D6.4	
Version	1.0
Author	URJC
Dissemination	PU
Date	30-12-2019
Status	FINAL

D6.4 ElasTest platform toolbox and integrations v2

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	20-12-2019
Actual delivery date	30-12-2019
Keywords	Open source software, cloud computing, software engineering, operating systems, computer languages, software design & development

Funded by the European Union

Contributors

Name	Affiliation
Francisco Gortázar	URJC
Francisco Ramón Díaz	URJC
Eduardo Jiménez	URJC
Micael Gallego	URJC

Version history

Version	Date	Author(s)	Description of changes
0.1	03/12/2019	Francisco Gortázar	Initial version (DRAFT)
0.2	05/12/2019	Francisco Ramón Díaz	Description of ElasTest modes
0.3	10/12/2019	Eduardo Jiménez	Description of command line options and component diagrams. Updates on TestLink and Jenkins.
0.4	12/12/2019	Francisco Gortázar	Review of sections 3 and 4. Writing sections 1, 2, 5
1.0	30/12/2019	Micael Gallego	Final Review

Table of contents

List of figures

List of tables

No tables in this document

Glossary of acronyms

Acronym	Definition
API	Application Programming Interface
AWS	Amazon Web Services
CI	Continuous Integration
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
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
K8s	Kubernetes
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 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 several cloud provides (Openstack and Amazon AWS), on a machine with Docker engine available and on a Kubernetes cluster on top of any 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 describes how ElasTest Toolbox is able to install and execute ElasTest in a system with Docker and how to deploy it 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[2] containers. Docker provides a clean and simple way to package software and is a widely accepted distribution platform for open source software artefacts. As a result of this, the standard FOSS repository to publish all ElasTest binary artefacts is DockerHub. Nevertheless, since DockerHub is a marketplace for providing images only, the orchestration of all the containers must be managed separately. It is needed to download required images and start them in the correct order by taking care of the dependency resolution among all the components. To achieve this task, in the first period of the project (from M1 to M18) URJC team developed a component called "ElasTest Platform". The ElasTest Platform component is also distributed and executed as a Docker container and needs Docker engine installed to be used. To make the installation of ElasTest in a cloud provider's server even easier, URJC team also provided a CloudFormation description file to deploy ElasTest in AWS with very simple steps.

In this second period of the project (from M19 to M36) URJC team has provided deployment descriptors for Kubernetes, that allows users to deploy ElasTest on top of a Kubernetes cluster. Also, ElasTest Platform has been extended to be able to deploy a Kubernetes cluster if no cluster available, so that users do not need to understand how this Kubernetes deployment is done. The different deployment modes of ElasTest 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.*

[1] Some components are distributed with other formats, like ElasTest Jenkins plugin.

[2] https://www.docker.com/

During the first period we decided to provide integrations with Jenkins, as a common CI server in the industry and TestLink, based on our scoping of the industry. However, the Jenkins integration still required many manual interventions by ElasTest users, which could stop it from being adopted. During the second period of the project, URJC team has worked towards a seamless integration of ElasTest with state-of-the-art practices in Jenkins for an effortless integration of both tools. In addition, several improvements have been made to the TestLink integration, mostly in support of the ATOS vertical and its validation experiments within the project as part of Work Package 7, with the supervision of CNR. Section 4 contains detailed descriptions of the improvements done within these two integrations.

3 ElasTest Toolbox

Deliverable 6.2 describes ElasTest platform toolbox and integrations for the first period of the project (from M1 to M18) and can be considered as a previous version of this document. In it, ElasTest Toolbox is the generic name used as an umbrella for all the tools, artefacts and procedures designed to facilitate the installation of ElasTest in different environments. This section will describe the new forms added in the second period of the project to deploy ElasTest:

- Deploy ElasTest in Mini mode
- Deploy ElasTest in Singlenode mode
- Deploy ElasTest on K8s manually, using ElasTest manifests
- Deploy ElasTest on K8s using the Toolbox and the EPM

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

3.1 System Requirements

ElasTest is a distributed application aimed at testing and gathering data from other distributed applications (the system under test, SUT). It is expected that ElasTest will ingest a huge amount of data while tests are run, and the services ElasTest provides for consuming this data require a lot of resources. The following is a list describing the resources needed for each ElasTest deployment. These shall be considered as a bare minimum, and more resources might be needed depending on the use case:

- ElasTest Mini: this version was built as a reduced version requiring less resources. 2 CPUs and 8Gb are the minimum requirements needed for this deployment mode.
- ElasTest Singlenode: 16Gb and 2 CPUs
- ElasTest EK (ElasTest mini on Kubernetes): 16Gb and 2 CPUs
- ElasTest HEK (Highly elastic ElasTest on Kubernetes): At least a Kubernetes cluster with two worker nodes with 8Gb and 2CPUs.

3.2 Technologies used for ElasTest distribution

ElasTest continues to be based on Docker containers, but now its deployment can be managed with Kubernetes. Kubernetes is an open-source container-orchestration system for automating application deployment, scaling, and management. It is considered the de-facto standard for deploying containerized applications in production, and is supported for many companies through the Cloud Native Computing Foundation (CNCF). This foundation comprises companies like Amazon, Google, Microsoft, Oracle, RedHat, Pivotal, among others.

3.3 Execution modes

Currently ElasTest can be executed in one of the following modes:

- **ElasTest Mini:** Currently this mode is the lightest of the four and it is the default execution mode. In this mode, Test Engines (TE) cannot be used. It is ideal to try ElasTest to test the core features and test web applications. In this mode, ETM acts as a substitute for Logstash, Elasticsearch, RabbitMQ and ESM.
- **ElasTest Singlenode:** In this mode ElasTest is executed with all the components. It is the mode recommended for production use in a single virtual or physical machine. It needs a machine with high computing resources. This mode can take several minutes to start all services provided by ElasTest.
- **EK (ElasTest Kubernetes):** In this mode, the same components available in ElasTest Mini mode are deployed in a Kubernetes cluster.
- **HEK (Highly Scalable ElasTest Kubernetes):** In this mode, the same components available in ElasTest Singlenode mode are deployed in a Kubernetes cluster. Moreover, EDM and EMP are deployed in an elastic way (EMP agents deployed through all the cluster to gather data from all the nodes)

3.4 ElasTest Platform

The ElasTest platform offers an easy way to manage the installation of ElasTest. To deploy ElasTest in a single virtual or physical machine the `--mode` parameter has to be used with values `mini` (default) or `singlenode` values to start with the respective modes. It uses docker-compose[3] under the hood to pull images and manage the execution of related docker containers.

ElasTest Platform can create a new VM in a cloud platform to deploy ElasTest. Currently, OpenStack and AWS are supported. To configure VM creation the following configuration properties have to be used:

- **--paas-type**: to indicate if it uses OpenStack (default) or AWS
- **--paas-url**: to indicate url when --pass-type is openstack
- **--paas-user**: username for OpenStack, access-key for AWS
- **--paas-pass**: password for OpenStack, secret-access-key for AWS
- **--paas-project-name**: project_name for OpenStack, region for AWS

[3] https://docs.docker.com/compose/

To deploy ElasTest in kubernetes (EK or HEK modes) the parameter `--kubernetes` have to be used.

3.5 Architecture

This section describes how ElasTest Platform is architected and how its modules interact. ElasTest Platform is implemented as a container packaging several Python script files. The most important ones are the following:

- **main:** as the name suggests, is the main script 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.
- **k8sDeployment:** is used to communicate ElasTest Platform with the EPM to deploy instances in OpenStack or AWS and start ElasTest on them.

Figure 1 shows the components and their interactions.

