Services
CI	Continuous Integration
CLI	Command Line Interface
CPU	Central Processing Unit
DoA	Description of Actions
EC2	Elastic Compute Cloud
ECE	ElasTest Cost Engine
EDM	ElasTest Data Manager
EDS	ElasTest Device Emulator Service
EJ	ElasTest Jenkins Plugin

EMP	ElasTest Monitoring Platform
EMS	ElasTest Monitoring Service
EPM	ElasTest Platform Manager
ERE	ElasTest Recommendation Engine
ESS	ElasTest Security Service
EUS	ElasTest User Emulator Service
FOSS	Free and Open-Source software
HTML	HyperText Markup Language
IaC	Infrastructure as Code
JSON	JavaScript Object Notation
NPM	Node.js Package Manager
RAM	Random Access Memory
REST	Representational State Transfer
RSA	Rivest Shamir Adleman
SotA	State of the Art
SPA	Single Page Application
SUT	System Under Test
TE	Test Engines
TE	Test Engine
TJob	Testing Job
TSS	Test Support Service
URL	Uniform Resource Locator
YAML	YAML Ain't Markup Language

1 Executive summary

The present document describes all the software artefacts of the ElasTest Toolbox enabling the seamless installation and administration of ElasTest in different platforms. In the current version, ElasTest can be easily installed on a machine with Docker engine available and also on Amazon Web Services (AWS) cloud provider. Another important part of the document is tailored to describe the integrations of ElasTest with external tools. In the current version, ElasTest provides integrations with Jenkins CI system and with TestLink Test management system.

The rest of the document is structured as follows. In section 2, the strategic context and objectives of ElasTest Toolbox and integrations are described. Section 3 is tailored to ElasTest Toolbox with different sections for installing with Docker and deploying in AWS cloud provider. In section 4, the integration with external tools, such as, Jenkins and TestLink is explained. Finally, section 5 includes the conclusions and future work, and section 6 contains the references.

2 Strategic context and objectives

The ElasTest Project DoA defines two main tasks related to the ElasTest platform Toolbox and Integrations in WP6. Let us quote literally the description of tasks here to fix the context of the tasks that need to be accomplished:

Regarding to ElasTest installers, the DoA states the following:

> ***Task 6.3: ElasTest platform toolbox***
>
> *This task shall be in charge of creating the appropriate mechanism and tools suitable for distributing ElasTest artifacts inside and outside the consortium. As a result of executing this task, developers should be able to install and use ElasTest in a seamless way. For this, this task shall distinguish two types of situations:*
>
> *• Distribution of ElasTest FOSS artifacts. The ElasTest platform and many of its modules shall be released basing on FOSS licenses. For the associated software artifacts, we shall use the widely accepted FOSS mechanism for software distribution including robust versioning mechanisms as well as repositories such as Maven Central (for Java artifacts), NPM and Bower (for JavaScript repositories), Docker Hub (for Docker images), Launchpad (for Debian/Ubuntu packages), etc.*
>
> *• Distribution of ElasTest proprietary artifacts. For the non-FOSS artifacts, the project needs to provide the appropriate distribution mechanisms that shall be designed and implemented in this task.*
>
> *For complying with this, this task shall assume all additional developments that are necessary for the installation, administration and management of ElasTest as a whole. This task shall also assume the generation of the documentation and guidelines enabling successful installation and use of ElasTest.*

The consortium has decided to distribute almost[1] all ElasTest components as Docker containers. Docker provides a clean and simple way to package software and is a widely accepted distribution platform for open source software artifacts. As a result of this, the standard FOSS repository is DockerHub to publish all ElasTest binary artefacts. Nevertheless, since DockerHub is a marketplace for providing images only, the orchestration of all the containers must be managed separately including the download of required images and starting them in the correct order by taking care of the dependency resolution among all the components. To achieve this task, a new component called "ElasTest Platform" has been developed. ElasTest Platform is also distributed and executed as a Docker container and needs Docker engine installed to be used. For that reason, to make the installation of ElasTest in a cloud provider's server even easier, we have developed a CloudFormation description file to deploy ElasTest in AWS. These two components will be described in more detail in section 3.

With regard to ElasTest integrations, the DoA states the following:

> ***Task 6.4: ElasTest toolbox external integrations***
>
> *As specified in Section 1.1 on Part B of the DoA of this GA, we want ElasTest to be compatible with current SotA CI tools and methodologies so that developers can use it without disrupting their common practices. For this, we shall create the appropriate modules fully integrating ElasTest into, at least, one popular CI tool so that ElasTest can be used as a plugin of it. The specific CI tool to be used needs to comply with the following requirements:*
>
> *• It must be a FOSS CI tool so that the ElasTest FOSS strategy is strengthened by this integration.*
>
> *• It must be a very popular CI tool having a strong community spread worldwide.*
>
> *• The tool must provide an API enabling the creation of extensions and plugins into it.*
>
> *This task assumes the responsibility of 1) selecting the appropriate CI tool, 2) developing the appropriate CI extensions and plugins enabling the use of ElasTest into it, 3) validating the suitability and stability of the plugins and extensions along the whole duration of the project.*

We have studied [4] different sources about software development tools usage and all share that today Jenkins[2] is the most used open source continuous integration system. This tool meets all the requirements specified in the DoA, then we have selected it to integrate with ElasTest. Moreover, used the lean methodology applied in the project, we have found that some potential users are interested also in the integration of ElasTest with open source test management tools. For that reason, ElasTest has been integrated with TestLink[3]. Section 4 contains detailed descriptions of these integrations.

[1] Some components are distributed with other formats, like ElasTest Jenkins plugin.
[2] https://jenkins.io/
[3] http://testlink.org/

3 ElasTest Toolbox

ElasTest Toolbox is an umbrella for all the tools, artefacts and procedures designed to facilitate the installation of ElasTest in different environments. The toolbox can be described in a high-level way with the following points:

- ElasTest components are executed as Docker containers and delivered using standard Docker image registries. Open source containers are published in DockerHub registry.
- ElasTest Platform is a component distributed as a Docker container designed to orchestrate the downloading and execution of the rest of the component containers.
- ElasTest can be installed easily in any mayor operating system (Linux, Windows or Mac) using a single command with the only requirement of having installed standard Docker tools.
- ElasTest can be deployed easily in AWS cloud provider using a CloudFormation template - the standard mechanism to deploy complex systems in that provider. The support of more cloud platforms (public and private) are planned for future releases. One of the most interesting ones is Kubernetes for its popularity.

In the next subsections, all these main points will be described in more detail.

3.1 System Requirements

For now, ElasTest is intended to be deployed on a dedicated server due to the considerable number of modules and technologies that are part of it. Table 1 shows the recommended system specifications:

Recommended system specifications	
Processor	1GHz or faster
RAM	8GB (highly recommended 16GB)
SWAP	4GB (if RAM < 16GB)
Hard Disk	30GB

Table 1. Recommended system specifications

These requirements are very high and avoid to try ElasTest in a basic development machine. For that reason, we have planned to create a new slim down version of ElasTest (called “mini”) replacing some components for other less powerful ones but with less resource requirements.

3.2 ElasTest distribution selected technologies

Currently the distribution of ElasTest is based on Docker containers [1]. This technology provides a lightweight, stand-alone, executable package of each of the components of ElasTest that includes everything needed to run it: code, system tools, system libraries, and settings. We have selected Docker containers for the following reasons:

- Available in Linux, Mac and Windows operating systems.
- Containerized software will always run the same, regardless of the environment.
- Deployment of ElasTest containers can be done easily and interdependence between themselves is easily configured.

- Different configurations for ElasTest can be deployed by selecting which components containers would be launched.

3.3 ElasTest Platform container

The "ElasTest Platform" is a Docker container that would manage the retrieval, running of each of the components and also will provide a way to interact with the whole platform. It allows to install and execute ElasTest as a whole in a system with Docker installed by running the following command:

```
$ docker run --rm -v /var/run/docker.sock:/var/run/docker.sock elastest/platform
start
```

It is implemented using Python programming language. The Python scripts are used to analyse command line options to execute ElasTest in different modes. Basically, every mode defines what components shall be started. ElasTest Platform container uses the docker-compose tool under the hood. In this way, every ElasTest component can be composed by several containers if needed. All the configuration needed to execute every component is described in a docker-compose.yml file stored in its GitHub repository.

ElasTest Platform container is being developed in the ElasTest Toolbox GitHub repository[4].

3.3.1 Execution modes

ElasTest has three operating modes:

- **normal**: Currently this mode is the lightest of the three and it is the default operating mode. In this mode, you can only use the ElasTest User Emulator Service (EUS) as a Test Support Services (TSSs) and you won't be able to use any of the Test Engines (TE). It is ideal if you want to try ElasTest to test the core features and test web applications.
- **experimental-lite:** This mode allows the user to use any of the TSS provided by ElasTest by default: ElasTest Security Service (ESS), ElasTest Monitoring Service (EMS), ElasTest Device Emulator Service (EDS) and EUS. Also, any TE can be used. The available TEs are the ElasTest Cost Engine (ECE) and the ElasTest Recommendation Engine (ERE). Please note that ERE can only be used if the ElasTest private components repository have been configured in the machine. Detailed instructions about this repository and how to configure it are given in D6.1: ElasTest Continuous Integration and Validation System. The limitations of this mode are that ElasTest Monitoring Platform (EMP), ElasTest Platform Manager (EPM) and the full version of ElasTest Data Manager (EDM) are not executed. This is recommended to test all the features but it is not recommended for production use.
- **experimental:** In this mode ElasTest is executed with all the components. It is the mode recommended for production use, but it is ideal to execute in a server

[4] https://github.com/elastest/elastest-toolbox

with high computing resources. It can take several minutes to start all services provided by ElasTest.

3.3.2 Commands

ElasTest Platform container has several commands to perform different actions:

- **Start:** Start ElasTest and install it if it is not installed.
- **Stop:** Stop ElasTest if it is in execution
- **Wait:** Blocks the command until ElasTest is ready to be used. This command is useful for scripts automating the start and stop of ElasTest.
- **Inspect:** Shows information about the running ElasTest.
- **Update:** Execute the necessary steps to migrate the persistent data of ElasTest and download new version of the components.
- **Help:** Show all available commands.

These commands are described in the following subsections.

3.3.2.1 Start command

The command "start" is used to start ElasTest. If ElasTest components are not available in the machine, this command also downloads the components. The command blocks until ElasTest is ready to be used and prints the URL where ElasTest GUI is accessible.

In the case ElasTest is not started for some error or it takes too long to be ready, the command will print an informative message and will exit with non-zero result.

The user can use the following options for the configuration of the running ElasTest platform:

- **--mode=<mode>:** This will configure the ElasTest to run with any of the previous explained modes. Valid values for the mode option are: "normal", "experimental-lite" and "experimental".
- **--server-address=<PUBLIC_IP>:** In order to execute ElasTest in a server for remote usage this option should be used. If ElasTest will be used only locally, this is not necessary.
- **--user=<user> and --password=<password>:** ElasTest currently supports Http basic authentication, when these two options are used in the ElasTest start the authentication will be automatically activated.
- **--version=<version_tag>:** ElasTest can be forced to be launched with a specific version. If a version is not selected, by default the *latest* version will be used. To check available versions, go to: https://hub.docker.com/r/elastest/platform/tags/.
- **--pullall:** Force refreshing all the components to the latest version not using the ones found in the computer if any.
- **--pullcore:** Force refreshing only the core components latest versions not using the ones found in the computer if any.
- **--noports:** No binds service ports to host.
- **--logs:** Shows during the start the logs of all the components.

- **--testlink** or **-tl**: when this option is passed, a TestLink service will be started as part of ElasTest platform.
- **--help:** This option will show the help for the start command:

```
usage: docker run -v /var/run/docker.sock:/var/run/docker.sock --rm elastest/platform
start [-h] [--mode {experimental,experimental-lite,normal}] [--dev]
              [--pullall] [--pullcore] [--noports] [--logs]
              [--server-address SERVER_ADDRESS] [--user USER]
              [--password PASSWORD] [--testlink]
              {start,pull-images,stop,update}

positional arguments:
  {start,pull-images,stop,update}
                        Platform command to execute: start or stop

optional arguments:
  -h, --help            show this help message and exit
  --mode {experimental,experimental-lite,normal}, -m {experimental,experimental-
lite,normal}
                        Set ElasTest execution mode. Usage:
                        --mode=experimental
  --dev, -d             Configure ElasTest for development.
  --pullall, -pa        Force pull of all images. Usage: --pullall
  --pullcore, -pc       Force pull of only necessary images. Usage: --pullcore
  --noports, -np        Unbind all ports. Usage: --noports
  --logs, -l            Show logs of all containers. Usage: --logs
  --server-address SERVER_ADDRESS, -sa SERVER_ADDRESS
                        Set server address Env Var. Usage: --server-
                        address=XXXXXX
  --user USER, -u USER  Set the user to access ElasTest. Use together
                        --password. Usage: --user=testuser
  --password PASSWORD, -p PASSWORD
                        Set the user password to access ElasTest. Use together
                        --user. Usage: --password=passuser
  --testlink, -tl       Start the TestLink Tool integrated with ElasTest.
                        Usage: --testlink
```

ElasTest `start` command by default blocks the shell until ElasTest is stopped. This is generally desirable when a developer uses the shell to start ElasTest, but it is not very convenient if ElasTest has to be started within a script, for example in a CI system.

