D6.1	
Version	1.0
Author	NAEVATEC
Dissemination	PU
Date	27-06-2018
Status	FINAL

D6.1: ElasTest Continuous Integration and Validation System v1

Project acronym	ELASTEST
Project title	ElasTest: an elastic platform for testing complex distributed large software systems
Project duration	01-01-2017 to 31-12-2019
Project type	H2020-ICT-2016-1. Software Technologies
Project reference	731535
Project website	http://elastest.eu/
Work package	WP6
WP leader	Guiomar Tuñón de Hita
Deliverable nature	PUBLIC
Lead editor	Guiomar Tuñón de Hita
Planned delivery date	30-06-2108
Actual delivery date	27-06-2018
Keywords	Open source software, cloud computing, continuous integration, continuous validation,

Funded by the European Union

Contributors

Name	Affiliation
Guiomar Tuñón	NAEVATEC
Francisco Ramón Díaz	URJC
Gordon Doyle	IBM
Piyush Harsh	ZHAW
Felipe Gorostiaga	IMDEA
Varun Gowtham	TUB
David Rojo	ATOS
Nikolaos Gavalas	REL
Francisco Gorostiaga	URJC
Micael Gallego	URJC

Version history

Version	Date	Author(s)	Description of changes
0.1	11/12/2017	Guiomar Tuñón	Initial version (DRAFT)
0.2	30/01/2018	Guiomar Tuñón	Contribution from partners for e2e tests.
0.3	20/04/2018	Guiomar Tuñón	2nd round of contributions from partners for e2e tests.
0.4	26/04/2018	Guiomar Tuñón	3rd round of contributions from partners for e2e tests.
0.5	11/06/2018	Enric Pages	Quality Review Process
0.6	21/06/2018	Guiomar Tuñón	Quality Review Process, apply changes.
0.7	22/06/2018	Guiomar Tuñón	Added Index of tables and figures
0.8	24/06/2018	Micael Gallego	Review alignment with architecture terminology

Table of contents

List of Figures

List of Tables

Glossary of acronyms

Abbreviation	Full definition
APIs	Application programming interfaces
AMI	Amazon Machine Images
AWS	Amazon Web Services
CI	Continuous Integration
CV	Continuous Validation
E2E	End-to-end
ECR	Elastic Container Registry
GUI	Graphical User Interface
OS	Operating System
SW	Software
UI	User Interface
QoE	Quality of Experience
SuT	Software under Test
SiL	Systems in the Large
TiL	Test in the Large
TSS	Test support service
TORM	Test Orchestration and Recommendation Manager
QoS	Quality of Service
UAT	User Acceptance Testing
IPR	Intellectual Property Rights

1 Executive summary

The present document describes the design, architecture and maintenance of the ElasTest Continuous Integration (CI) and Continuous Validation (CV) System used in the project. This system has been designed and maintained in the context of the Work Package 6 (WP6) “Continuous Integration & Validation”.

This document describes the design, architecture and maintenance of the ElasTest CI environment, including aspects such as the following:

- Description of the strategic objectives to be fulfilled.
- Description of the environment design and architecture.
- Description of the available tools in the environment.
- Description of the maintenance plan.
- Description of the CI and CV methodology.

During the upcoming eighteen months the content of this document will be reviewed and updated with any new tools that could be added to the environment, new or revised procedures as well as new cloud instances that would be needed.

The initial environment devised for running CI/CV tasks for the project started with a single instance that ran the main tools related to the software development process and from the first release of the ElasTest platform it has grown having now four instances, each one with a clear objective regarding the tasks that are meant to be executed over them:

I. Main Instance. Holds the main tools related to the software development process – CI server, repositories, credential generator, etc. –
II. ElasTest Nightly Instance. Hosts the latest developed version of ElasTest (not necessarily stable). This instance main objective is to provide an ElasTest platform where latest changes on the code could be tested.
III. ElasTest Stable Instance. Host the latest stable version of ElasTest. This instance will be used to test the ElasTest Nightly Instance with ElasTest, as specified in the DoA.

In order to have a complete and intensive test suite for the whole ElasTest platform all the components have contributed with specific test suits for their components, these tests are continuously changing as the functionalities of the components expand and mutate, so the suites cover this new or changed functionalities. In the deliverable these suites are described as they are at the moment of writing, whereas these descriptions could be outdated in a few weeks, as the platform is continuously improved.