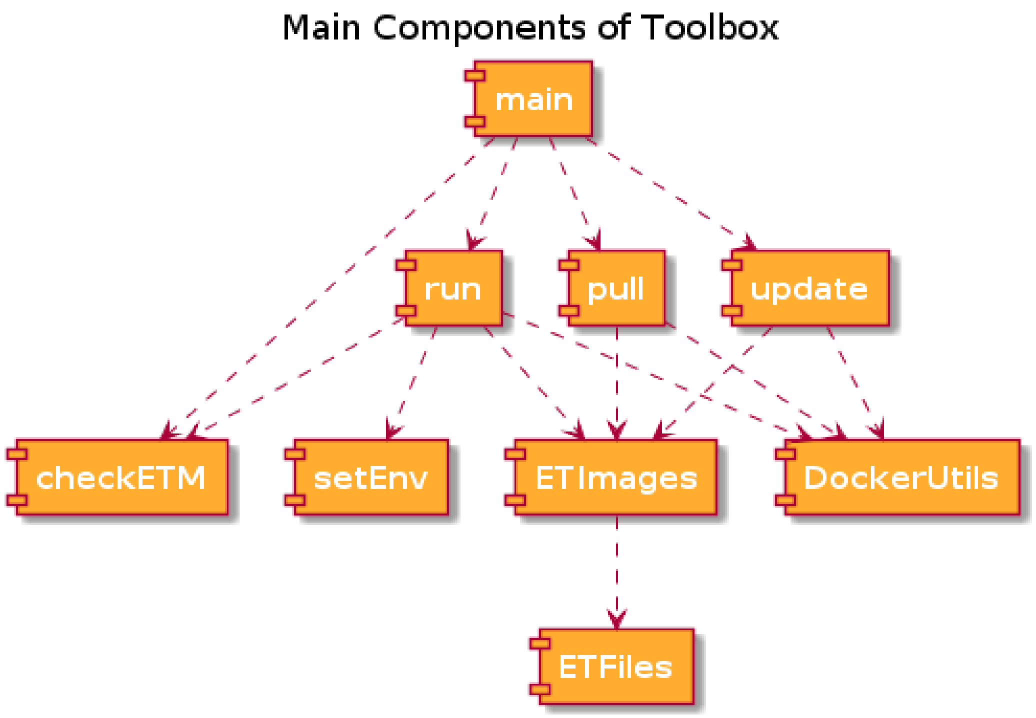


Figure 1. Toolbox Component Diagram

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

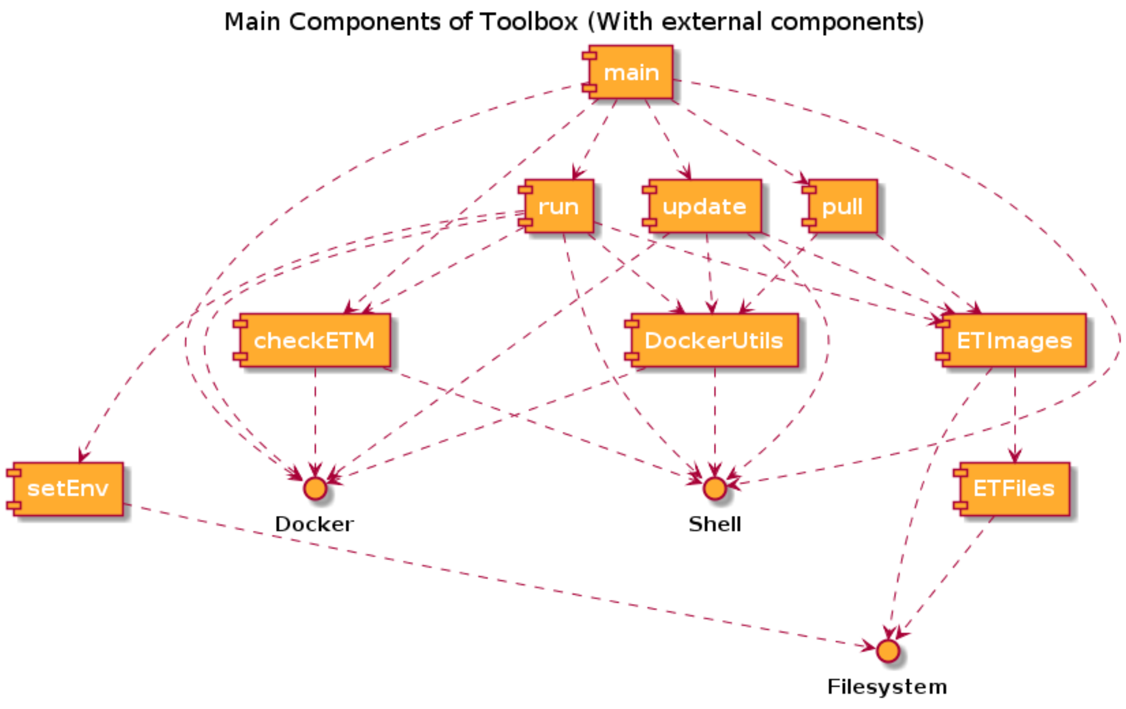


Figure 2. Toolbox Component Diagram including external dependencies

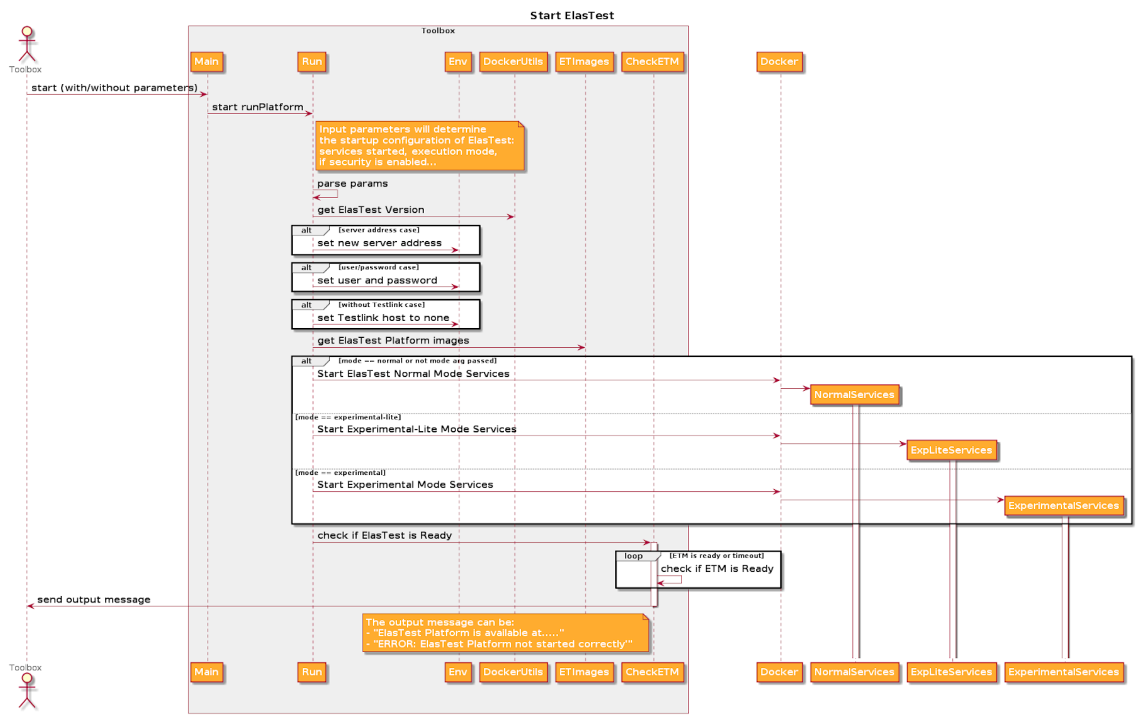

Start ElasTest
Toolbox
Toolbox
Main
Run
Env
DockerUtils
ETImages
CheckETM
Docker
start (with/without parameters)
start runPlatform
Input parameters will determine
the startup configuration of ElasTest:
services started, execution mode,
if security is enabled...
parse params
get ElasTest Version
alt [server address case]
set new server address
alt [user/password case]
set user and password
alt [without Testlink case]
set Testlink host to none
get ElasTest Platform images
alt [mode == normal or not mode arg passed]
Start ElasTest Normal Mode Services
NormalServices
[mode == experimental-lite]
Start Experimental-Lite Mode Services
ExpLiteServices
[mode == experimental]
Start Experimental Mode Services
ExperimentalServices
check if ElasTest is Ready
loop [ETM is ready or timeout]
check if ETM is Ready
send output message
The output message can be:
- "ElasTest Platform is available at....."
- "ERROR: ElasTest Platform not started correctly'"
Toolbox
Main
Run
Env
DockerUtils
ETImages
CheckETM
Docker
NormalServices
ExpLiteServices
ExperimentalServices

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.

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:

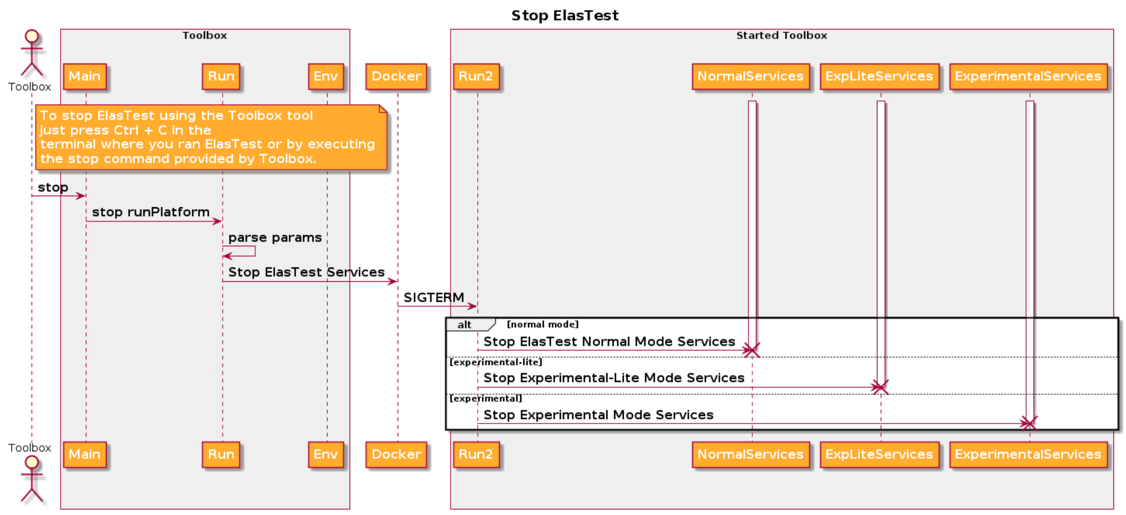

Stop ElasTest
Toolbox
Started Toolbox
Toolbox
Main
Run
Env
Docker
Run2
NormalServices
ExpLiteServices
ExperimentalServices
To stop ElasTest using the Toolbox tool just press Ctrl + C in the terminal where you ran ElasTest or by executing the stop command provided by Toolbox.
stop
stop runPlatform
parse params
Stop ElasTest Services
SIGTERM
alt
[normal mode]
Stop ElasTest Normal Mode Services
[experimental-lite]
Stop Experimental-Lite Mode Services
[experimental]
Stop Experimental Mode Services

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.