When `start` command is executed with `-d` docker option (detached), the command shows the container id just created and returns immediately. ElasTest is executed in background:

```
docker run -d --rm -v /var/run/docker.sock:/var/run/docker.sock elastest/platform
start
```

Using detached mode, the ElasTest platform command "wait" can be used to wait until ElasTest is ready to be used. This command will be described below in the document.

3.3.2.2 Stop command

If ElasTest is started with `start` command without detached mode, the shell is blocked. If the user hits Ctrl+C in this shell, ElasTest is stopped in a controlled manner. If ElasTest is started in detached mode, the `stop` command can be used.

```
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock elastest/platform stop
```

3.3.2.3 Wait command

When the platform is launched detached, user won't have feedback of when ElasTest is ready to be used. For this reason, the platform container provides a way to check if the platform has already started. A timeout can be configured in this command.

```
docker run -v /var/run/docker.sock:/var/run/docker.sock --rm elastest/platform wait
```

The wait command can be configured for different outcomes with the following options:

- **--container=<seconds>:** specifies the seconds that the command should be waiting to the ETM container to be created until exit. If the ETM container is created while waiting or it had been created before the execution of the command, it will return 0. Otherwise it would return a value different than 0.
- **--running=<seconds>:** specifies the seconds that the commando should be waiting to the ETM service to be ready to be used. If the ETM service is started while waiting or it had been started before the execution of the command, it will return 0. Otherwise it would return <> 0.
- **--help:** This option will show the help for the wait command:

```
usage: docker run -v /var/run/docker.sock:/var/run/docker.sock --rm elastest/platform
wait [-h] [--container CONTAINER] [--running RUNNING]

optional arguments:
  -h, --help            show this help message and exit
  --container CONTAINER, -c CONTAINER
                        Sets timeout in seconds for wait to the ETM container
                        creation. Usage: --container=240
  --running RUNNING, -r RUNNING
                        Sets timeout in seconds for wait to ETM is running.
                        Usage: --running=290
```

3.3.2.4 Inspect command

Once ElasTest is started, it is important to know what the URL to reach it is. The command `inspect` will return information about ElasTest. At the moment, the only information returned is the graphical user interface URL obtained with `--api` parameter.

```
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock elastest/platform
inspect --api
```

The command is prepared to be extended so information that users could require from the ElasTest platform container would be added to the information that the inspect command can provide.

3.3.2.5 Update command

If you already have an ElasTest version installed on your computer and you want to upgrade to the latest released version, you can use the *update* command.

```
usage: docker run -v /var/run/docker.sock:/var/run/docker.sock --rm elastest/platform
update [-h] [--mode {experimental,experimental-lite,normal}]

optional arguments:
  -h, --help            show this help message and exit
  --mode {experimental,experimental-lite,normal}, -m {experimental,experimental-
lite,normal}
                        Set ElasTest execution mode. Usage:
```

```
                        --mode=experimental
```

This only works if a pre-installed version is already available.

3.3.2.6 Help command

This command will provide general information about the available options of the ElasTest tool-box.

```
docker run -v /var/run/docker.sock:/var/run/docker.sock --rm elastest/platform -h
usage: docker run -v /var/run/docker.sock:/var/run/docker.sock --rm elastest/platform
[-h] {pull-images,inspect,stop,update,start,wait}

positional arguments:
  {pull-images,inspect,stop,update,start,wait}
                        Instruction to execute
optional arguments:
  -h, --help            show this help message and exit
```

3.4 Architecture

This section describes how ElasTest platform container is architected and how its modules interact. Platform container is made up by multiple Python files, but the most important are the following:

- **main:** as the name suggests, is the central file that parses received arguments to execute commands and is responsible for calling the corresponding python scripts of the "second level": run, update and pull.
- **run:** is used to start or stop ElasTest components. It makes use of "third level" components, like **setEnv** to modify environment variables from docker-compose of the ET services, **ETImages** to get services' image or **checkETM** to wait until ETM is ready.
- **update** and **pull:** are used to update ElasTest. They make use of **DockerUtils** to update ElasTest components' docker images and **ETImages** to obtain those images through Platform container service files.

Figure 1 shows the components and their interactions.

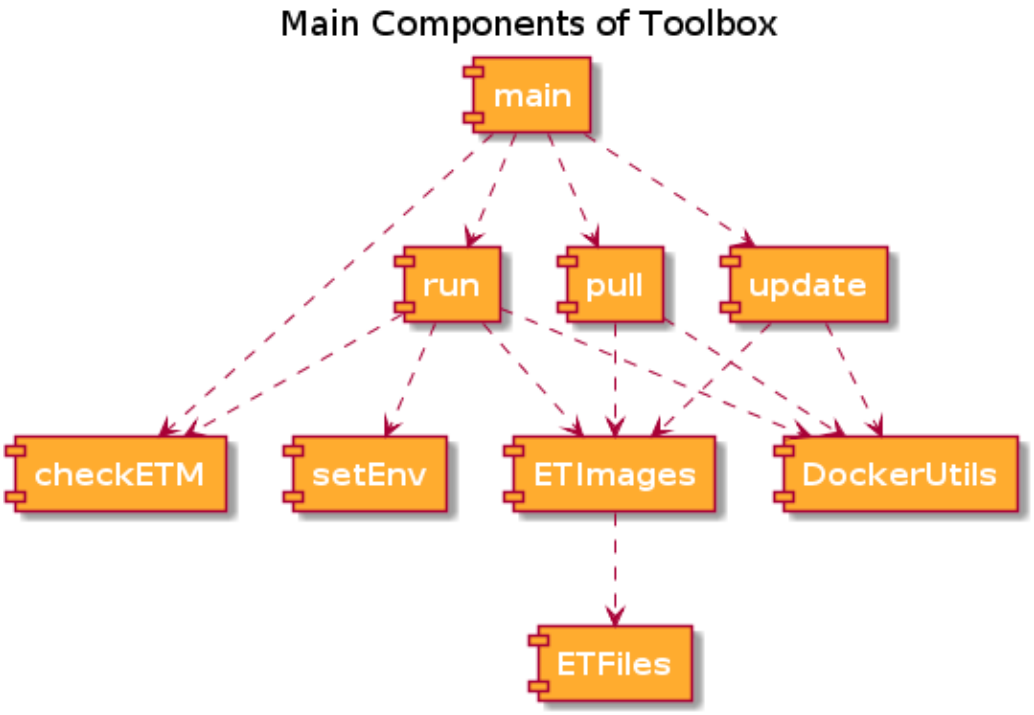

Figure 1. Toolbox Component Diagram

All scripts interact directly or indirectly with Docker, Filesystem and execute commands in shell as you can see in the Figure 2.

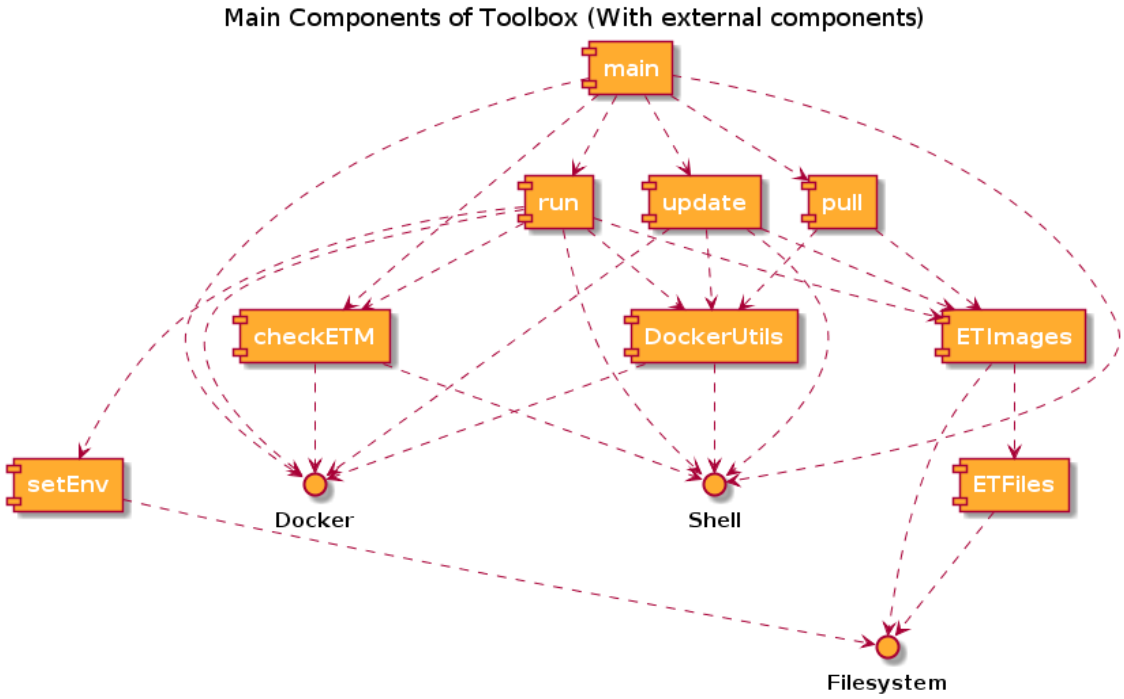

Figure 2. Toolbox Component Diagram

3.5 Interactions

The modules described in the previous section interact between them to provide the requested features. In the following subsections, the interactions in every command are described.

3.5.1 Start command

When the user executes platform container with `start` command, ElasTest core components will be started. Figure 3 shows the interaction between platform container modules when this command is used. As it can be seen, platform container will parse command line options, will get images of the corresponding components and start them. During this process, it will inform the user through console messages and, when the ElasTest is ready to be used, it will print the URL where it is accessible.

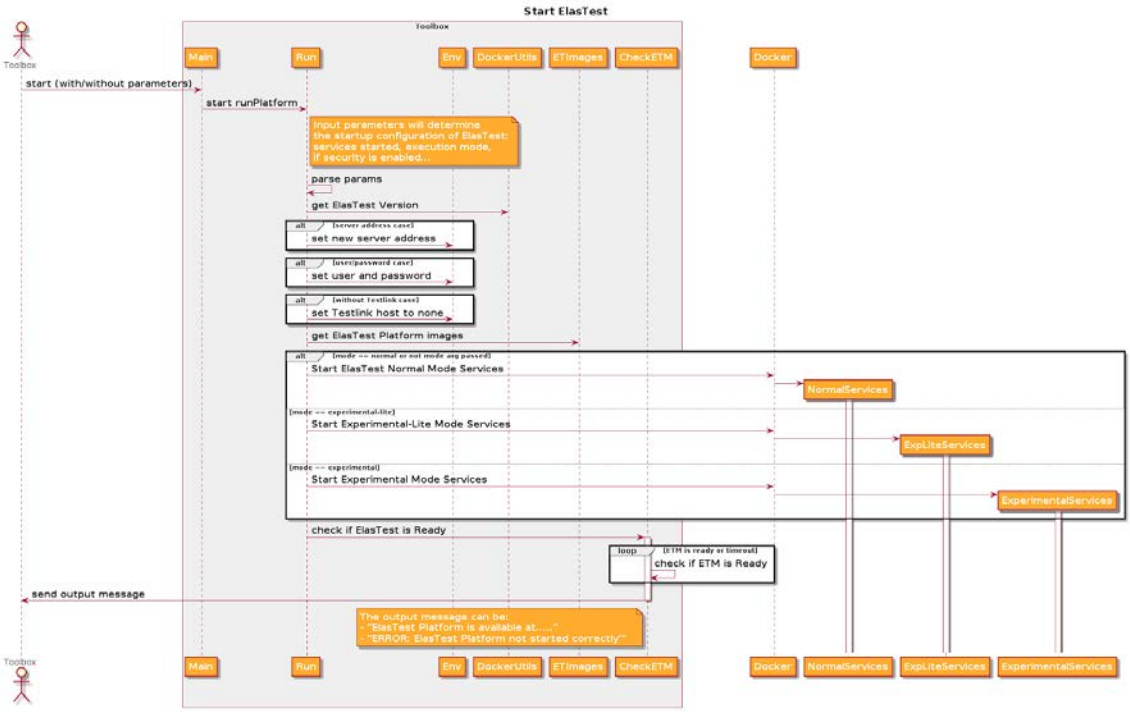

Figure 3. Platform container "start" command

The message "Start ElasTest Normal Mode Services" of the diagram of Figure 3 represents the action of execute docker-compose command to start the docker-compose files used to define ElasTest core components. Because docker-compose command is in charge of pulling needed images and start the containers these low-level actions are not shown in the diagram. The rest of the messages of type "Start ElasTest … Mode Services" perform similar actions but using a different set of core components.