In section 5 of the document we define the scheduled next steps into the Continuous Validation and Integration System that covers tasks such as completion the validation of ElasTest with ElasTest.

2 Strategic context and objectives

The ElasTest CI environment and methodology has been designed with the objective of providing the project with a complete set of tools and procedures that must grant the appropriate level of quality of each component and the right integration of all of them.

The CI methodology comprises all the tasks that assure:

- High quality of each of the components from development to release.
- High quality of integration between components.
- High quality of ElasTest as a whole.
- High quality of the CI methodology and CI environment.

The CI environment comprises all the tools that help to achieve and maintain the highest levels of quality in all the steps of the development, testing and release.

The specific configuration of the consortium and the diverse licenses (public/Apache 2.2 and Proprietary) of the components are managed within the CI tasks and tools to grant the appropriate access and dissemination of each component.

The following sections contain the details of the CI Environment [Section 3], and the CI and CV methodology [Section 4].

The current status of the CI and CV system is aligned with the declared Milestones in the DoA, where MS4 scheduled for month 16, where it was required from the ElasTest CI system to be fully qualified and in production to enable the delivery and testing of all ElasTest components. As this document demonstrates this objective is completed. For month 24 it is defined MS5 where it is expected to have a mature implementation of ElasTest validated in laboratory through the ElasTest CI system and tests. Here we include to have ElasTest tested by ElasTest. Actually the CI / CV system includes an ElasTest instance with the objective of being used with this purpose and as soon as the ElasTest platform is mature enough it will be used on for validating newer versions of ElasTest.

3 CI environment

The CI environment is composed by a set of tools managed by NaevaTec and available to the consortium partners.

The CI environment has two kinds of applications/tools: self-hosted services and provided services. Self-hosted services are those that have been deployed on our own managed servers. Those are fully managed by NaevaTec. This requires the CI administrator (NaevaTec) to manage security, access policy, system stability and maintenance (corrective and upgrades). On the other hand, provided services are those that hosted on the providers premises or clouds and serve the technologies and services mainly through an accessible web URL.

3.1 Self-hosted Services

3.1.1 Main Instance.

The core of the environment is a cloud (AWS) hosted instance that runs the self-hosted tools. All the applications hosted in this instance are containerized for 3 main reasons:

- **Simplicity of maintenance:** having a well-maintained environment where times without service are rare and short is a very much valued property for any service.
- **Simplicity of recovery:** if somehow the system is unstable, breaks, or its security is compromised, the application configuration and data can be recovered easily and the risks of data loss are reduced considerably.
- **Simplicity for replication:** This environment can be easily reproduced and customized to be used in other projects.