3.6 ElasTest on Kubernetes

As stated above, ElasTest can be deployed in Kubernetes in two different modes: **EK** and **HEK**. These modes can be deployed in an already existing Kubernetes cluster using the kubernetes manifests found in ElasTest Toolbox GitHub repository. The manifests of the EK mode are located in elastest-toolbox/kubernetes/ek folder and the ones of HEK mode are located in elastest-toolbox/kubernetes/hek folder.

Also, ElasTest Platform can be used to deploy a Kubernetes cluster in a cloud provider first and then deploy ElasTest on top of it. Openstack and AWS are currently supported. The following command will connect ElasTest Platform to specified cloud provider, deploy a kubernetes cluster on it and then deploy ElasTest on that Kubernetes cluster:

docker run --rm -v ~/.elastest:/data -v /var/run/docker.sock:/var/run/docker.sock elastest/platform start --kubernetes --paas-type=value --paas-url=value --paas-user=value --paas-pass=value --paas-project-name=value

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:

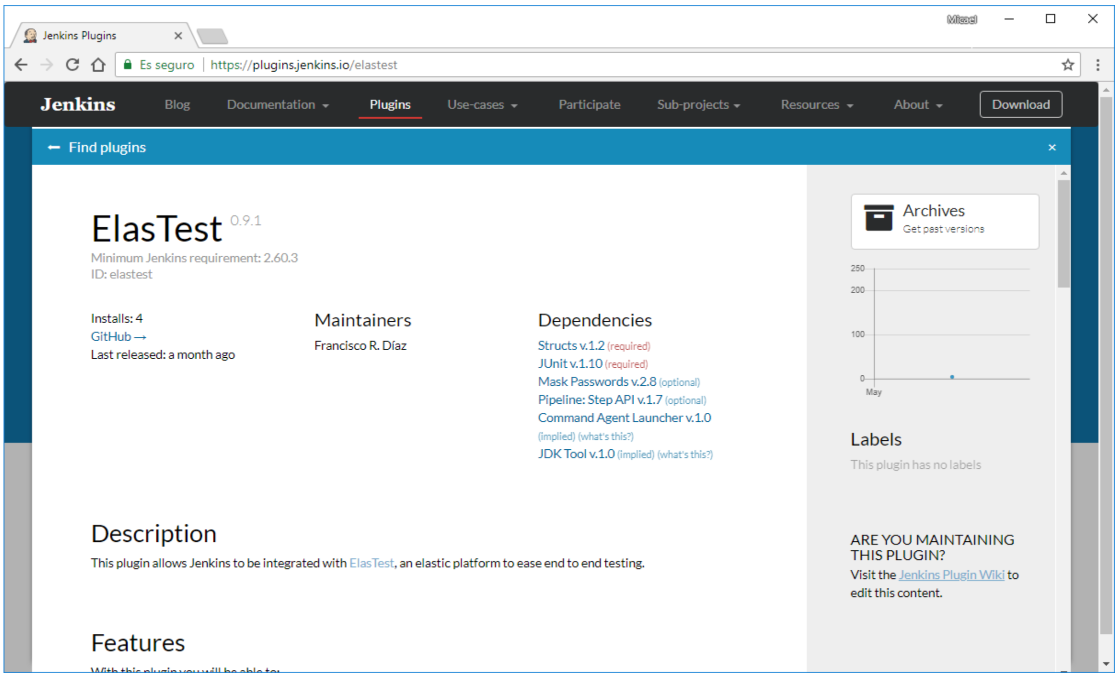
- **Jenkins:** The leading open source continuous integration tool. It can be extended with plugins to augment its features and integrate it with other tools.
- **TestLink:** The leading open source tool for test management, especially with manual testing.

In the following subsections, these two integrations are described.

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 5 shows ElasTest Jenkins plugin official web page[4].

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.



ElasTest 0.9.1
Minimum Jenkins requirement: 2.60.3
ID: elastest

Installs: 4
GitHub →
Last released: a month ago

Maintainers
Francisco R. Díaz

Dependencies
Structs v.1.2 (required)
JUnit v.1.10 (required)
Mask Passwords v.2.8 (optional)
Pipeline: Step API v.1.7 (optional)
Command Agent Launcher v.1.0 (implied) (what's this?)
JDK Tool v.1.0 (implied) (what's this?)

Labels
This plugin has no labels

Description
This plugin allows Jenkins to be integrated with ElasTest, an elastic platform to ease end to end testing.

ARE YOU MAINTAINING THIS PLUGIN?
Visit the Jenkins Plugin Wiki to edit this content.

Features

Figure 5. ElasTest Jenkins Plugin official web page

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

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[5] and uses another framework called Stapler[6] to export plugin objects with a REST-like API.

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 6 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).

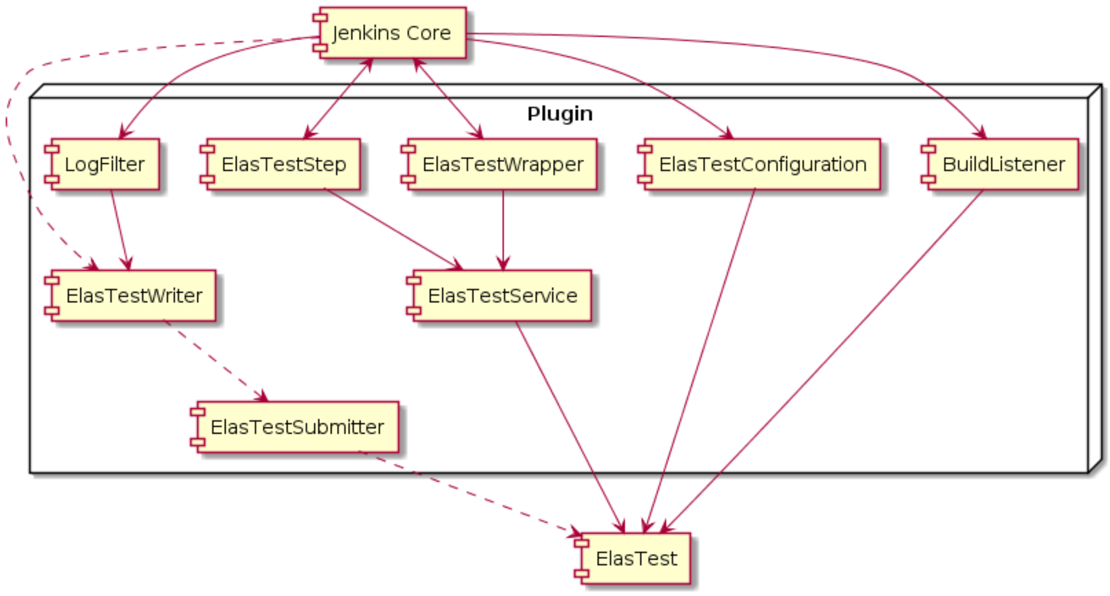


Figure 6. ElasTest Jenkins Plugin modules diagram

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

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

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

pipeline job is used, the module used is ElasTestStep. Job's log is sent to ElasTest using ElasTestWriter. Plugin configuration is managed with ElasTestConfiguration.

Plugin Modules

The main modules of the plugin 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[7] 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 deployment. The REST API provided by ETM allows ElasTest Jenkins plugin 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 7 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.

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

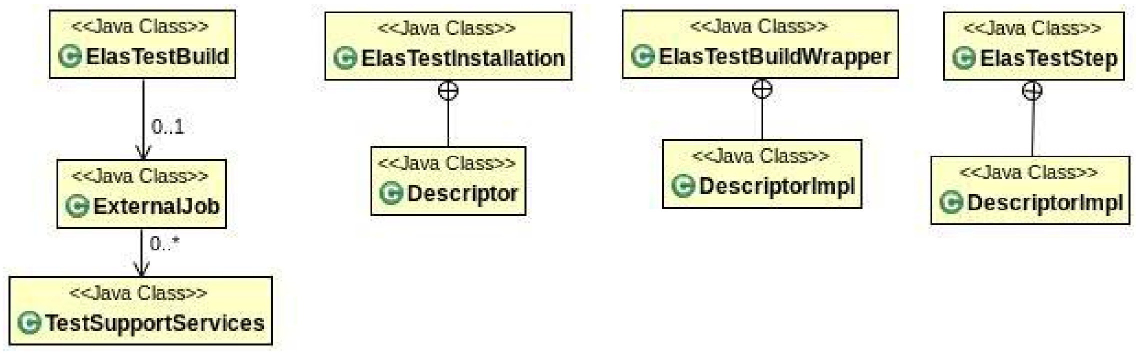


Figure 7. Data model of the Jenkins plugin

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.