3.5.2 Stop command

Once ElasTest is started, the user can stop it pressing Ctrl + C in the shell. She also can stop it executing the platform container using `stop` command. This command sends a SIGTERM signal to the platform container launched with `start` command. Then, it terminates all components. These interactions can be seen in Figure 4:

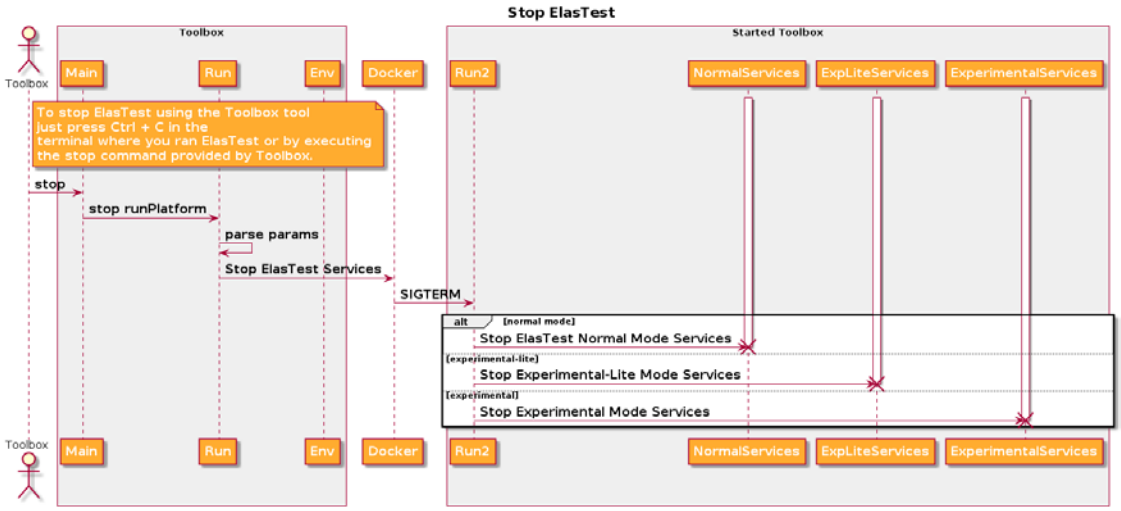

Figure 4. Platform container "stop" command

The messages "Stop … Mode Services" in Figure 4 are basically a call to docker-compose down command, to stop the containers started with the previous docker-compose up command.

3.5.3 Update command

The `update` command is used to update all component docker images to the latest version. It only updates the components already downloaded in the machine. Also, you can select the components to update using the `mode` option as with the `start` command.

Figure 5 shows what happens when the user runs platform container with `update` command. First it checks the ElasTest version and then asks to the user if he wants to continue with installation. If so, platform container checks if ElasTest is already running and if so, informs the user about the need to stop it and asks for confirmation. If the answer is yes ("Y"), `update` container sends SIGTERM to the platform container running ElasTest to stop it and starts the update process. This process is composed by several steps:

1. The `platform_services` volume will be removed, as it contains the descriptions of all Test Support Services.
2. All the core components' Docker images are updated with the new version.
3. Finally, it gets the already pulled ElasTest images by given mode and update them to the last version. To do that, a new platform container is launched with the `pull-images` command (view Figure 6).

When pull is done, platform container shows to the user the message "Update finished successfully" and ends the execution.

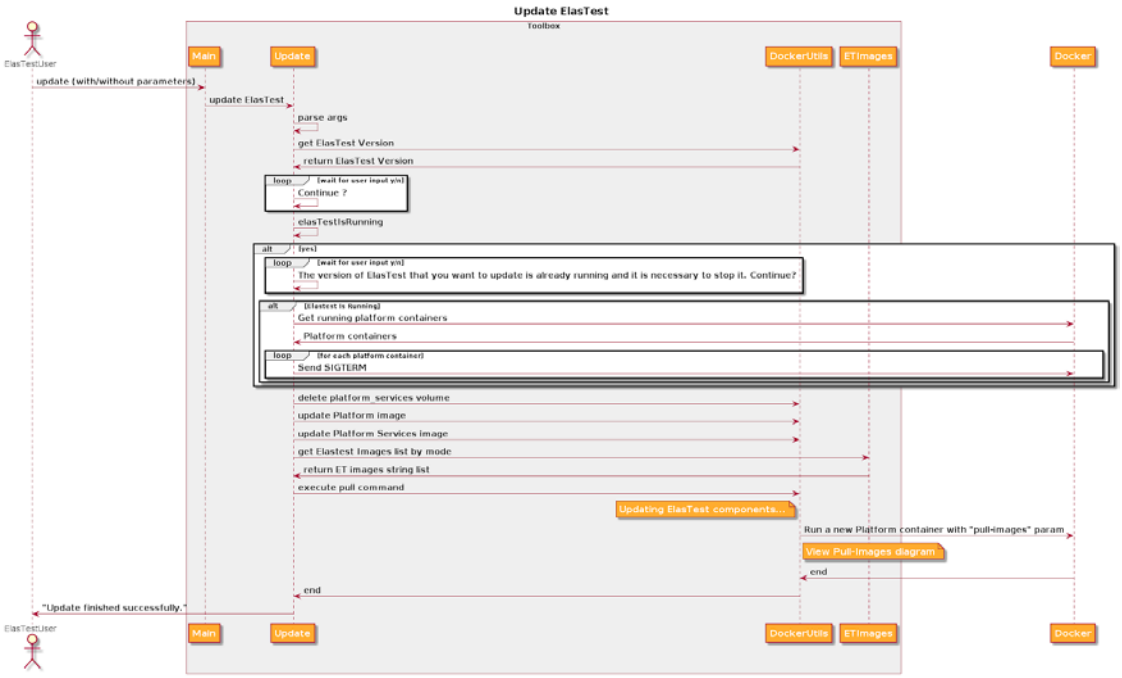

Figure 5. Platform container "update" command

3.5.4 Pull-images command

When `pull-images` command (Figure 6) is executed (by means of `update` command) first gets the ElasTest images of the given mode. Then, for each one of those images, checks if it already exists locally and if so do pull. Next, old images will be removed and for last, dangling images too.

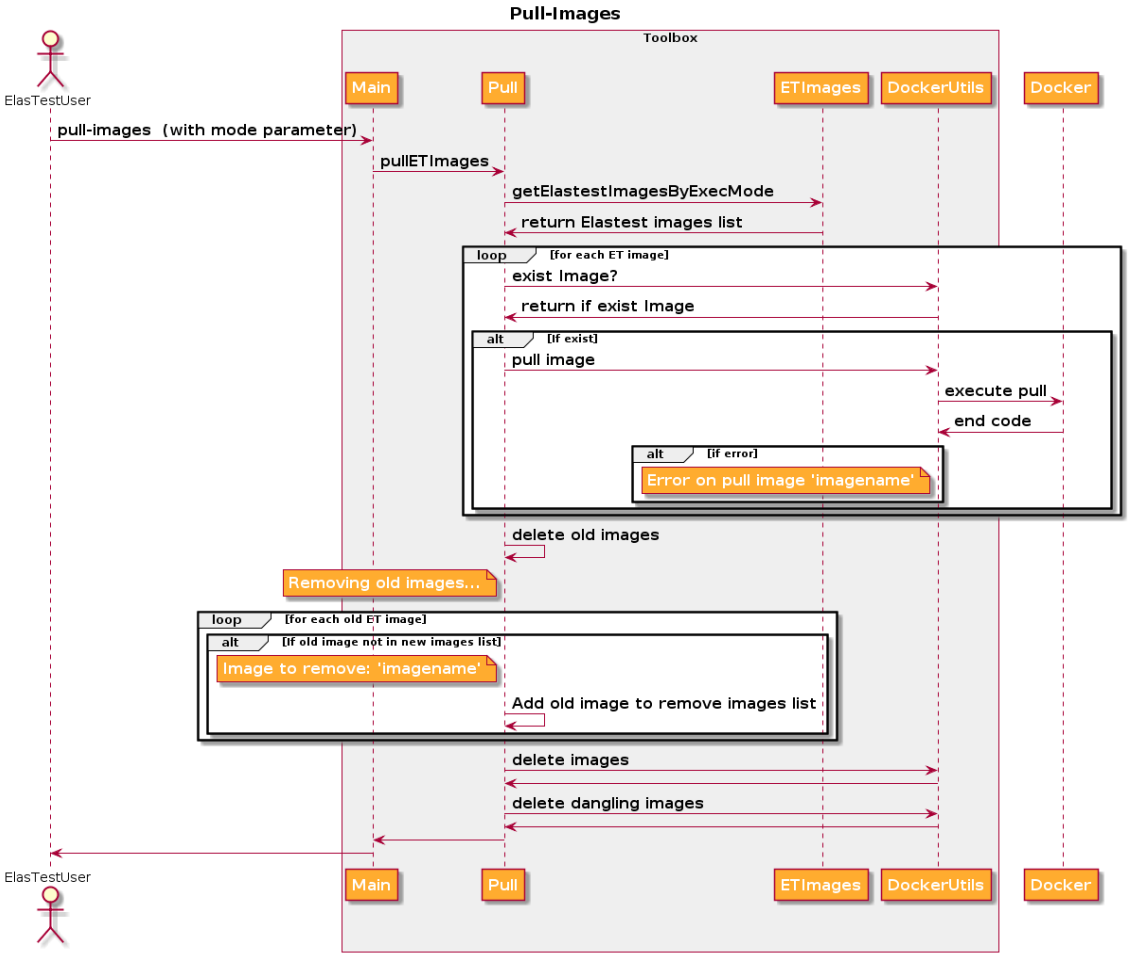

Figure 6. Platform container "pull-images" command

3.6 ElasTest on AWS deployment

Simplifying the installation and configuration process, and to provide alternatives for the deployment in different cloud environments is one of the major concerns. Cloud platforms are very easy to use because a template file can be used to deploy some ready to use virtual machine with selected software already pre-installed. Among all the available cloud providers and private platforms, AWS have been selected as the first supported cloud provider to deploy ElasTest.

The most straightforward way to deploy ElasTest in AWS is using a CloudFormation file. This file can be used with AWS CLI tools or AWS Console, a graphical web interface. This file contains the description of all resources needed to deploy services in AWS. In the current version, ElasTest CloudFormation file specifies that the platform is deployed in a single EC2 instance. In the future versions, it is planned to allow ElasTest deployment in a cluster of nodes.

ElasTest CloudFormation uses platform container under the hood to install and manage ElasTest in the EC2 instance.

3.6.1 How to deploy ElasTest in AWS

Using the AWS Console the ElasTest platform can be deployed filling an online web form. The necessary steps are:

1. Open AWS Console with a valid AWS Account
2. Go to AWS CloudFormation dashboard and create a new stack.
3. Select the option “Choose a template” and use the ElasTest CloudFormation file [2]. Figure 7 shows a screenshot of the creation stack page.
4. Fill the form with the information as listed in Table 2.
5. Deploy your stack. No more configuration is needed. Click on “Next -> Next -> Create”
6. Stack status will show *CREATE_IN_PROGRESS*. Wait a few minutes until it shows *CREATE_COMPLETE*. Then check the *Output* tab to see the URL where ElasTest is available. Figure 8 shows a screenshot of the output tab.