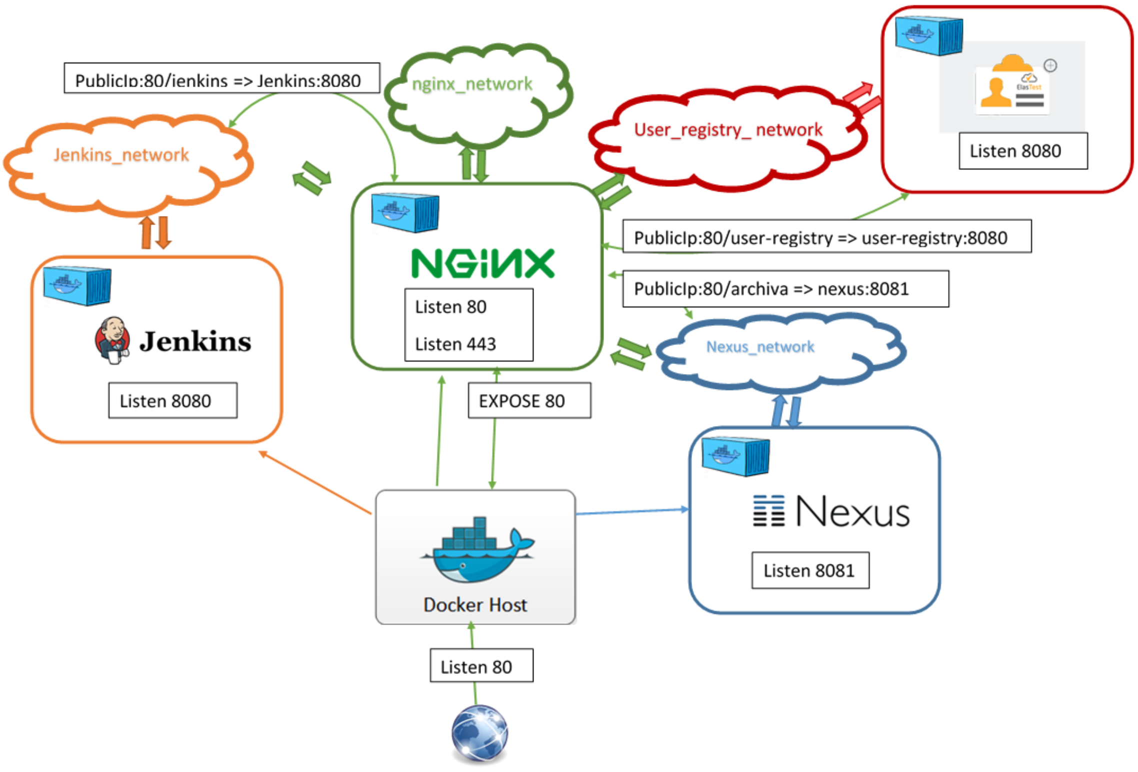


Figure 1. Self-hosted services architecture

The CI Server (Jenkins) should run different jobs for compiling, testing [4.5] and delivering [4.6] the ElasTest components. In order to maintain the core instance clean, secure and available without interference of the running jobs, we have designed a system of volatile slaves. Those slaves are launched on demand for the jobs into the (AWS) cloud, they are managed by the CI server and they are inaccessible from outside the environment.