- **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

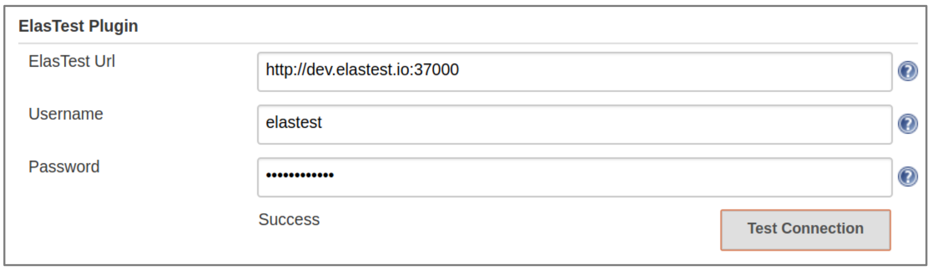
ElasTest Jenkins Plugin offers two main use cases to the user:

- 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:

Set up the Jenkins plugin

In this use case a user configures the plugin and test if that configuration is right (Figure 8. Plugin Configuration).


ElasTest Plugin

ElasTest Url	http://dev.elastest.io:37000
Username	elastest
Password	•••••••••••
	Success

Figure 8. ElasTest Jenkins Plugin Configuration

Figure 9 shows an UML sequence diagram with the interaction between Jenkins and ElasTest.

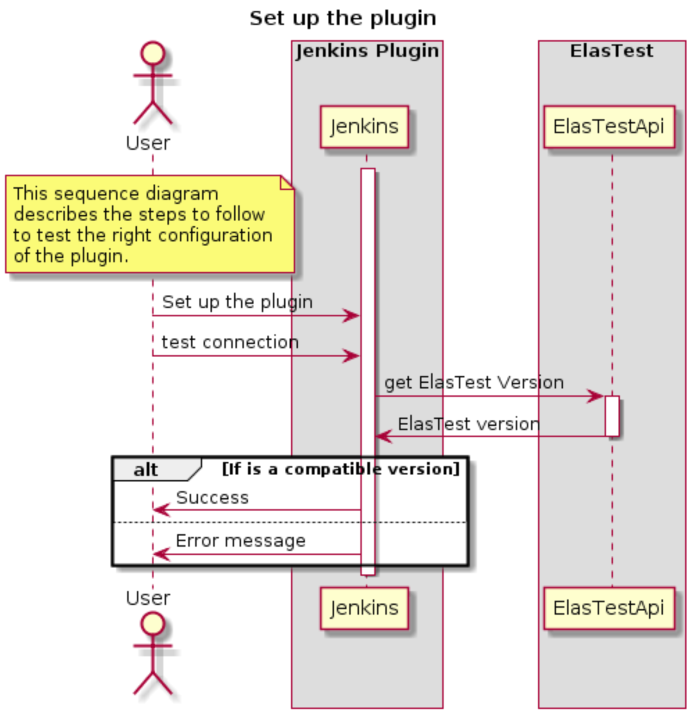


Figure 9. Jenkins and ElasTest interactions when plugin is configured

This diagram is expanded in Figure 10 to show how modules interact inside the plugin.

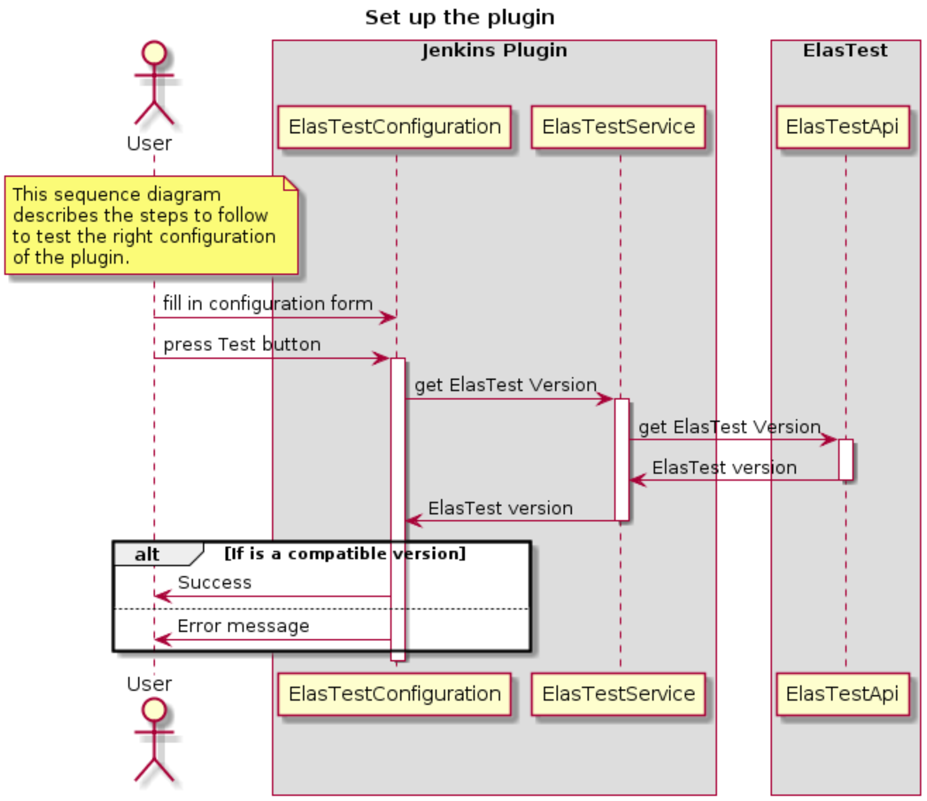


Figure 10. ElasTest Jenkins Plugin configuration

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.

Build a Job that uses the ElasTest Jenkins plugin

The interaction between Jenkins plugin and ElasTest when a job is executed is shown in the UML sequence diagram of Figure 11.