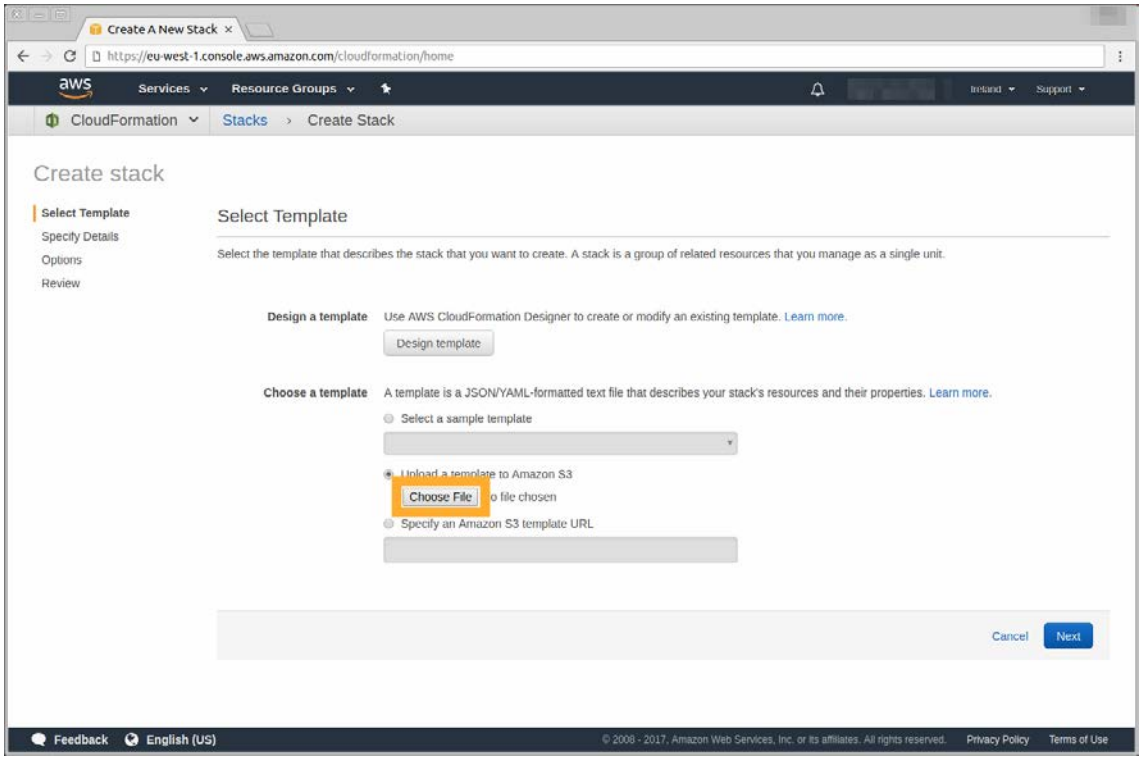

Figure 7. AWS Create stack page

Parameter	Value	Details
Stack name	The name of the stack	For example, “Elastest”
ElastestExecutionMode	*normal*, *experimental-lite* or *experimental*	Choose Elastest execution mode
ElastestPassword	Your password	Password to access the platform
ElastestUsername	Your username	Username to access the platform

ElastestVersion	*latest*	Which version of elastest do you want to launch. ***latest*** version points to the last stable release of ElasTest, so it is always safe to use
InstanceType	The type of machine you want (recommended at least *m4.large*)	Elastest needs high resources to run
KeyName	One of your AWS keys	RSA key to access the instance through SSH to execute maintenance commands
SwapSize	Recommended at least *4*	The amount of swap memory in GB

Table 2. CloudFormation options

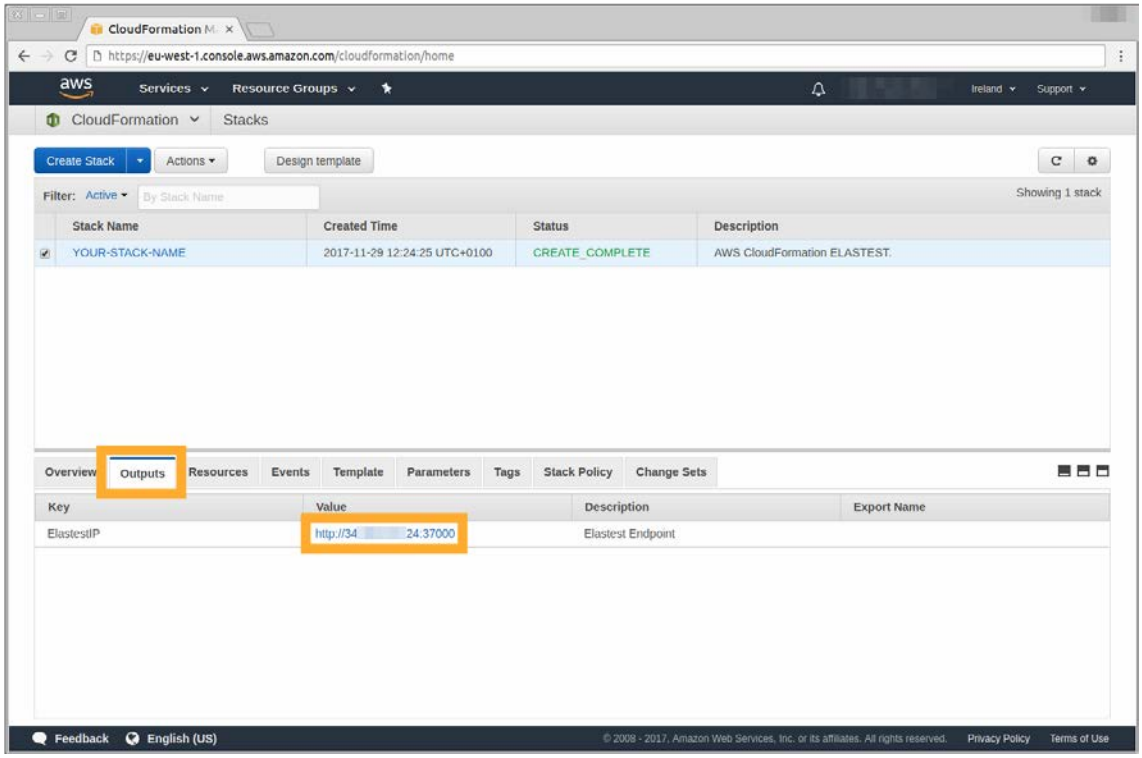

Figure 8. AWS Stacks page with output tab opened

3.6.2 Implementation details

CloudFormation is an AWS product for IaC (Infrastructure as Code). A file formatted with YAML syntax can be used to define what AWS resources are needed to deploy a service. For example, it can be specified an EC2 Instance (Virtual Machine), with specific features like memory, disc capacity or CPU.

In addition to AWS resources, it is necessary to specify the application to be deployed in the resources. There are mainly two possibilities: a) create the instance with an image that includes the software or b) create the instance with a plain operating system and provision the software when instance is started. The first option would allow a faster provisioning of the software artifacts since the software is pre-installed already whereas

the second option is more flexible because it allows to select the version of the software on deploy time. The consortium decided for the second option because while developing a highly complex software framework it is beneficial to 1) consider and using always the latest version of the artifacts including newest features and fixes and 2) to have another automated installation process for verification, testing and troubleshooting.

To provision the software there are also multiple possibilities. In ElasTest the tool used is Ansible[5]. Ansible is a product from Red Hat that automates software provisioning, configuration management, and application deployment. It's also based on YAML files, called playbooks, to define the tasks to perform on remote or local systems.

When the AWS EC2 instance is ready, Ansible will provisioning it turning the form input values into configured services. CloudFormation allows to execute scripts inside the instances.

The Ansible playbook used in ElasTest will perform the following tasks:

1. Installation of Docker (Docker is needed to execute ElasTest).
2. Configuration of the instance in order to execute correctly ElasticSearch[6] in EDM core component. ElasticSearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. The host system has to be configured to execute ElasticSearch container correctly.
3. Configuration of the system to automatically start ElasTest in every boot up using the start command available in platform container.
4. Creation of maintenance scripts to terminate ElasTest when necessary in order to not deal with duplicated containers that might cause malfunctioning.
5. Send a signal to CloudFormation API to let it know the work is done.

When the last task is executed, Cloud Formation will receive the signal to show in the output tab the URL that can be used to use ElasTest [3].

4 ElasTest integrations with external tools

ElasTest aims to be compatible with current SotA CI tools and methodologies so that developers can use it without disrupting their common practices. With this objective in mind we have integrated ElasTest with:

- **Jenkins:** Is the leading open source continuous integration tool. It can be extended with plugins to augment its features and integrate it with other tools.
- **TestLink:** Is the leading open source tool for test management, especially with manual testing.

In the following subsections, these two integrations will be described.

[5] https://www.ansible.com/
[6] https://www.elastic.co/products/elasticsearch

4.1 Jenkins integration

The ElasTest Jenkins Plugin (EJ) is a Jenkins plugin whose purpose is to integrate ElasTest with Jenkins. This plugin allows to use together Jenkins' capacity and experience to manage the continuous integration of projects and ElasTest's features for log analysis and additional capabilities provided by TSSs. Figure 9 shows ElasTest Jenkins plugin official web page[7].

This plugin allows to use ElasTest features from a Jenkins job. Specifically, it allows to send job logs to ElasTest and also let test code executed in the job to use ElasTest TSSs. For example, a test executed in a Jenkins job can use browsers provided by EUS TSS. Also, all the information gathered during test execution in Jenkins can be analyzed in ElasTest graphical user interface. These features can be used in Freestyle jobs and in jobs defined with the new Jenkins pipeline syntax.

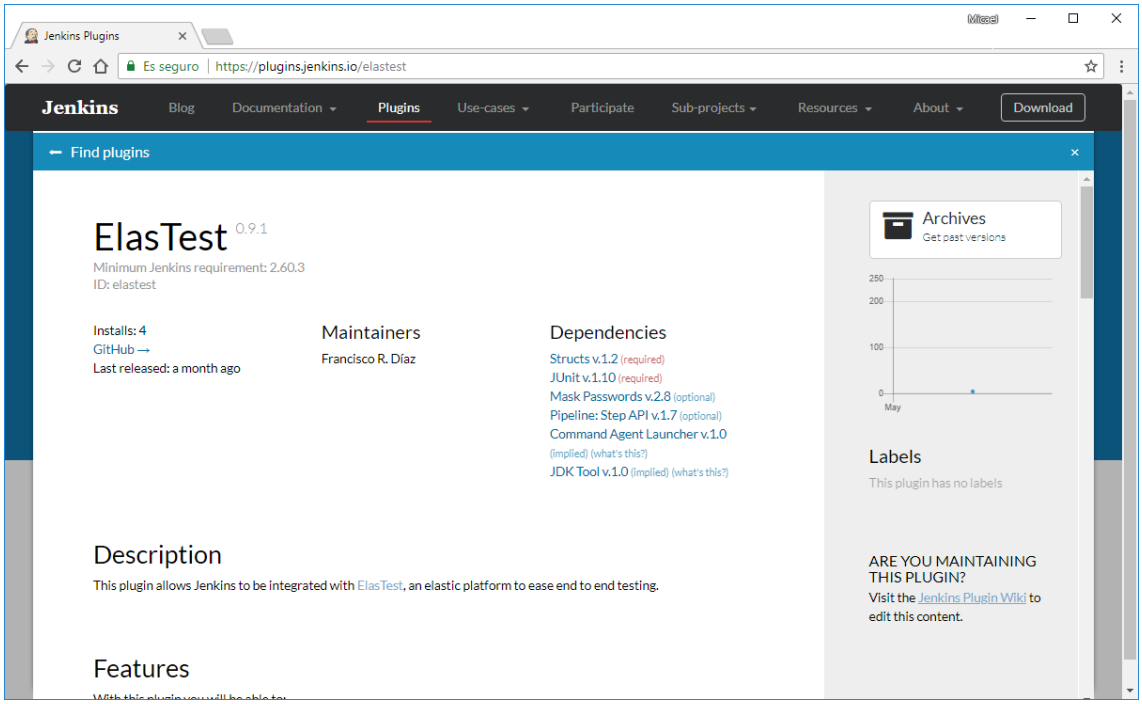

Figure 9. ElasTest Jenkins Plugin official web page

4.1.1 Baseline concepts and technologies

The EJ is developed using the framework for plugin development provided by Jenkins. This framework provides the necessary tools to extend the Jenkins functionality in a controlled and homogeneous way. It defines extensibility points (interfaces or abstract classes) to allow developers to extend or define new Jenkins functionality. It provides its own HTML rendering template called Jelly[8] and uses another framework called Stapler[9] to export plugin objects with a REST-like API.

[7] https://plugins.jenkins.io/elastest

[8] https://wiki.jenkins.io/display/JENKINS/Basic+guide+to+Jelly+usage+in+Jenkins

[9] http://stapler.kohsuke.org/

EJ communicates with ElasTest using the ElasTest Tests Manager (ETM) REST API. ETM core can manage completely the lifecycle of TJobs executed inside ElasTest. But it also allows external tools to manage part of this lifecycle. This feature is used by Jenkins plugin to send Jenkins' jobs log to ElasTest and provide to it TSSs.

4.1.2 Component Architecture

ElasTest Jenkins Plugin is composed by several modules. Figure 10 shows these modules and the relations between them. It also shows how some of the modules interact with ElasTest (by means of EMS's REST API).