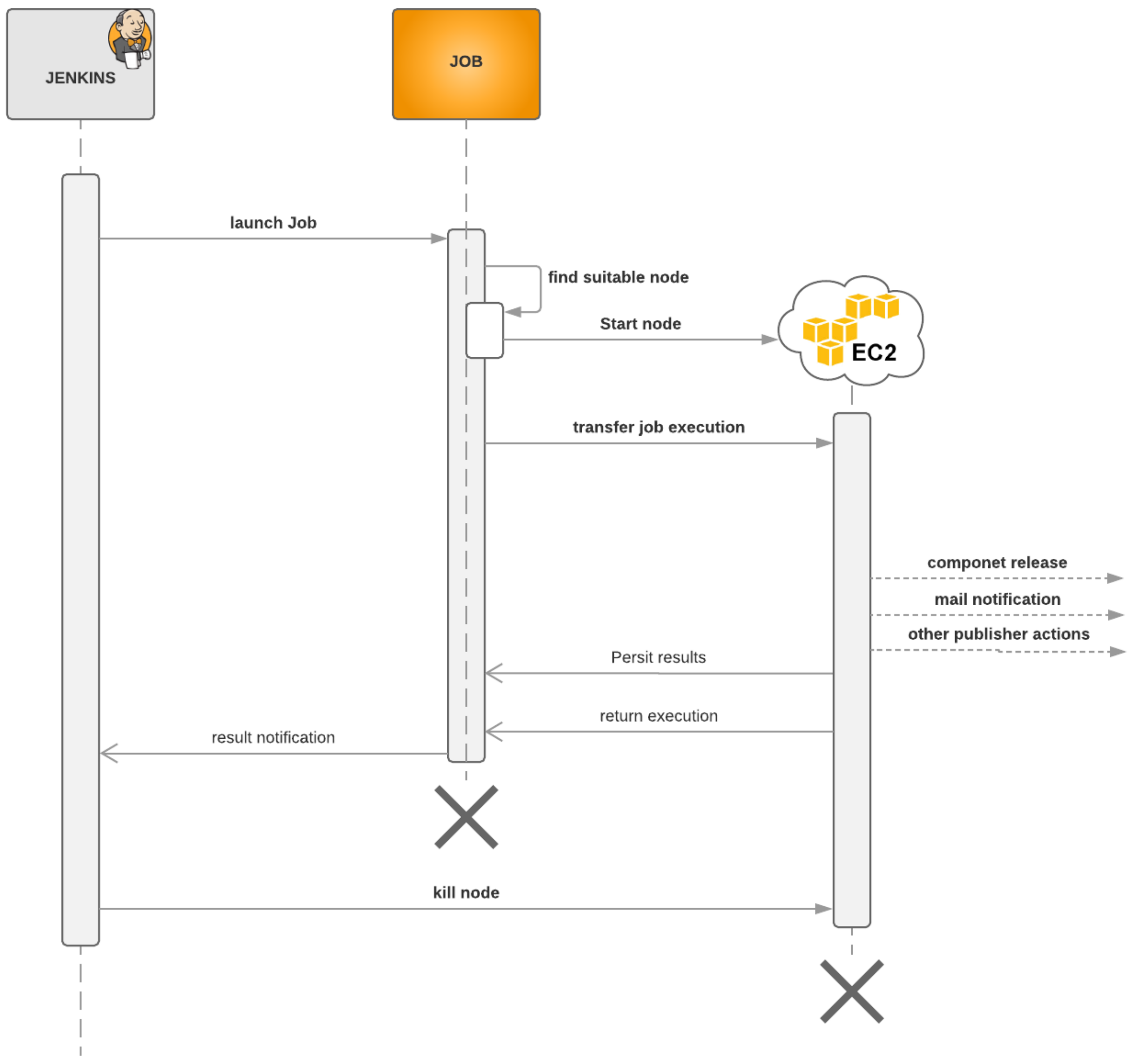


Figure 2. Jenkins jobs execution sequence

The last entity in our environment of self-hosted tools is an ElasTest Instance running in a (AWS) cloud instance. This ElasTest instance is accessible from everywhere but protected by basic authentication -currently the only one that is available in ElasTest-. This instance is isolated from the rest of the environment, it can be accessed by volatile nodes and the core instance but the other way is closed.

3.1.2 ElasTest Nightly Instance.

The ElasTest Nightly Instance is an AWS EC2 instance has been commissioned with the objective of having a full operative ElasTest platform running so the partners can test the functionality. It has been created with the Amazon CloudFormation template that can be found as a part of the elastest-toolbox repository [60].

This instance is exclusively devised with the objective of hosting an ElasTest platform. This ElasTest platform is automatically daily refreshed, so it is always running the last developed version of the platform.

As the platform has evolved the type of instance needed for a correct behaviour of the system has also changed. AWS EC2 instances allows us to adapt the instance running to the real requirements of the ElasTest platform.

3.1.3 ElasTest Stable Instance.

The ElasTest Stable Instance is the last addition to the ElasTest CI/CV system. It is nearly a copy of the ElasTest Nightly Instance which main difference is that on the ElasTest Stable Instance, there isn't an automatic update of the ElasTest platform, but the update of the ElasTest platform is done manually wherever a new stable version is released.

3.2 Tools.

3.2.1 Tool chain.

Name	Type	License	Self-hosted?	Access	Description
GitHub[8]	Source code repository	Proprietary	No	Public	GitHub is a Web-based Git version control repository hosting service. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.
Jenkins[1]	CI Server	OSS (MIT)	Yes	Consortium only	Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks such as building, testing, and deploying software.
DockerHub [2]	Docker image repository	Proprietary	No	Public	The Docker Hub Registry is free to use for public repositories. Plans with private repositories are available in different sizes. All plans allow collaboration with unlimited people.
OSSRH[3]	Maven and Gradle artifact repository	OSS (Eclipse)	No	Public	Sonatype OSSRH (OSS Repository Hosting) uses Sonatype Nexus Repository Manager to provide repository hosting service for open source project binaries.

Nexus Repository Manager OSS[4]	Maven and Gradle artifact repository	OSS (Eclipse)	Yes	Consortium only	Nexus Repository OSS is a universal repository manager with support for all major package formats and types.
Private User Registry[5]	Custom user access manager	Proprietary	Yes	Consortium only	Private User Registry is a service developed by NaevaTec to manage the access to private Nexus Repository and Amazon ECR, providing access to only Consortium members to the resources published there.
ElasTest[6]	--	OSS (Apache)	Yes	Consortium only	An elastic platform to ease end to end testing.
Amazon ECR[7]	Docker image repository	Proprietary	No	Consortium only	Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.

Table 1. CI environment main tools.

3.2.1.1 GitHub

All the public repositories regarding ElasTest components, documentation, webpages, etc. are hosted in GitHub.