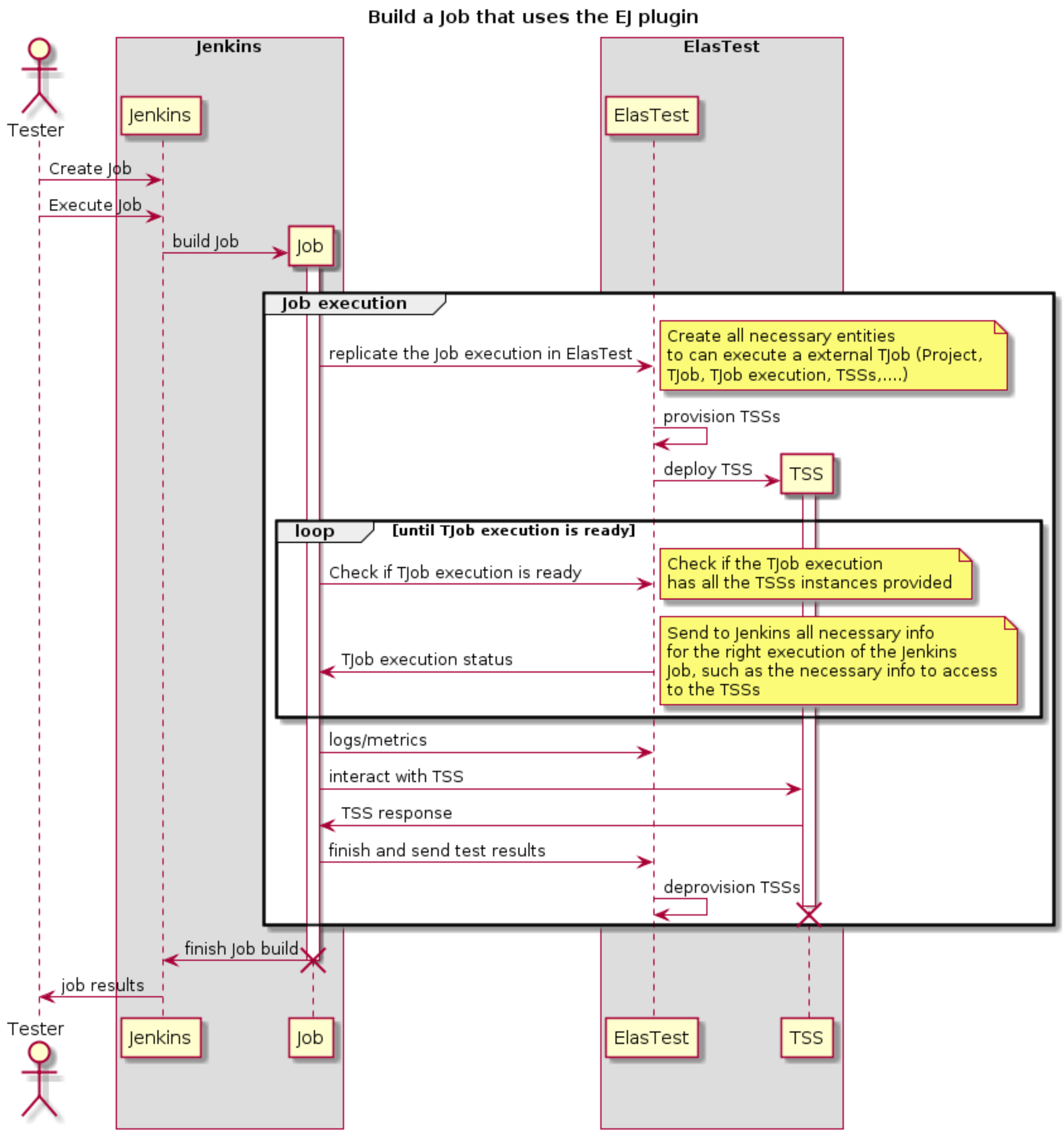

Build a Job that uses the EJ plugin
Jenkins
ElasTest
Tester
Jenkins
ElasTest
Create Job
Execute Job
build Job
Job
Job execution
replicate the Job execution in ElasTest
Create all necessary entities to can execute a external TJob (Project, TJob, TJob execution, TSSs,....)
provision TSSs
deploy TSS
TSS
loop
[until TJob execution is ready]
Check if TJob execution is ready
Check if the TJob execution has all the TSSs instances provided
TJob execution status
Send to Jenkins all necessary info for the right execution of the Jenkins Job, such as the necessary info to access to the TSSs
logs/metrics
interact with TSS
TSS response
finish and send test results
deprovision TSSs
finish Job build
job results
Tester
Jenkins
Job
ElasTest
TSS

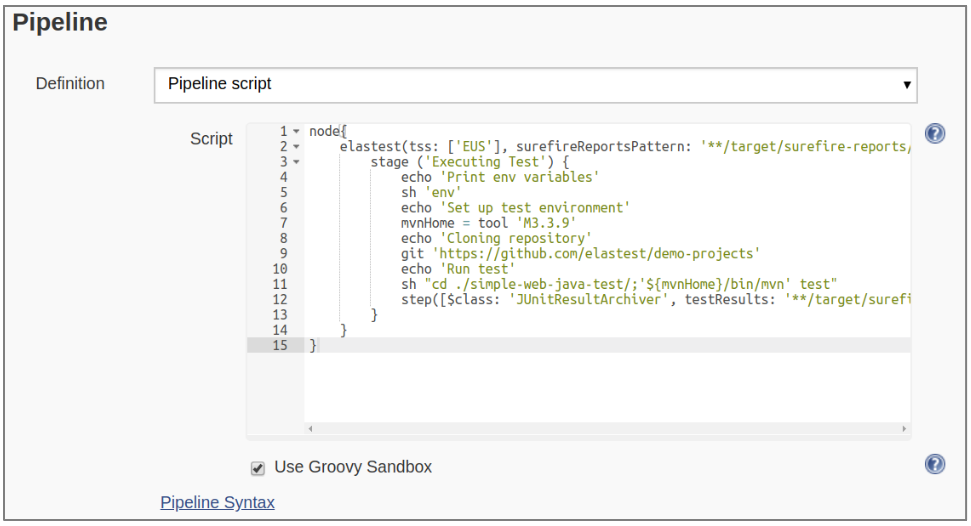
Figure 11. Jenkins job build using the ElasTest Jenkins plugin

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.

Using ElasTest in a pipeline Jenkins job

Figure 12 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 13 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.



```
node{
    elastest(tss: ['EUS'], surefireReportsPattern: '**/target/surefire-reports/
        stage ('Executing Test') {
            echo 'Print env variables'
            sh 'env'
            echo 'Set up test environment'
            mvnHome = tool 'M3.3.9'
            echo 'Cloning repository'
            git 'https://github.com/elastest/demo-projects'
            echo 'Run test'
            sh "cd ./simple-web-java-test/;'${mvnHome}/bin/mvn' test"
            step([$class: 'JUnitResultArchiver', testResults: '**/target/surefi
        }
    }
}
```

Figure 12. Using the plugin in a pipeline job

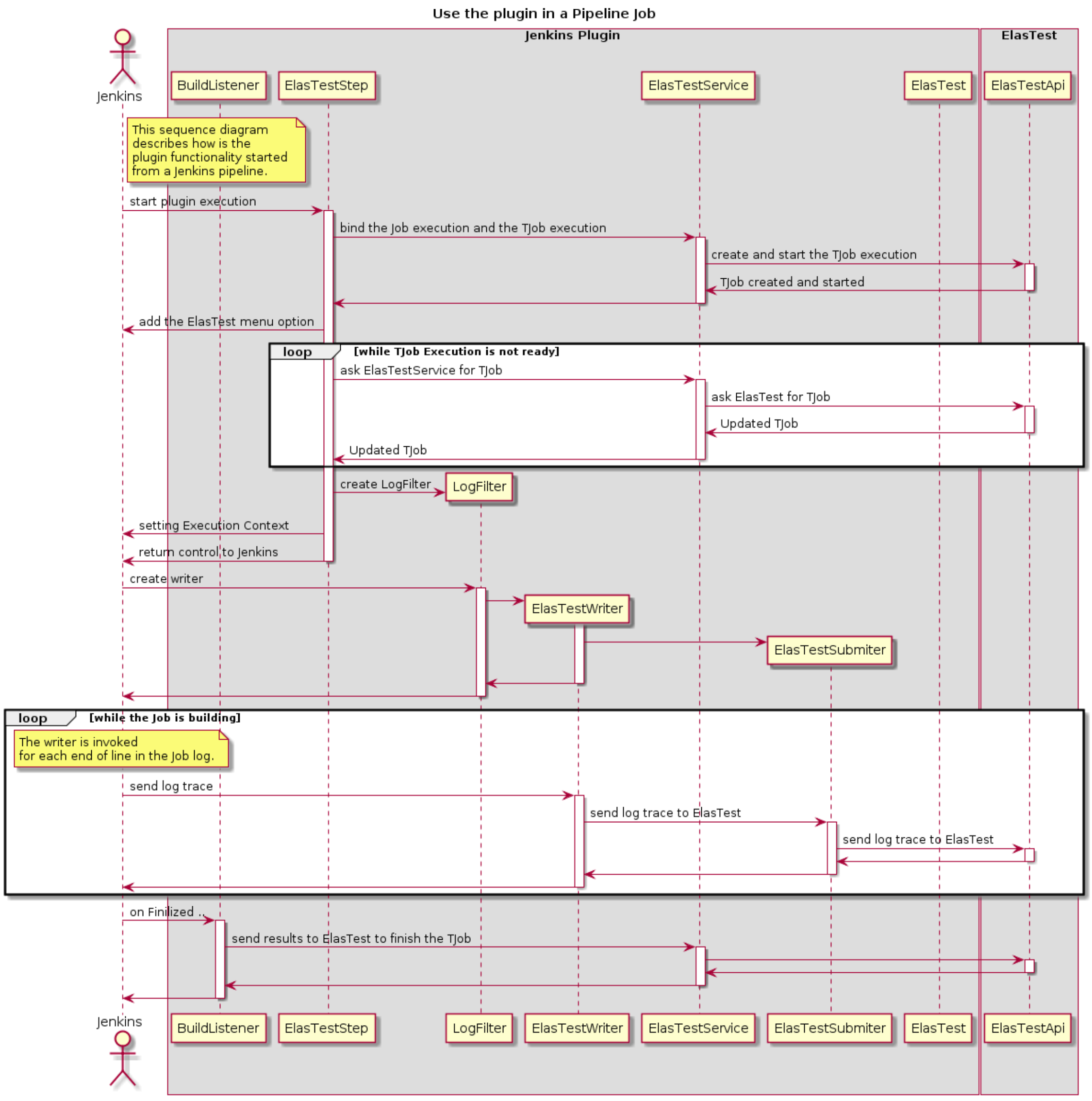
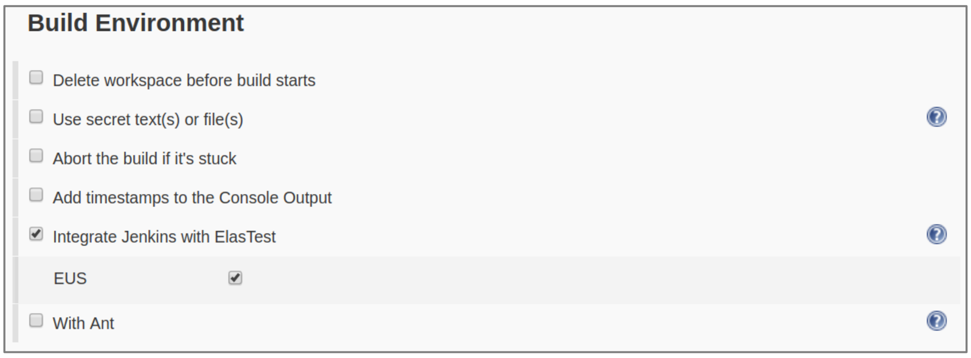