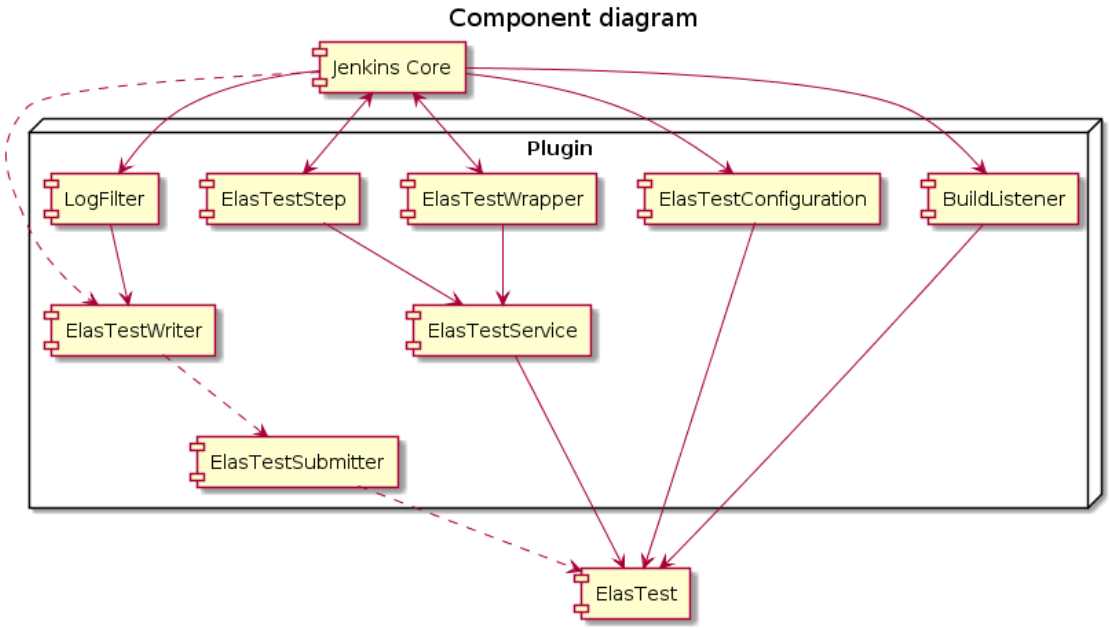

Figure 10. ElasTest Jenkins Plugin modules diagram

Jenkins Core

This module represents the Jenkins logic and its main concepts: jobs, builds, context execution, etc. This module is responsible to execute the actions provided by the plugin. When a freestyle job is used, ElasTestWrapper module is used. Otherwise, when pipeline job is used, the module is ElasTestStep. Job's log is sent to ElasTest using ElasTestWriter. Plugin configuration is managed with ElasTestConfiguration.

Plugin Modules

The main component modules are:

- **LogFilter:** Implementation of the *ConsoleLogFilter* class, an abstract class provided by Jenkins to allow log manipulation.
- **ElasTestWriter:** Used by Jenkins to send the log traces of a Job builds to ElasTest. Manages the sending cycle of a message, message composition and message delivery.

- **ElasTestSubmitter:** Client used by the ElasTestWriter to send a composed message to ElasTest. It sends messages to LogStash[10] interface provided by ElasTest.
- **ElasTestStep:** This component allows the plugin to be used from a Pipeline Job and generates a new step that you can use in a pipeline script (*elastest(){....}*). Prepares the build environment and integrates the Job execution with ElasTest.
- **ElasTestWrapper:** This component allows the plugin to be used from a Freestyle Job. From the configuration of a Job, in the *Build Environment* section you can select the *Integrate with ElasTest* option. It prepares the build environment and integrates the Job execution with ElasTest.
- **ElasTestConfiguration:** Entity that stores the plugin configuration and that provides the functionality to test the connection with ElasTest.
- **ElasTestService:** Interface with ElasTest REST API. It allows to create the external TJob execution in ElasTest, to check if the TJob execution is ready so the Job execution can continue and to send the Job execution results to ElasTest.
- **BuildListener:** Implementation of the *RunListener* class, an abstract class provided by Jenkins to receive notifications from every build that happens in Jenkins. When the build of a job is finished, it starts the process to obtain the test results and send them to ElasTest.

ElasTest

Represents the ElasTest Platform itself. Its API allows an external tool to create the necessary entities to integrate the execution of a job in Jenkins with the execution of a TJob in ElasTest.

4.1.3 Data Model

The data model managed by the EJ plugin is shown in Figure 11 and is split into two parts. On the one hand, the data managed and stored by Jenkins, such as the global configuration of the plugin and the configuration of the plugin in a Job, and on the other hand the data used to exchange information between the plugin and ElasTest.

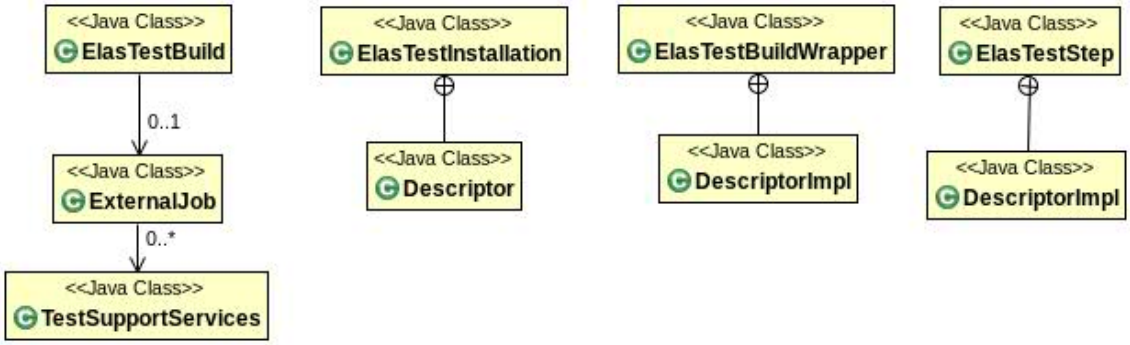

Figure 11. Data Model Diagram

Exchange Data: These classes stored the necessary info to integrate both Jobs, the Job on the Jenkins side and the TJob on the ElasTest side.

[10] https://www.elastic.co/products/logstash

- **ElasTestBuild:** Main entity that stores all the data related to a construction that the plugin requires for its proper operation. It contains the workspace path for the current build and an instance of the ExternalJob class.
- **ExternalJob:** Main entity of this model with the information exchanged with ElasTest.
- **TestSupportService:** Entity that stores the data related to each TSS requested to ElasTest to be used within the Jenkins job.

Configuration Data: Jenkins has its own way of persisting the information used by a plugin.

- **ElasTestInstallation:** This class contains global configuration of the plugin.
- **ElasTestBuildWrapper:** This class is used for freestyle jobs.
- **ElasTestStep:** This makes the same as the previous one, but for a pipeline job.

4.1.4 Use Cases

ElasTest Jenkins Plugin offers to main use cases to the user:

- Use case 1: Set up the plugin
- Use case 2: Build a Job that uses the plugin.

In the following subsections, these two uses cases will be described:

4.1.4.1 Set up the plugin use case

In this use case a user configures the plugin and test if that configuration is right (Figure 12. Plugin Configuration).

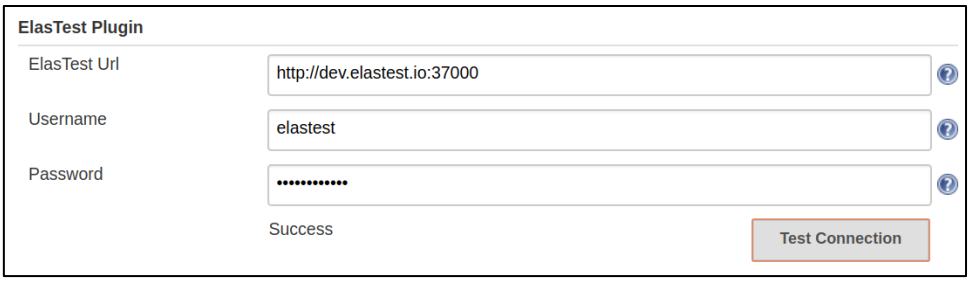

Figure 12. Plugin Configuration

Figure 13 shows an UML sequence diagram with the interaction between Jenkins and ElasTest.

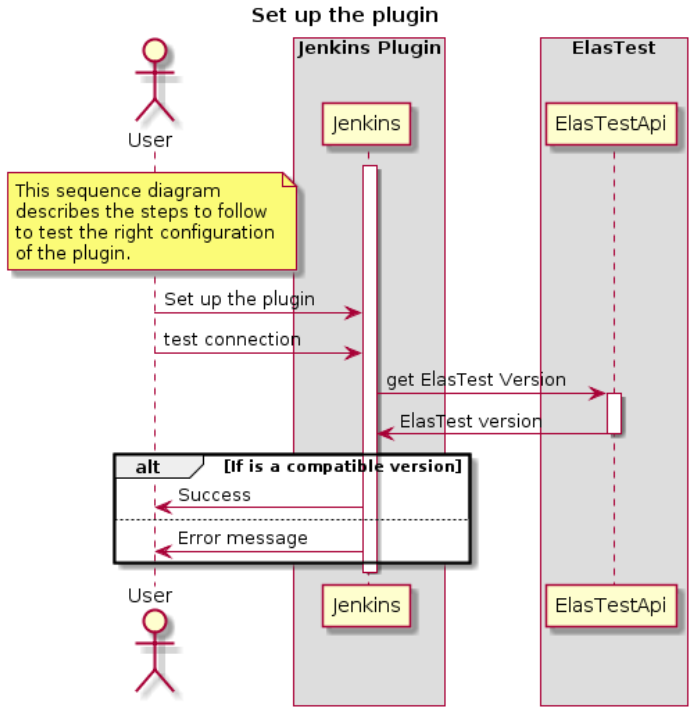

Figure 13. Jenkins and ElasTest interactions when plugin is configured

This diagram is expanded in Figure 14 to show how modules interact inside the plugin.

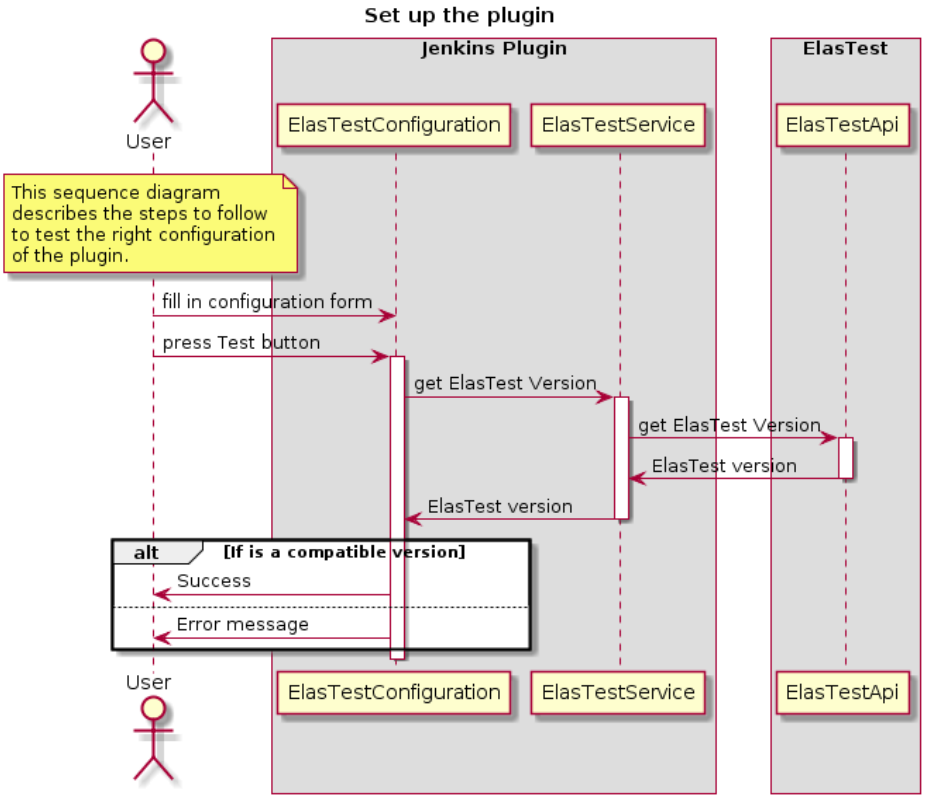

Figure 14. Plugin configuration

The components involved in this use case are ElasTestConfiguration component and the ElasTestService component. The first one is used to store the plugin configuration and contains the logic that allows to know if ElasTest version is compatible with the plugin version. The second one contains a client to access ElasTest REST API.

4.1.4.2 Build a Job that uses the plugin

The interaction between Jenkins plugin and ElasTest when a job is executed is shown in the UML sequence diagram of Figure 15.