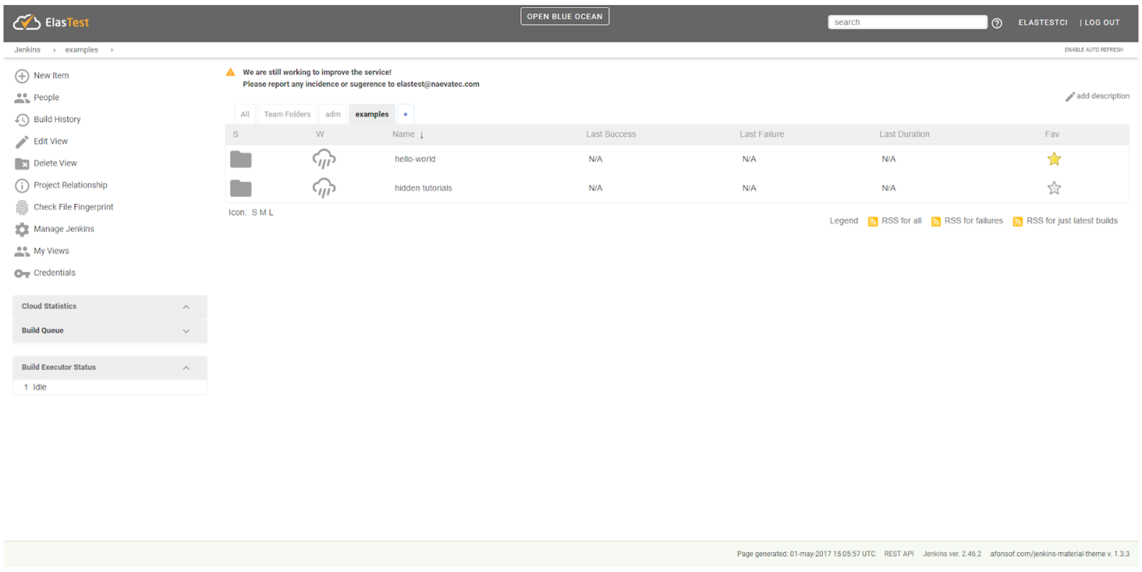
Also, GitHub is the main tool used for managing access to the different tools and contents related to the ElasTest project. This way teams for each partner and component are managed trough GitHub Teams (See 3.2.2.1 User Access).

3.2.1.2 CI Server: Jenkins

Jenkins [1] will serve as CI server. We have chosen Jenkins over other alternatives (Atlassian Bamboo [55], CruiseControl [56], Team Foundation Server [57], Travis CI [58], etc.) because of the characteristics described in the original proposal:

- *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.*

Travis CI and Jenkins both comply with the project requirements. Being both suitable we chose Jenkins because among the open source alternatives, Jenkins is the one with the biggest market share, with a strong community and a lot of plugins and the knowledge and expertise we already have on Jenkins configuration, usage and plugins development.



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

S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
		hello-world	N/A	N/A	N/A	★
		hidden tutorials	N/A	N/A	N/A	☆

1 Idle

Page generated: 01-may-2017 15:05:57 UTC REST API Jenkins ver. 2.46.2 afonsof.com/jenkins-material-theme v. 1.3.3

Figure 3. ElasTest Jenkins.

The partners would be using Jenkins in order to validate and integrate their work. Also, Jenkins will provide a safe environment for each of the partners where only authorized people will run partners' defined jobs.

Users within the ElasTest Organization in GitHub that are members of any of the partner's teams would be granted access. Anybody inside the ElasTest Organization that isn't member at least of one of the partners' team won't be able to access.

Each user will have permissions over the different folders. By default, there will be a folder for each team in the GitHub organization and each user will have permission over the jobs inside the folders of all the groups he/she belongs to.

All users that need to be granted additional permissions such as administrative permissions or permissions to other jobs outside his/her groups, can request them filling a row in the Jenkins sheet of the User Registration spreadsheet.

3.2.1.2.1 Slaves

Jenkins supports the "master/slave" mode, where the workload of building projects is delegated to multiple "slave" nodes, allowing a single Jenkins installation to host a large number of projects, or to provide different environments needed for builds/tests.

A Jenkins slave can be seen as an external provider of executors for running jobs.