Use the plugin in a Pipeline Job
Jenkins Plugin
ElasTest
Jenkins
BuildListener
ElasTestStep
ElasTestService
ElasTest
ElasTestApi
This sequence diagram describes how is the plugin functionality started from a Jenkins pipeline.
start plugin execution
bind the Job execution and the TJob execution
create and start the TJob execution
TJob created and started
add the ElasTest menu option
loop [while TJob Execution is not ready]
ask ElasTestService for TJob
ask ElasTest for TJob
Updated TJob
Updated TJob
create LogFilter
LogFilter
setting Execution Context
return control to Jenkins
create writer
ElasTestWriter
ElasTestSubmiter
loop [while the Job is building]
The writer is invoked for each end of line in the Job log.
send log trace
send log trace to ElasTest
send log trace to ElasTest
on Finilized
send results to ElasTest to finish the TJob

Figure 13. Plugin behavior in a pipeline job

Using ElasTest in a freestyle Jenkins job

When the plugin is used in a freestyle job (Figure 14), *ElasTestWrapper* module is the main plugin module. The behavior is different in pipeline jobs (described before) and in freestyle jobs (as shown in Figure 15). 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 Job. 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).



Build Environment

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s)
- ☐ Abort the build if it's stuck
- ☐ Add timestamps to the Console Output
- ☑ Integrate Jenkins with ElasTest
  - EUS ☑
- ☐ With Ant

Figure 14. Plugin configuration in a Freestyle Job

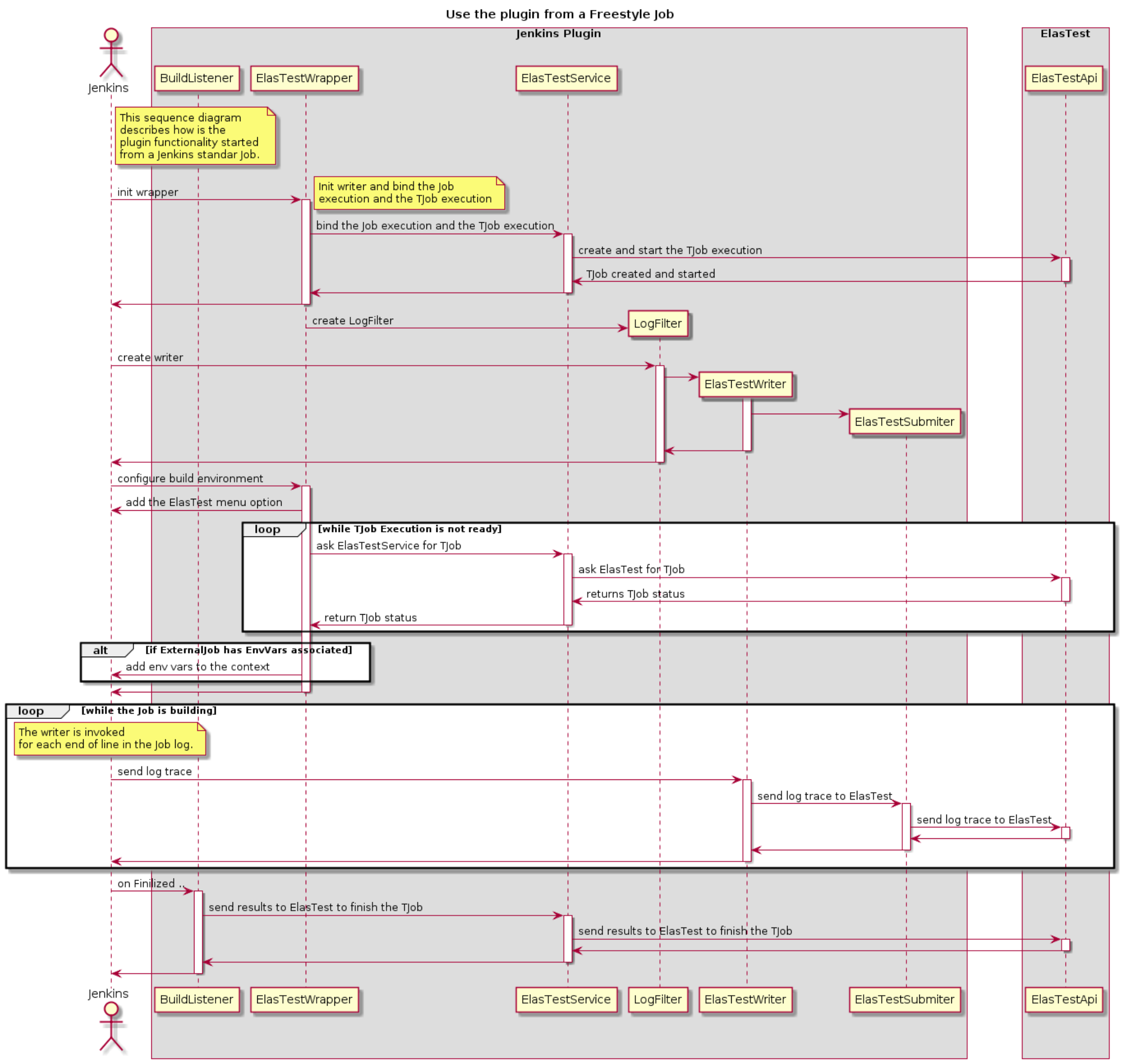

Use the plugin from a Freestyle Job
Jenkins Plugin
ElasTest
Jenkins
BuildListener
ElasTestWrapper
ElasTestService
ElasTestApi
This sequence diagram describes how is the plugin functionality started from a Jenkins standar Job.
init wrapper
Init writer and bind the Job execution and the TJob execution
bind the Job execution and the TJob execution
create and start the TJob execution
TJob created and started
create LogFilter
LogFilter
create writer
ElasTestWriter
ElasTestSubmiter
configure build environment
add the ElasTest menu option
loop
[while TJob Execution is not ready]
ask ElasTestService for TJob
ask ElasTest for TJob
returns TJob status
return TJob status
alt
[if ExternalJob has EnvVars associated]
add env vars to the context
loop
[while the Job is building]
The writer is invoked for each end of line in the Job log.
send log trace
send log trace to ElasTest
send log trace to ElasTest
on Finilized ..
send results to ElasTest to finish the TJob
send results to ElasTest to finish the TJob
Jenkins
BuildListener
ElasTestWrapper
ElasTestService
LogFilter
ElasTestWriter
ElasTestSubmiter
ElasTestApi

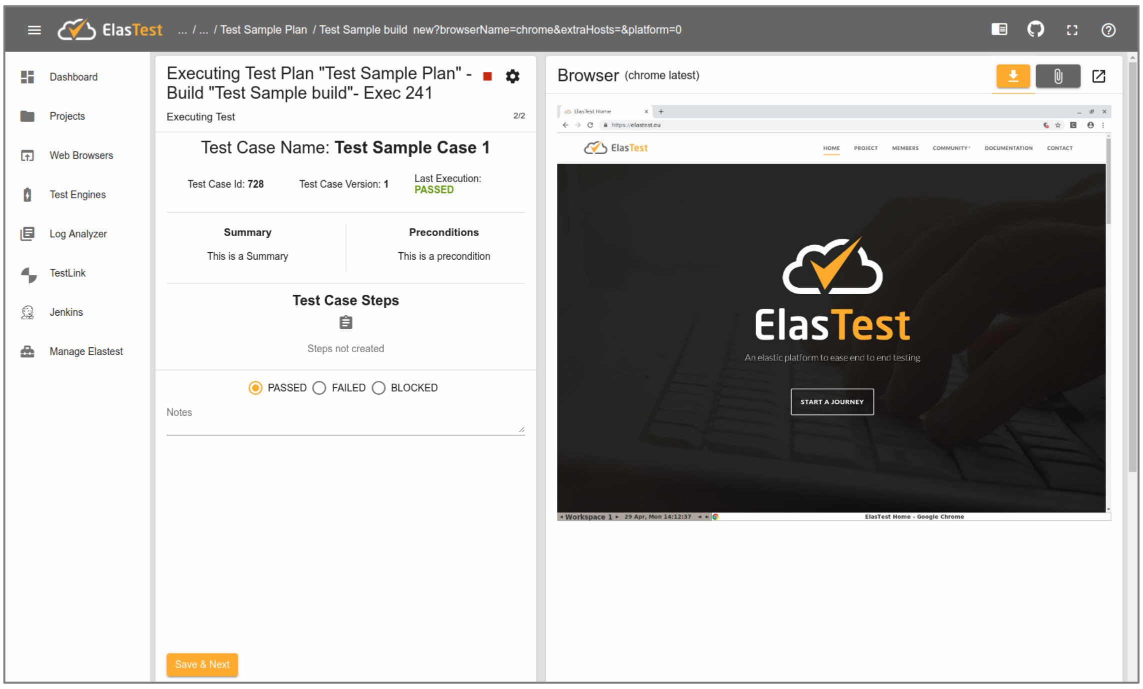
Figure 15. Plugin behavior in a Freestyle Job

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 behaviour 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 behaviour is caused by a bug that needs to be fixed. The process to register the current behaviour 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 behaviour 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 behaviour 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 16). 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, she can analyze all the gathered information using ElasTest tools.