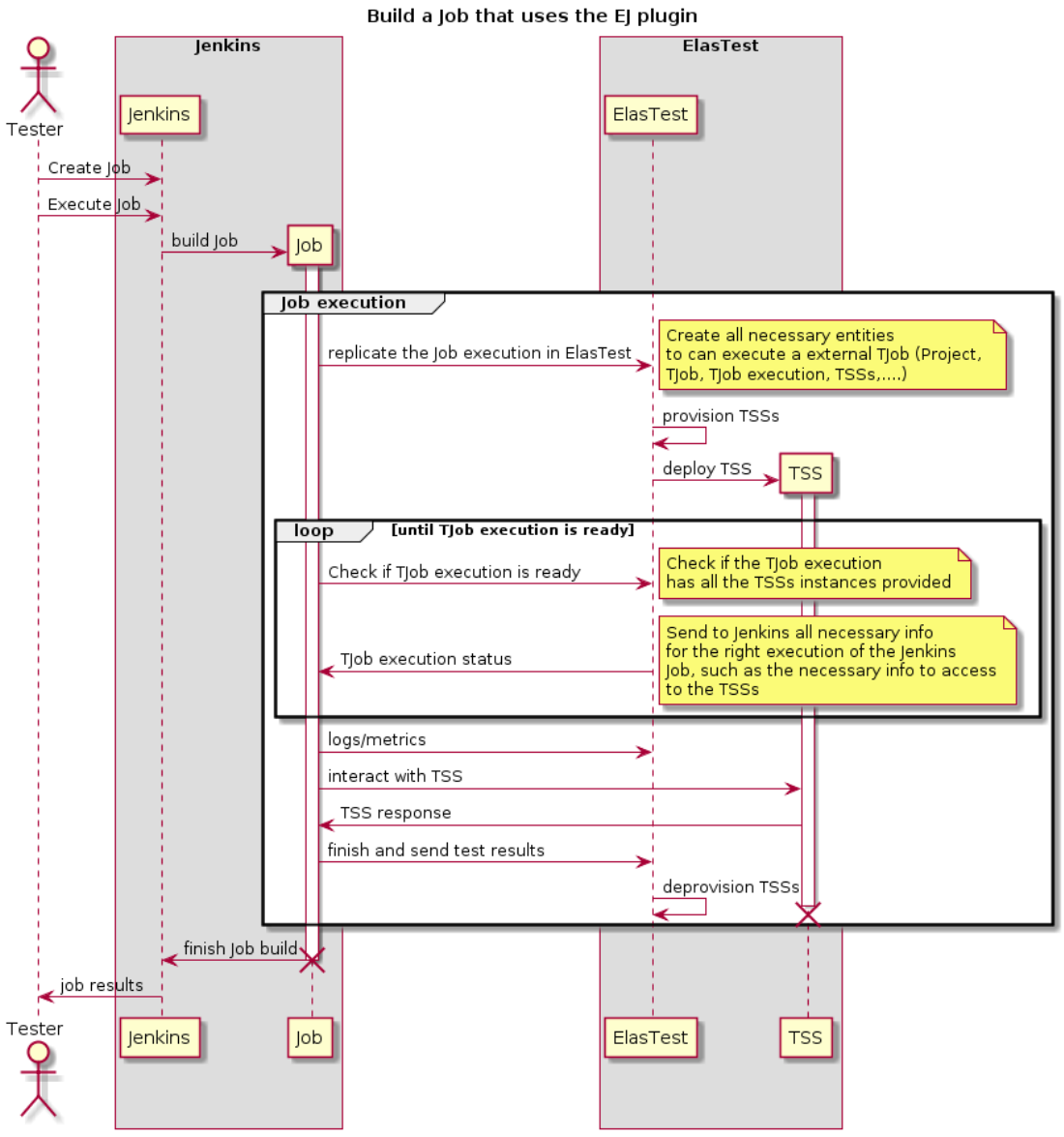

Figure 15. Jenkins job build using the plugin

The actions performed by the modules of the plugin depends on the specific plugin configuration of the Job. By default, if the plugin is configured in a job then the console logs are produced when the job is executed, then sent to ElasTest. If a TSS is selected to

be used in the tests executed, then the plugin asks for selected TSSs to ElasTest as shown in the sequence diagram.

The specific actions performed and the interactions between internal plugin modules depends on the job type. They are very different if the job is a freestyle job or a pipeline job.

ElasTest plugin in a pipeline job

Figure 16 shows how to use ElasTest Jenkins plugin in a pipeline job. In it, EUS service is requested and surefireReportsPattern is set to send to ElasTest test results when job is executed. In Figure 17 can be seen how *ElasTestStep* is the first plugin module be invoked. It creates the TJob execution in ElasTest and associates it with the actual Jenkins job build. Also, is the responsible to set up the build context with the environment variables needed to allow test code to know the URL of requested TSSs. Finally, *LogFilter* is invoked to create *ElasTestWriter* and *ElasTestSubmitter* modules to be used to process the build logs. During the build, Jenkins sends the log traces to ElasTest and, when the build finishes, Jenkins informs the *BuildListener* to send build results to ElasTest.

Figure 16 . Using the plugin in a pipeline job

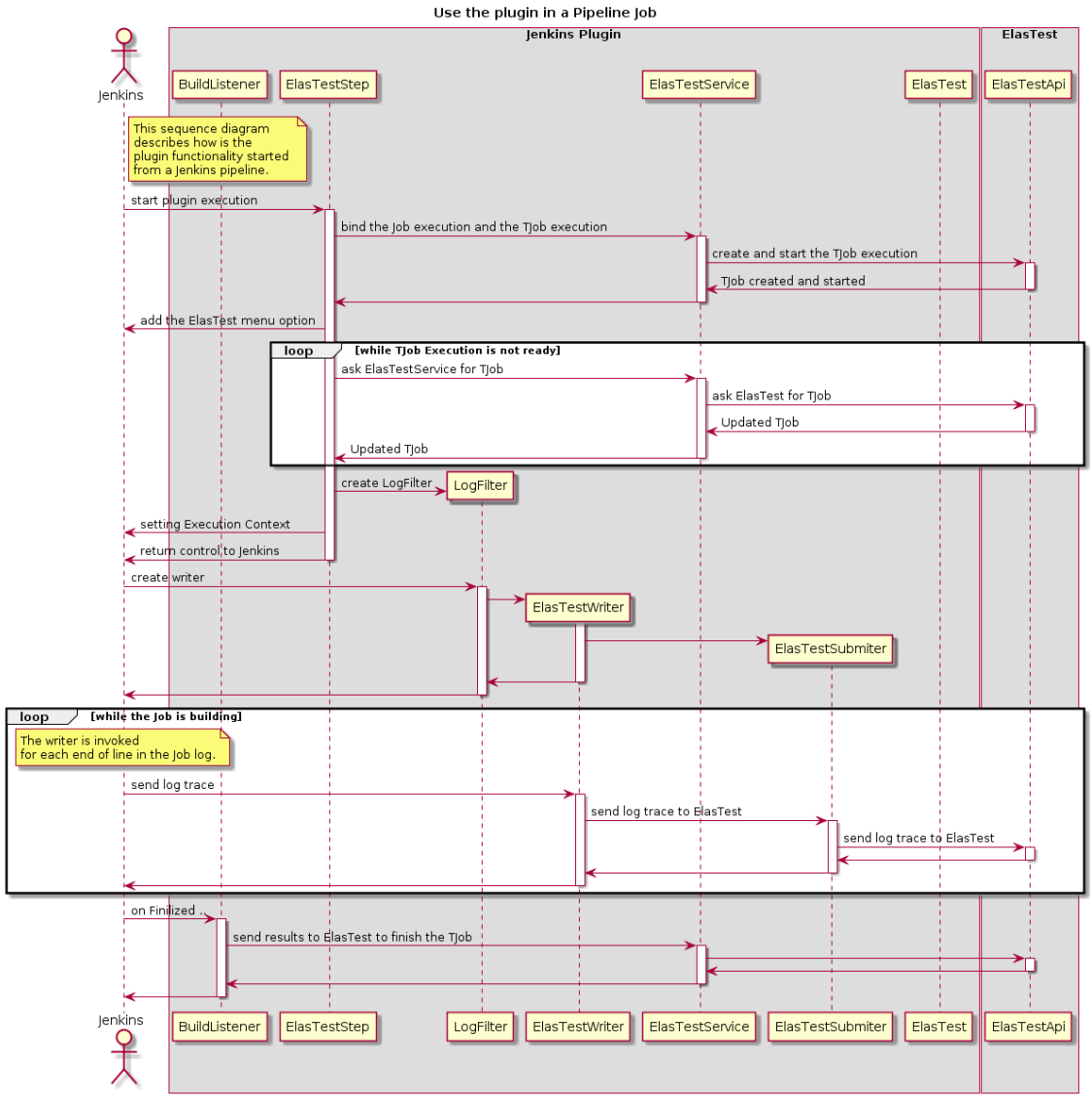

Figure 17. Plugin behavior in a pipeline job

ElasTest plugin in a freestyle job

When the plugin is used in a freestyle job (Figure 18), *ElasTestWrapper* module is the main plugin module. The behavior is different in pipeline jobs (described before) and in freestyle jobs (as shown in Figure 19). In this case, when Jenkins invokes the *ElasTestWrapper*, the first thing to do is to create a TJob execution in ElasTest and associate it with the actual build of the mmmsdfsdfdddfJob. Then, *LogFilter* module is initialized so that it can be invoked from Jenkins. After that, Jenkins invokes the *LogFilter* to create the *ElasTestWriter* and the *ElasTestSubmitter* to use them later to process the logs. In the next step, *ElasTestWrapper* is invoked again to wait for the TJob execution to be ready in the ElasTest side. With the data returned by ElasTest (in the updated TJob) the plugin updates the build execution context in Jenkins. During the build, Jenkins sends the log traces to ElasTest and, when the build finishes, Jenkins informs the *BuildListener* and it sends the build result to ElasTest (including the test reports if the Junit plugin is used in the job).

Figure 18. Plugin configuration in a Freestyle Job

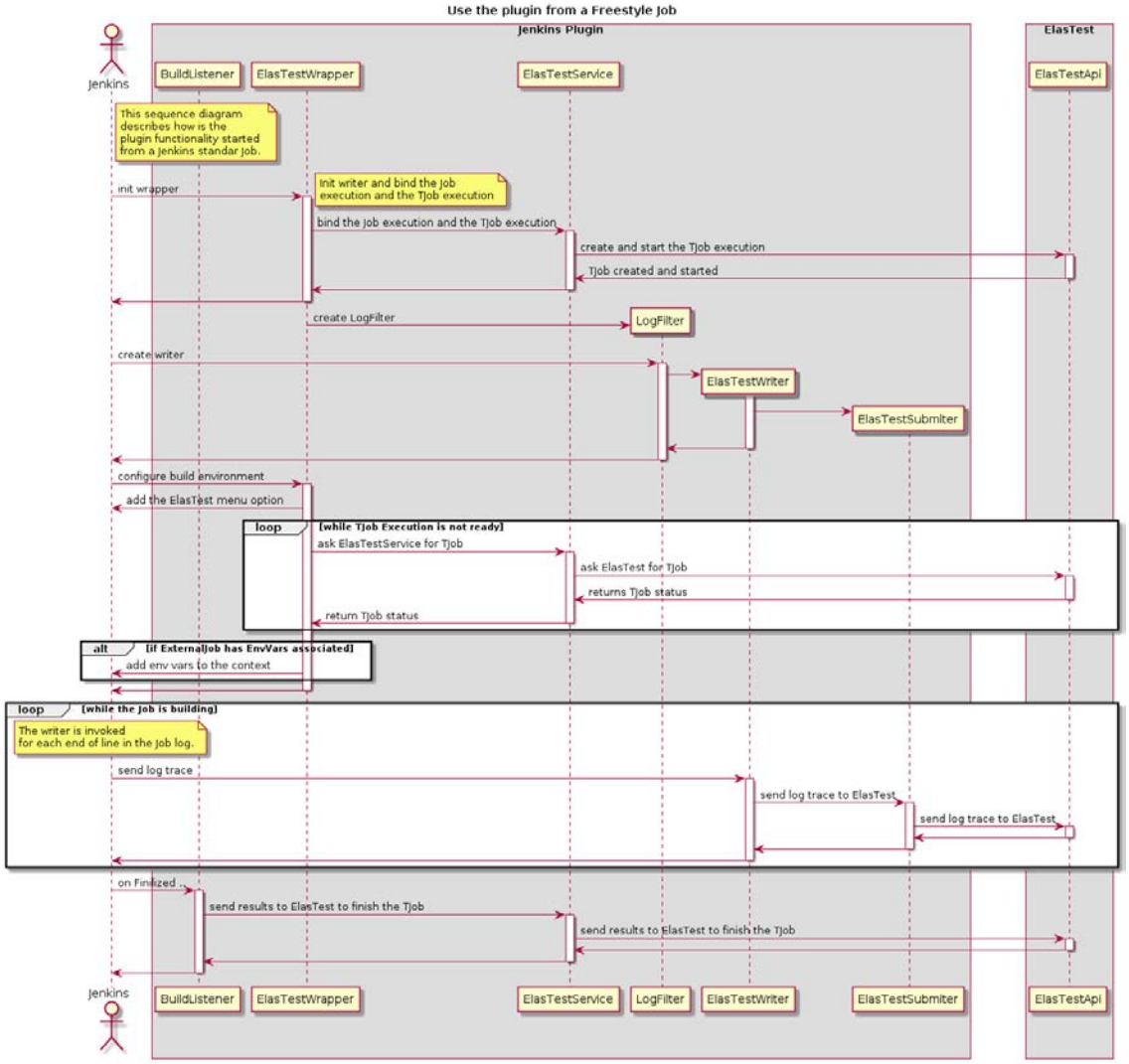

Figure 19. Plugin behavior in a Freestyle Job

4.2 TestLink Integration

TestLink is one of the most popular open source tools to define test plans to be performed manually during a QA test process. It allows to define test cases, test plans, builds, etc. A test case in TestLink consists of several steps. Every step contains the exact actions that should be performed in this step and the expected outcome of this actions.