The Jenkins master can't be used for any user jobs, all the user jobs by default will be launched on an Amazon EC2 Instance that will act as Jenkins Slave. This slave will be launch on demand as a job is programmed to be run. If the slave is already available and has free executors it will be launch at the moment but if the slave isn't up or no executors are available the job will wait to the slave to be launched (automatically) or an executor is released.

3.2.1.2.2 Jobs

All users will be allowed to create jobs inside the folders that belongs to the teams that he/she belongs to.

Users are encouraged to containerize their builds and tests.

Procedures and/or recommendations will be explained in section 4.1 -*Methodology and Procedures*- for different kinds of jobs. Also, there are shared folders with some jobs examples inside the Jenkins.

3.2.1.2.3 ElasTest Plugin

The ElasTest Jenkins plugin developed in the context of the WP6 Task 6.4 and described in D6.2 ElasTest platform toolbox and integrations v1 isn't currently available in the ElasTest CI server. It will be added in future maintenances.

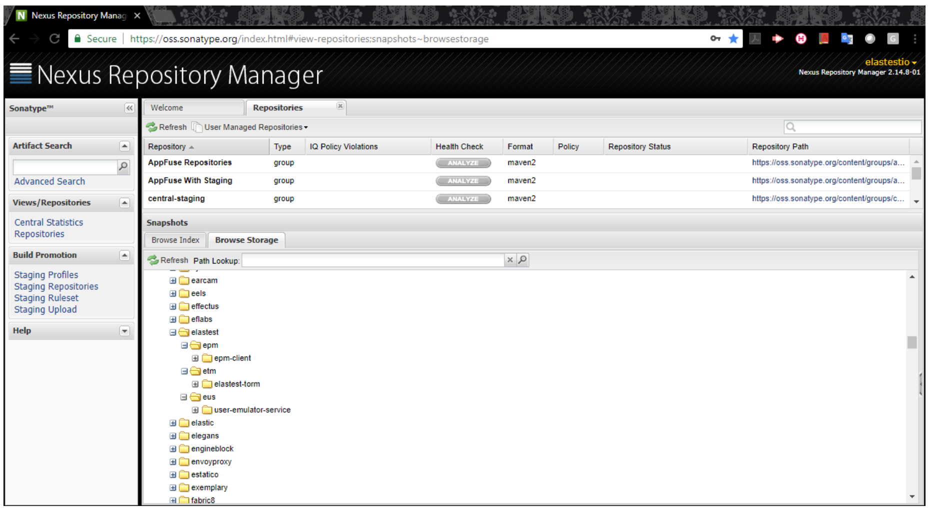
3.2.1.3 Sonatype OSSRH

As ElasTest is an Open Source project, all the public artifacts can be hosted on Sonatype OSSRH (OSS Repository Hosting) [3]. This repository hosting allows ElasTest to maintain a public release repository that will be published to Central automatically, and also a public snapshot repository.

We have been granted with the io.elastest *groupId* inside the central maven repository. So, all the artifacts developed in the context of ElasTest should be within the io.elastest group.

Full documentation on how to publish ElasTest artifacts have been provided to all the partners. And the procedure description can be found in section **4.6 Releasing.**

Any user (consortium member or not) will have access to the artifacts here deployed (releases and snapshots).



Repository	Type	IQ Policy Violations	Health Check	Format	Policy	Repository Status	Repository Path
AppFuse Repositories	group		ANALYZE	maven2			https://oss.sonatype.org/content/groups/a...
AppFuse With Staging	group		ANALYZE	maven2			https://oss.sonatype.org/content/groups/a...
central-staging	group		ANALYZE	maven2			https://oss.sonatype.org/content/groups/c...

```
earcam
eels
effectus
eflabs
elastest
  epm
    epm-client
  etm
    elastest-torm
  eus
    user-emulator-service
elastic
elegans
engineblock
envoyproxy
estatico
exemplary
fabric8
```