ElasTest
... / ... / Test Sample Plan / Test Sample build new?browserName=chrome&extraHosts=&platform=0
Dashboard
Projects
Web Browsers
Test Engines
Log Analyzer
TestLink
Jenkins
Manage Elastest
Save & Next
Browser (chrome latest)
An elastic platform to ease end to end testing
START A JOURNEY

Executing Test Plan "Test Sample Plan" - Build "Test Sample build"- Exec 241

Executing Test 2/2

Test Case Name: **Test Sample Case 1**

Test Case Id: **728** Test Case Version: **1** Last Execution: **PASSED**

Summary	Preconditions
This is a Summary	This is a precondition

Test Case Steps

Steps not created

[✓ PASSED] FAILED BLOCKED

Notes

Figure 16. Running a TestLink test plan within ElasTest

4.2.1 Baseline concepts and technologies

TestLink provides a REST API[8] 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 17. The integration is designed to allow the user interacting with TestLink directly and through ElasTest main web interface.

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

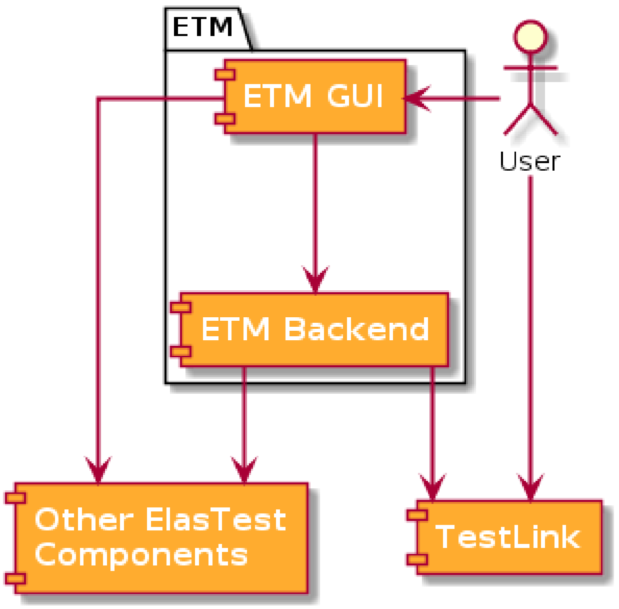


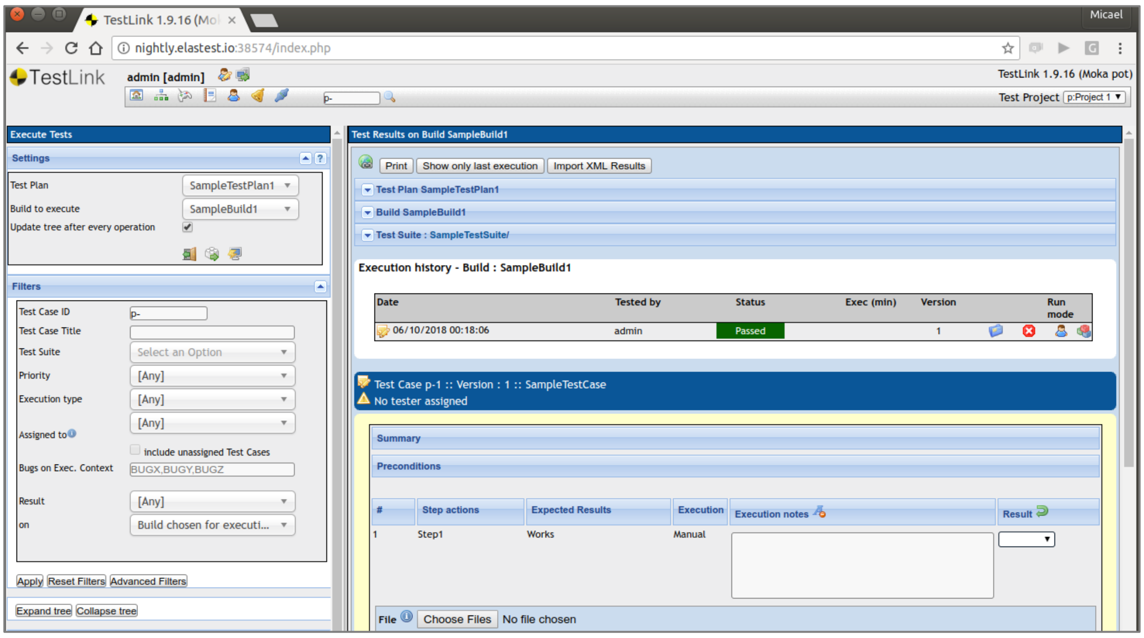
Figure 17. Module diagram of the integration between ElasTest and TestLink

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[9] and ETMBackend is the backend part implemented with Java and SpringBoot[10]. 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 18 shows the TestLink web interface. TestLink is started within ElasTest by default in singlenode mode.

[9] https://angular.io/

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


TestLink 1.9.16 (Mo
nightly.elastest.io:38574/index.php
admin [admin]
TestLink 1.9.16 (Moka pot)
Test Project: p:Project 1

Execute Tests

Settings

Test Plan	SampleTestPlan1
Build to execute	SampleBuild1
Update tree after every operation	✓

Filters

Test Case ID	p-
Test Case Title	
Test Suite	Select an Option
Priority	[Any]
Execution type	[Any]
Assigned to	[Any]
	include unassigned Test Cases
Bugs on Exec. Context	BUGX,BUGY,BUGZ
Result	[Any]
on	Build chosen for executi...

Test Results on Build SampleBuild1

Test Plan SampleTestPlan1
Build SampleBuild1
Test Suite : SampleTestSuite/

Execution history - Build : SampleBuild1

Date	Tested by	Status	Exec (min)	Version	Run mode
06/10/2018 00:18:06	admin	Passed		1	

Test Case p-1 :: Version : 1 :: SampleTestCase
No tester assigned

Summary
Preconditions

#	Step actions	Expected Results	Execution	Execution notes	Result
1	Step1	Works	Manual		

File: No file chosen

Figure 18. TestLink screenshot

4.2.3 Data Model

The data model of this integration between ElasTest and TestLink is shown in Figure 19. 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.

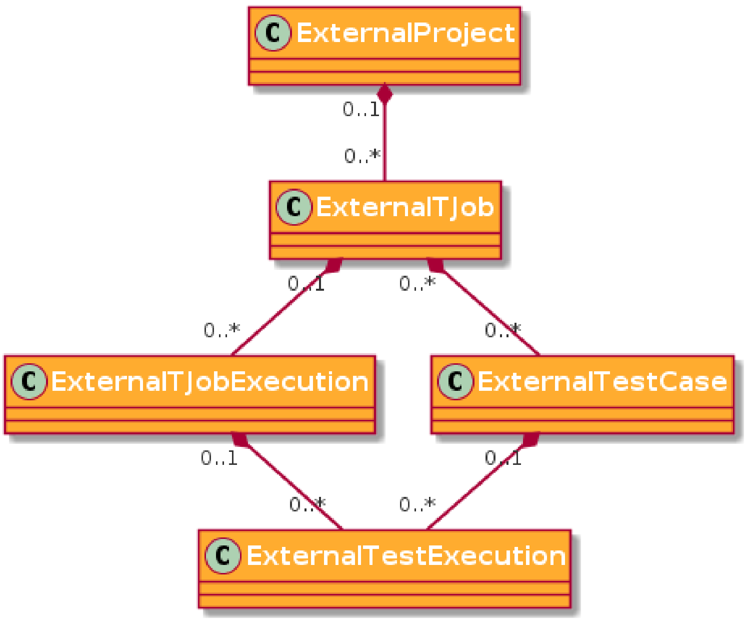


Figure 19. ETM TestLink Data Model

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.

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 20.

A test plan may have an associated sut, which will be started automatically by ElasTest when test plan is run.

The execution of a test plan can be terminated in a normal way, that is, by running all the test cases, but it can also be paused to resume later or simply terminate the execution abruptly.