To verify if the SUT behaves as expected, a tester has to “execute” manually the steps defined in every selected test case and verify if the obtained results are the expected ones. If this is the case, then the test case execution is marked as PASSED. However, if results are different than expected, the test execution is marked as FAILED. Moreover, the current behavior needs to be annotated in the test case execution or in some ticketing system to be managed by the development team. Usually, the not expected behavior is caused by a bug that needs to be fixed. The process to register the current behavior typically involves executing again the test case to take screenshots of the steps, log into remote systems where SUT is deployed to gather logs, configuration files, etc.

Gathering all evidences and describing current behavior can be time consuming. The integration between ElasTest and TestLink allows the tester to perform the manual actions defined in test case steps in a browser provided by ElasTest. In this way, when a tester marks a test case execution as FAILED, a recording of the interactions with the browser attached is provided to the test case execution, the browser console is also attached. Moreover, if a SUT properly configured is being tested in ElasTest, logs and metrics of the different SUT components are also registered. Using ElasTest integration with TestLink, tester doesn’t have to register the actual behavior manually because it is done automatically. The developers will have all the powerful ElasTest tools to analyze the information associated to the failed test case like Log Analyzer.

The integration between ElasTest and TestLink is implemented using the TestLink REST API to import test definitions into ElasTest. When tests’ information is imported into ElasTest, tester can execute TestLink test plans using ElasTest graphical interface (see Figure 20). Test execution results are written back to TestLink. The ElasTest URL for that specific test case execution is added to the comments section of test case execution. In that way, when a developer sees the bug report associated to this failed execution, he can analyze all the gathered information using ElasTest tools.

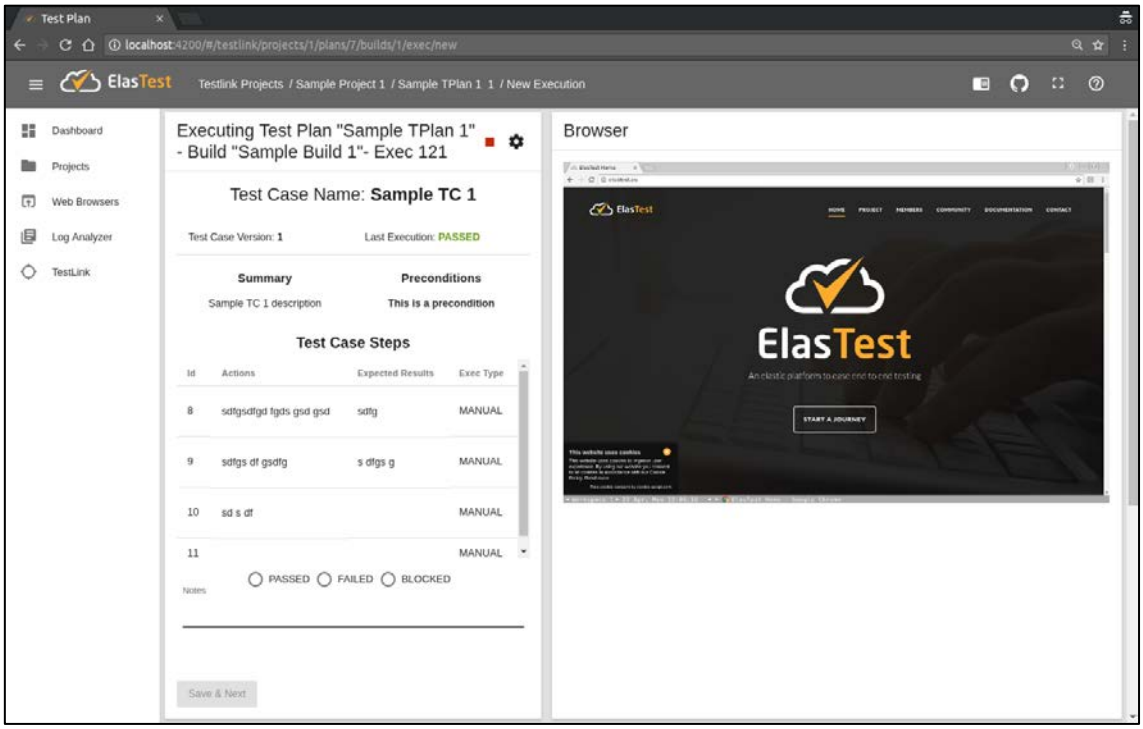

Figure 20. Running a TestLink test plan within ElasTest

4.2.1 Baseline concepts and technologies

TestLink provides a REST API[11] that can be used to perform the same actions that can be performed using the web interface. For example, creating a test case, define the steps, execute a test plan, etc. This API is used by ElasTest to import TestLink information to its own database. Then, when a test plan is executed using ElasTest tools, the result is saved in TestLink database by means of TestLink REST API.

4.2.2 Component Architecture

The interaction between the high-level modules involved in the TestLink integration is shown in the Figure 21. The integration is designed to allow the user interacting with TestLink directly and through ElasTest main web interface.

[11] https://metacpan.org/pod/TestLink::API

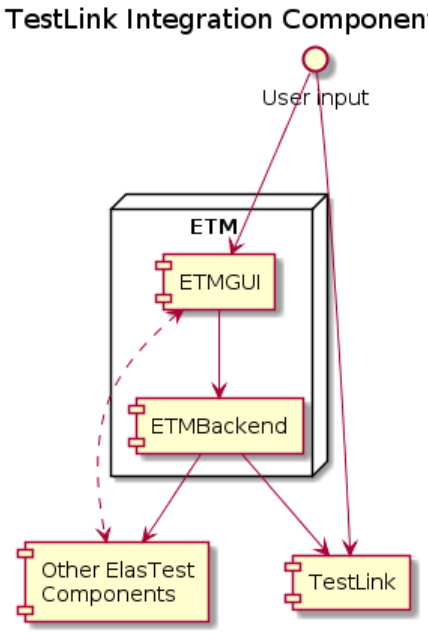

Figure 21. Module diagram of the integration between ElasTest and TestLink

ElasTest Tests Manager (ETM) is the brain of ElasTest and the main entry point for developers. ETM is implemented with a Single Page Application (SPA) architecture. ETMGUI is the frontend part implemented with Angular[12] and ETMBackend is the backend part implemented with Java and SpringBoot[13]. TestLink integration is mainly implemented in ETMGUI and ETMBackend, but other ElasTest components are used. For example, EUS Test Support Service is used to provide the browsers used to perform the tests.

TestLink itself can be started as an ElasTest component if required. This simplifies the setup and configuration of the interaction between services. Figure 22 shows the TestLink web interface. TestLink is started within ElasTest by default in *Experimental* mode, whereas for *Normal* or *Experimental Lite* mode it is needed to specify it explicitly in the command line when ElasTest is started.

[12] https://angular.io/

[13] https://spring.io/projects/spring-boot

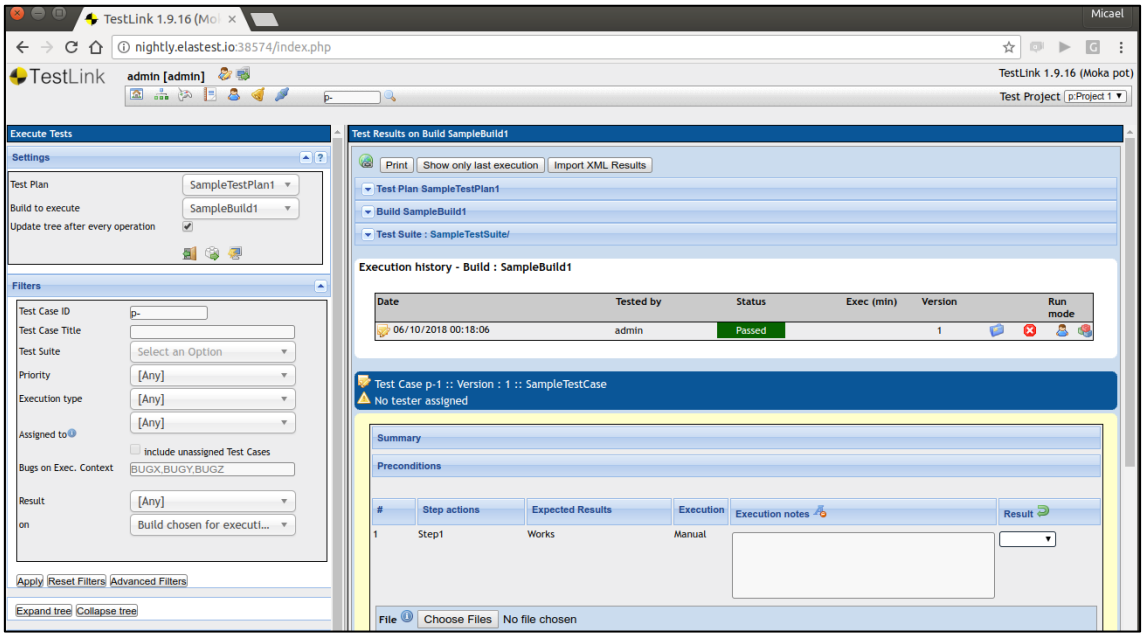

Figure 22. TestLink screenshot

4.2.3 Data Model

The data model of this integration between ElasTest and TestLink is shown in Figure 23. The Data model is very similar to the model used to store the information about TJobs executed by ElasTest, but in this case have different properties to maintain the relation with the external entities stored in TestLink database.

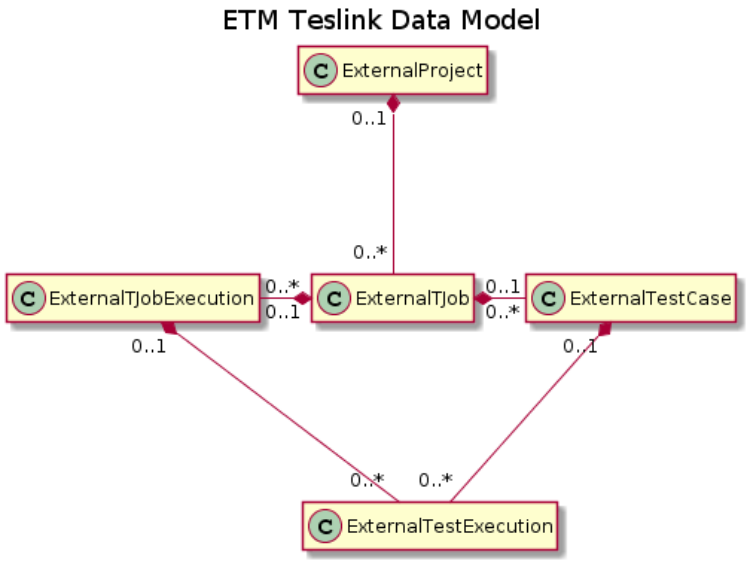

Figure 23. ETM TestLink Data Model

These entities are used to store the following information:

- **ExternalProject:** is the basic organizational unit, which groups a series of ExternalTJob under itself. It's related with one TestLink Test Project.

- **ExternalTJob:** is the basic unit for executing a set of tests on an external application. It can have a series of test cases associated to execute it. It is related with one TestLink Test Plan.
- **ExternalTestCase:** is a set of conditions or variables that make up the test of a simple case. It's related with one TestLink Test Case.
- **ExternalTestExecution:** is the execution of an ExternalTestCase that contains information about it. For example, information like test result (FAILED or SUCCED) and start/end date are stored. It is linked to one TestLink Test Execution.
- **ExternalTJobExecution:** is the execution of an ExternalTJob that contains information about it. When an ExternalTJob is executed, one execution is created for each one of its associated ExternalTestCases and its ExternalTestExecutions are grouped in ExternalTJobExecution. It is not linked to any TestLink data model.