Figure 4. Sonatype OSSRH. Public maven repository

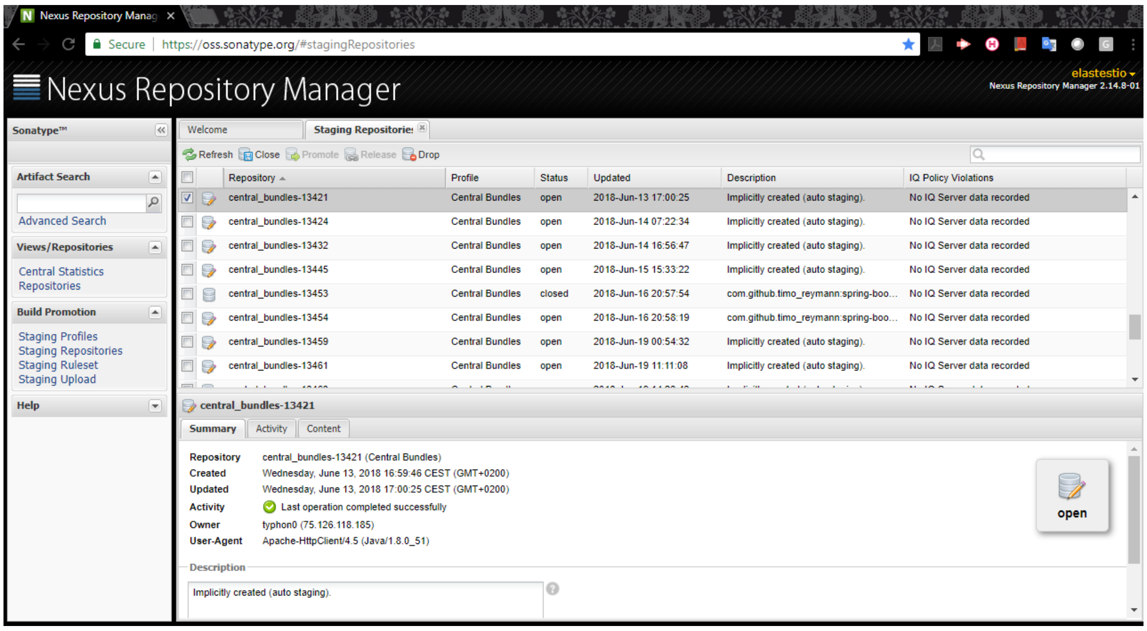
3.2.1.4 *Nexus Repository Manager (OSS).*

As described in D8.3 the IPR (Intellectual property rights) of most components are licensed under Apache 2.0 or MIT 2, so they are open source but there is one component at the moment that has one proprietary license (belonging to IBM). This forces the CI environment to provide private repositories for that component's artifacts and images.

All the artifacts that shouldn't be shared with the public but are necessary to collaborate within the consortium will be deployed to our own hosted nexus repository [4].

Access to this repository will be nominative even for read-only purposes, this will grant some control over the artifacts that are shared there. The access to the repository will be solicited with the User-registry application that will automatically check the access with the user GitHub account. Any user without a GitHub account or not member of the ElasTest organization in Jenkins will not be allowed to access to the repositories.

Full documentation on how to publish ElasTest private maven artifacts have been provided to all the partners. And the procedure description can be found in section **4.6 Releasing.**



Repository	Profile	Status	Updated	Description	IQ Policy Violations
central_bundles-13421	Central Bundles	open	2018-Jun-13 17:00:25	Implicitly created (auto staging).	No IQ Server data recorded
central_bundles-13424	Central Bundles	open	2018-Jun-14 07:22:34	Implicitly created (auto staging).	No IQ Server data recorded
central_bundles-13432	Central Bundles	open	2018-Jun-14 16:56:47	Implicitly created (auto staging).	No IQ Server data recorded
central_bundles-13445	Central Bundles	open	2018-Jun-15 15:33:22	Implicitly created (auto staging).	No IQ Server data recorded
central_bundles-13453	Central Bundles	closed	2018-Jun-16 20:57:54	com.github.timo_reymann:spring-boo...	No IQ Server data recorded
central_bundles-13454	Central Bundles	open	2018-Jun-16 20:58:19	com.github.timo_reymann:spring-boo...	No IQ Server data recorded
central_bundles-13459	Central Bundles	open	2018-Jun-19 00:54:32	Implicitly created (auto staging).	No IQ Server data recorded
central_bundles-13461	Central Bundles	open	2018-Jun-19 11:11:08	Implicitly created (auto staging).	No IQ Server data recorded

central_bundles-13421

Summary | Activity | Content

Repository	central_bundles-13421 (Central Bundles)
Created	Wednesday, June 13, 2018 16:59:46 CEST (GMT+0200)
Updated	Wednesday, June 13, 2018 17:00:25 CEST (GMT+0200)
Activity	Last operation completed successfully
Owner	typhon0 (75.126.118.185)
User-Agent	Apache-HttpClient/4.5 (Java/1.8.0_51)

open

Description

Implicitly created (auto staging).