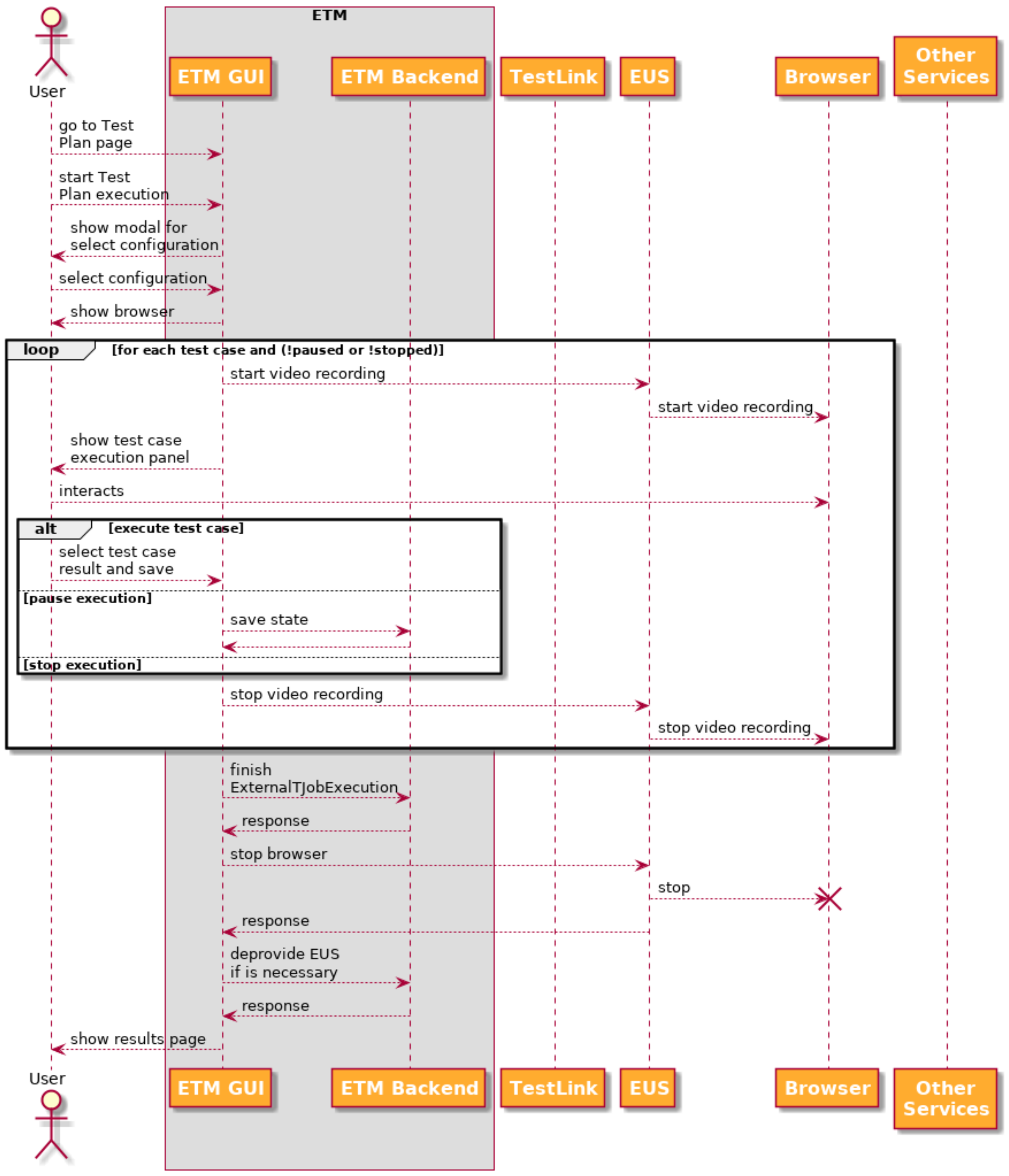


Figure 20. Execute a TestLink Test Plan in ElasTest

Another new feature is the ability to run a test plan with multiple browsers to do *cross browsing*. The user will be able to select two or more different browsers to be started with the test plan. The browsers will be shown during the execution and the user will be able to interact with any of them. When interacting with one, the events propagate to the rest, obtaining the same behavior in all of them, which is very useful to test an application in several browsers simultaneously, which will save time to the tester. Figure 21 shows how this functionality works.

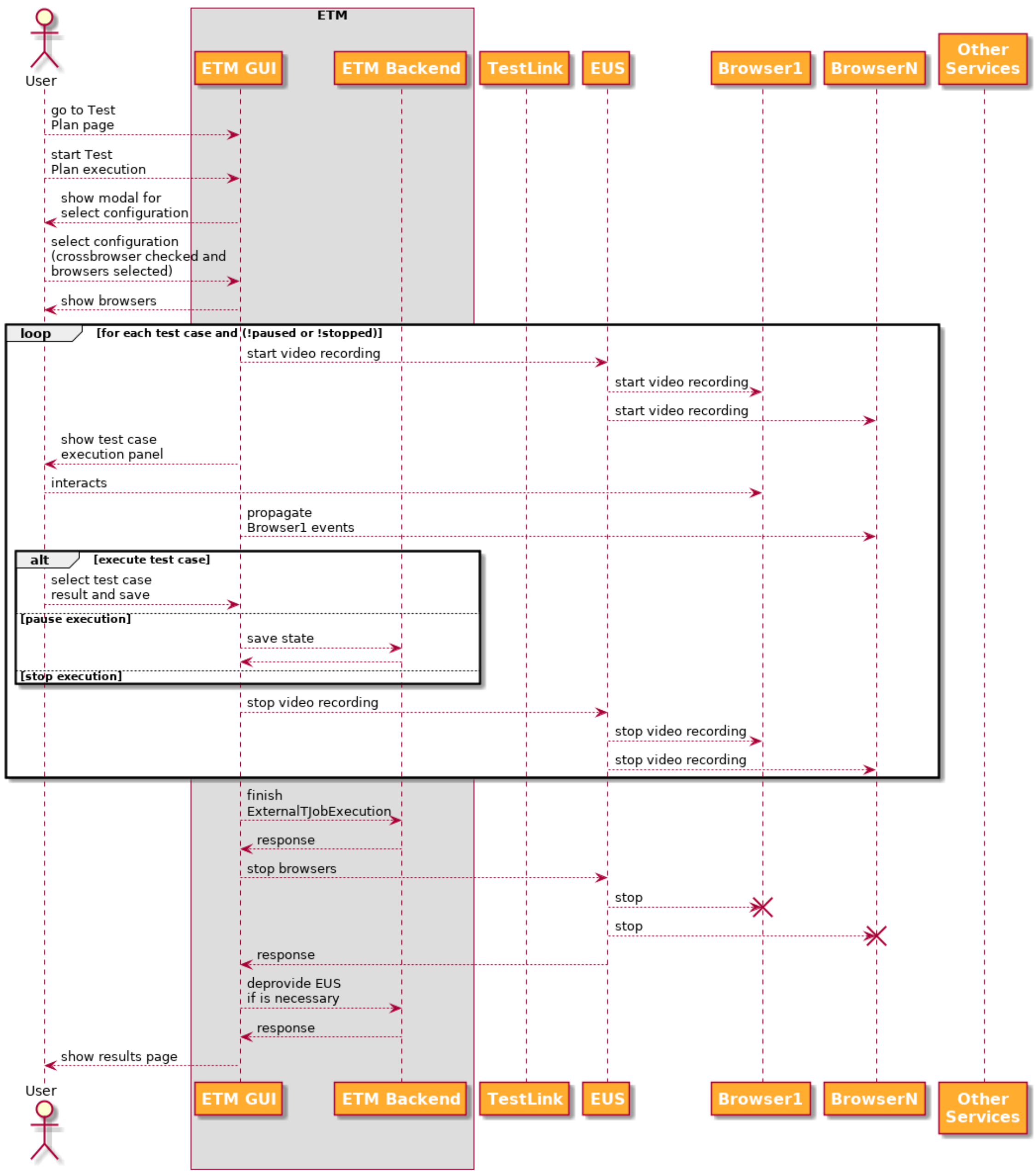


Figure 21. Execute a TestLink Test Plan in ElasTest with Cross Browsing

5 Conclusions and future work

During the second period of the project we focused on four different objectives regarding toolbox and integrations: provide a deployment mode with a reduced resource consumption to ease the triage of the tool, provide support for Kubernetes, improve the integration with Jenkins, and support our verticals through TestLink features.

We consider that we covered all of them in a better way that we anticipated. The ElasTest mini mode was made available short after our first review (end Summer 2018).

The Kubernetes support is available since September 2019, and it had some interest in the industry with a couple of companies interested on it (Okteto and Zooplus, see interest letters in D1.4). The Jenkins integration has been improved, so that all tasks on instrumentation are automatically performed by ElasTest without user intervention. And TestLink integration has been improved with the cross browser feature and some GUI enhancements.

As a general conclusion, ElasTest can be now used in many different environments (on public or private clouds, on Kubernetes, as standalone), thus increasing the number of potential users for the platform.

In the future, integrations for specific use cases from the companies with which we are collaborating are expected.

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