4.2.4 Use Cases

When a tester wants to execute in ElasTest some tests defined in TestLink, the following steps are needed:

1. Create the Test Project, Test Plan, Test cases, and the rest of the required entities in TestLink.
2. Import TestLink tests to ElasTest.
3. Execute a TestLink Test Plan within ElasTest interface.

The first one of these steps is out of the scope of this documentation because it the usual way to work with TestLink. The other two steps are described in the following paragraphs.

Import TestLink tests to ElasTest

Before executing a TestLink Test Plan in ElasTest it is necessary to import it to create in ElasTest the entities defined in the data model. To do this, the synchronization button that appears at the top right of the main TestLink page in ElasTest have to be used (Figure 24).

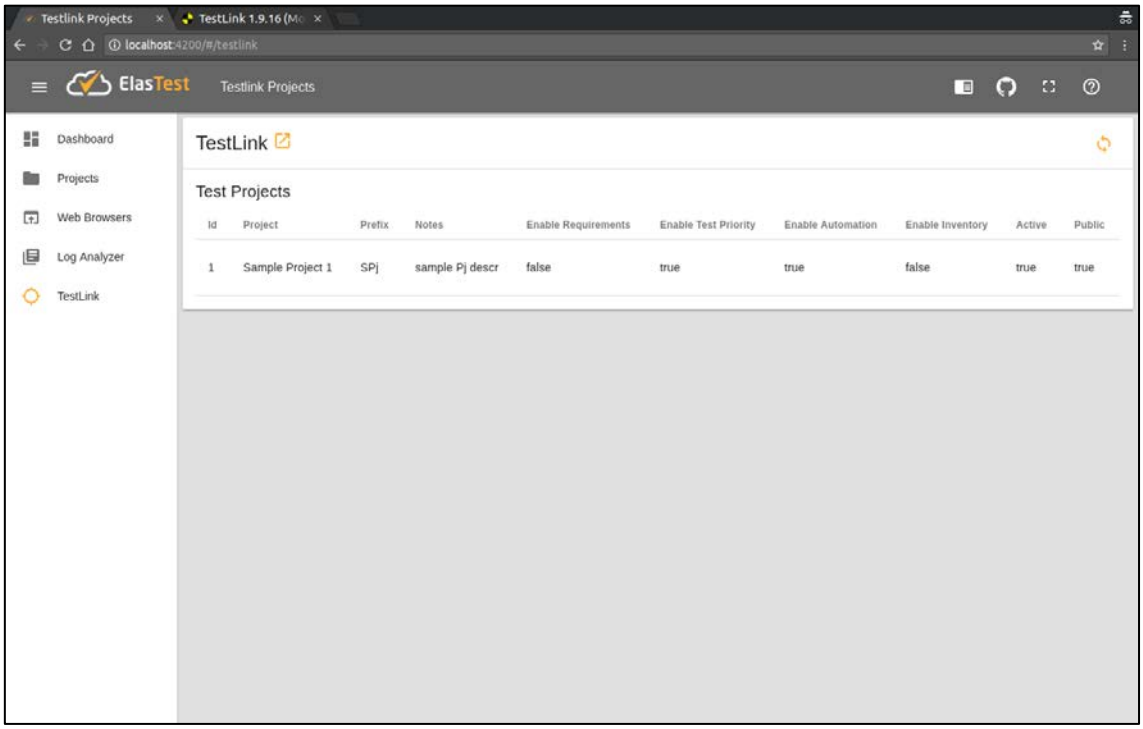

Figure 24. Main page of the TestLink interface in ElasTest

The internal actions performed when sync button is pressed are shown in Figure 25.

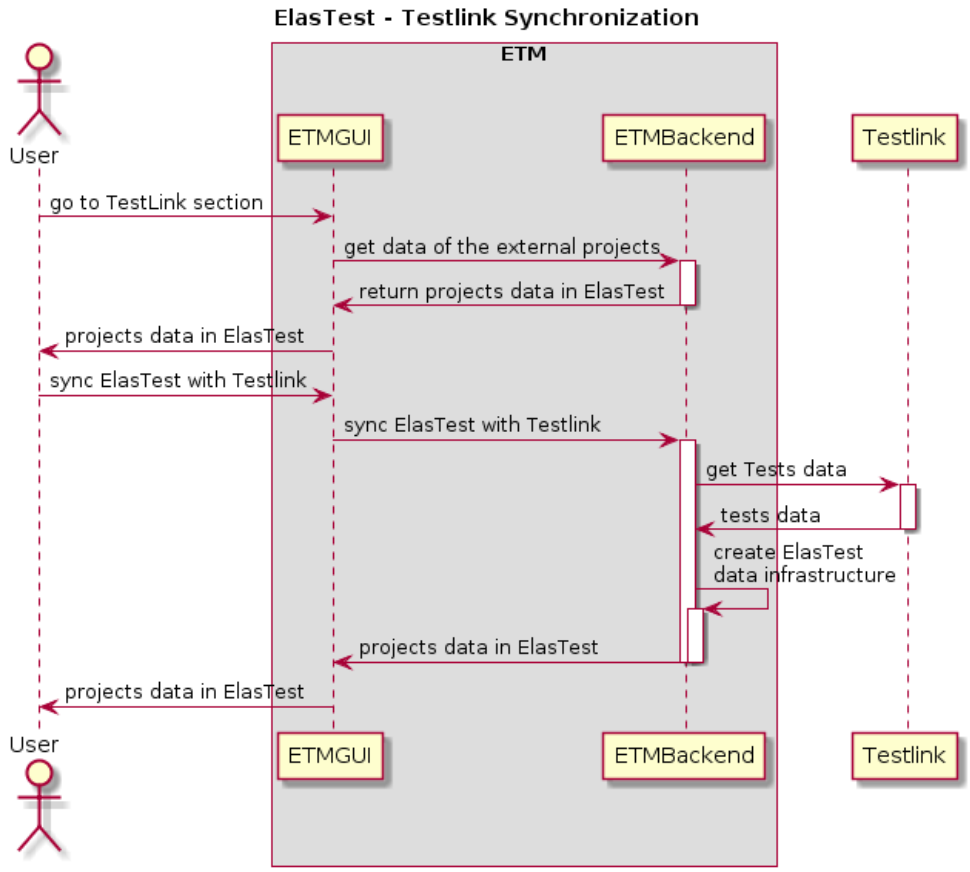

Figure 25. Synchronization between ElasTest and TestLink

Run a TestLink Test Plan from ElasTest

When TestLink information are synchronized with ElasTest entities, a TestLink Test Plan can be executed in ElasTest. In that way, all evidences gathered during the execution are associated to the execution, making easier to fix bugs when obtained results are not similar to the expected ones. To execute a Test Plan, the user has to navigate to the screen of the Test Plan and click on Play button. The internal interactions performed in this case are shown in Figure 26.

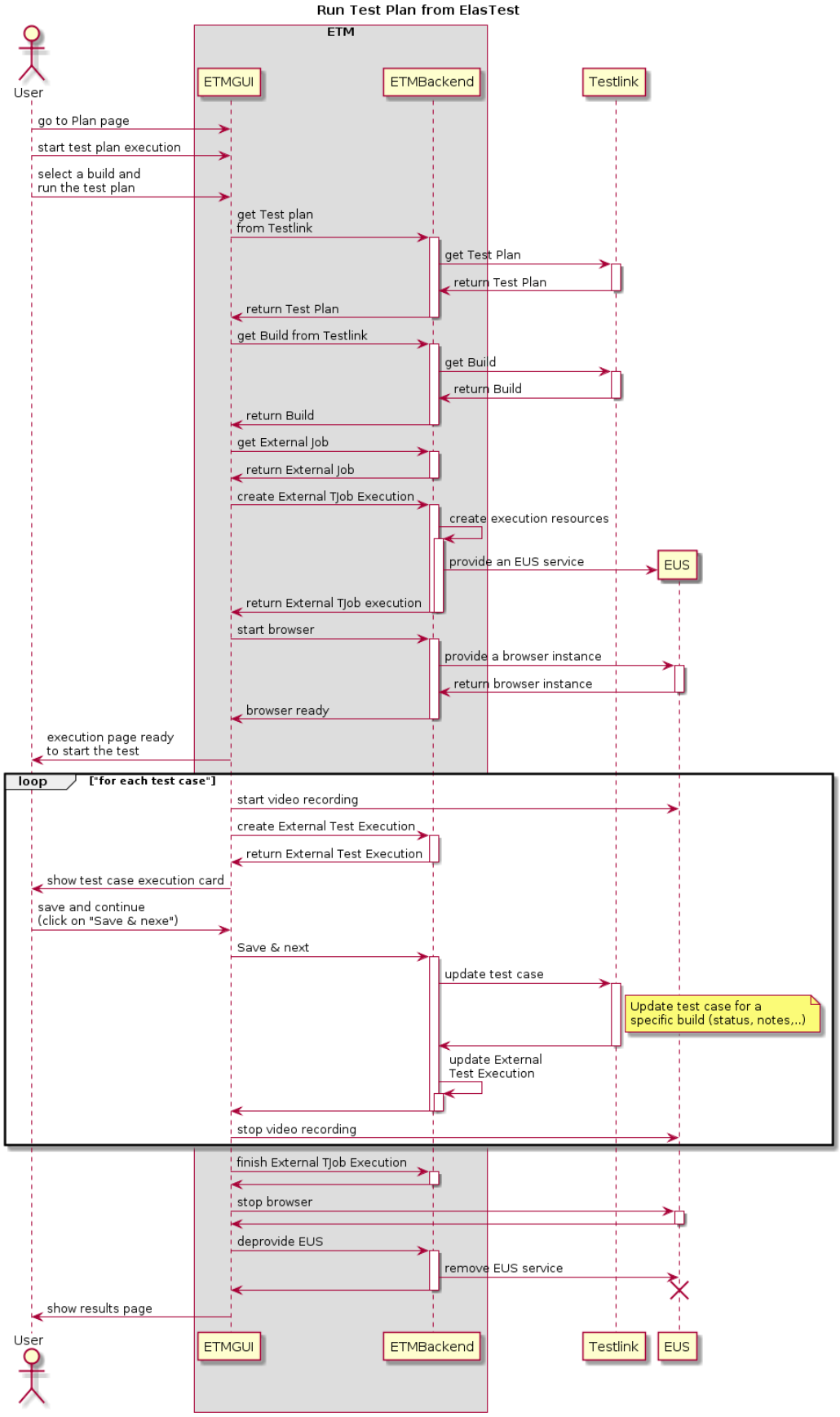

Figure 26. Execute a TestLink Test Plan in ElasTest

5 Conclusions and future work

ElasTest is a powerful tool that provides interesting features to testers and developers. But a tool difficult to install and difficult to adopt because requires to disrupt the existing practices of a team is a tool doomed to failure. To improve the adoption of a tool it must be easy to install and easy to adopt. For that reason, project members have put a lot of effort to simplify the ElasTest installation and the usage with popular open source tools. In particular, ElasTest can be installed and executed with only a command in a system with Docker tools. Also, the ElasTest CloudFormation template can be used to deploy ElasTest in an EC2 AWS instance with a few clicks. Regarding to integration with external tools, ElasTest main features can be used very easily from Jenkins, the most used open source continuous integration tool. In that way, it is very easy to adopt ElasTest if Jenkins is used. Only a few new lines in a pipeline job and a few clicks in a Freestyle Job. Lastly, if a team is using TestLink, ElasTest can be used to execute its test cases. Therefore, can be concluded that objectives of facility of installation and integration with other tools are met.

In the future, it is planned to improve existing installation options and add new supported platforms. For example, deploying ElasTest in a Kubernetes cluster using Helm or deploy in a CloudFormation stack are two options being considered. Regarding to integration with existing tools, Jenkins plugin can be improved in several ways. For example, in the current version, the user is responsible to instrument tests and SUT to send metrics to ElasTest. In the future versions, it is planned to automatically instrument them if requested by the user. TestLink integration can also be improved, for example, allowing the user to select the browser version and vendor, or improving the copy and paste behavior.

As a general conclusion, ElasTest has a Toolbox to install it easily in different platforms and can be used integrated with Jenkins and TestLink. However, these tools can be improved in different ways to improve user experience.

6 References

[1] Java Tools and Technologies Landscape 2016: https://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016/

[2] Docker containers. https://www.docker.com/what-container

[3] ElasTest AWS Stack JSON file. https://raw.githubusercontent.com/elastest/elastest-toolbox/master/AWS/cloud-formation-latest.json

[4] D2.2: SotA revision document v1