Figure 5. Nexus Repository Manager (OSS)

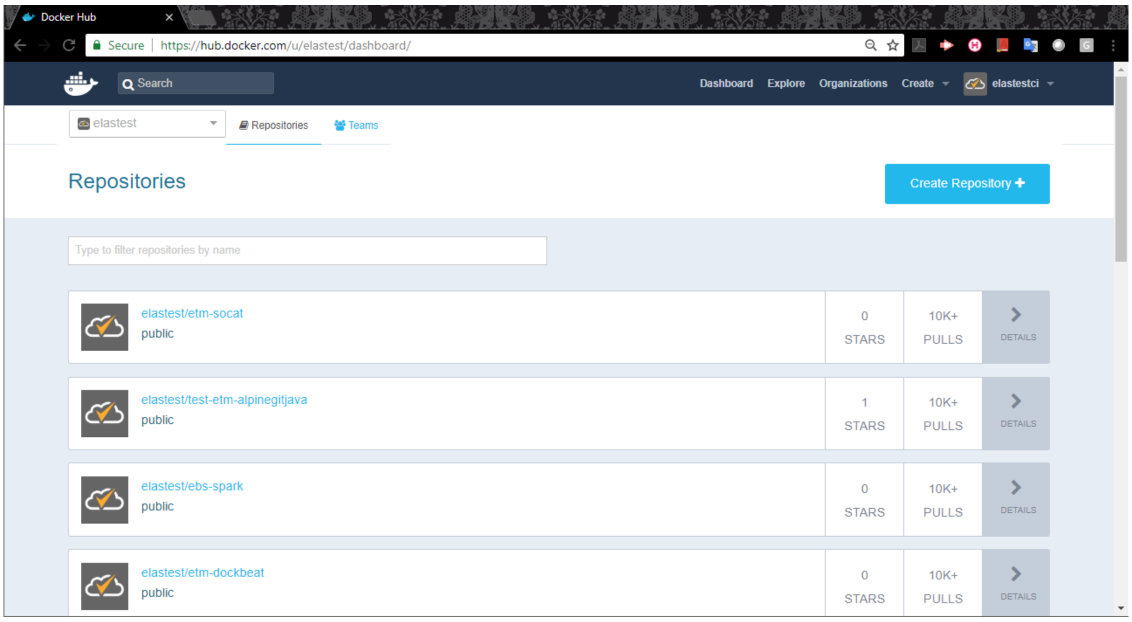
Both Sonatype OSSHR and Nexus Repository Manager OSS are the same tool and have exactly the same Graphic User Interface (GUI) –*See Figure 4. Sonatype OSSRH. Public maven repository and Figure 5. Nexus Repository Manager (OSS)*-. The main difference is that Sonatype OSSHR is a provided service and it is hosted on the Sonatype premises –or cloud- so it is maintained by them; whereas the Nexus Repository Manager (OSS) is self-hosted and it is deployed on our Main Instance –See *Figure 1. Self-hosted services architecture* –.

3.2.1.5 *DockerHub*

DockerHub [2] is public Docker Registry with possibility to host public and private Images. For public Images it is completely free of charges.

This Registry is widespread within the community of developers, and many well-known OSS projects have there their official images some examples are: nginx, ubuntu, Apache Tomcat… DockerHub is really easy to work with as it is the predefined Registry while working with Docker. This will provide a good way to share the public images with the community.

All the images published here will be public and any user will be able to pull it but images would be only pushed from Jenkins jobs (See 4.6 Releasing).



elastest/etm-socat public	0 STARS	10K+ PULLS	DETAILS
elastest/test-etm-alpinegitjava public	1 STARS	10K+ PULLS	DETAILS
elastest/ebs-spark public	0 STARS	10K+ PULLS	DETAILS
elastest/etm-dockbeat public	0 STARS	10K+ PULLS	DETAILS

Figure 6. DockerHub ElasTest Dashboard

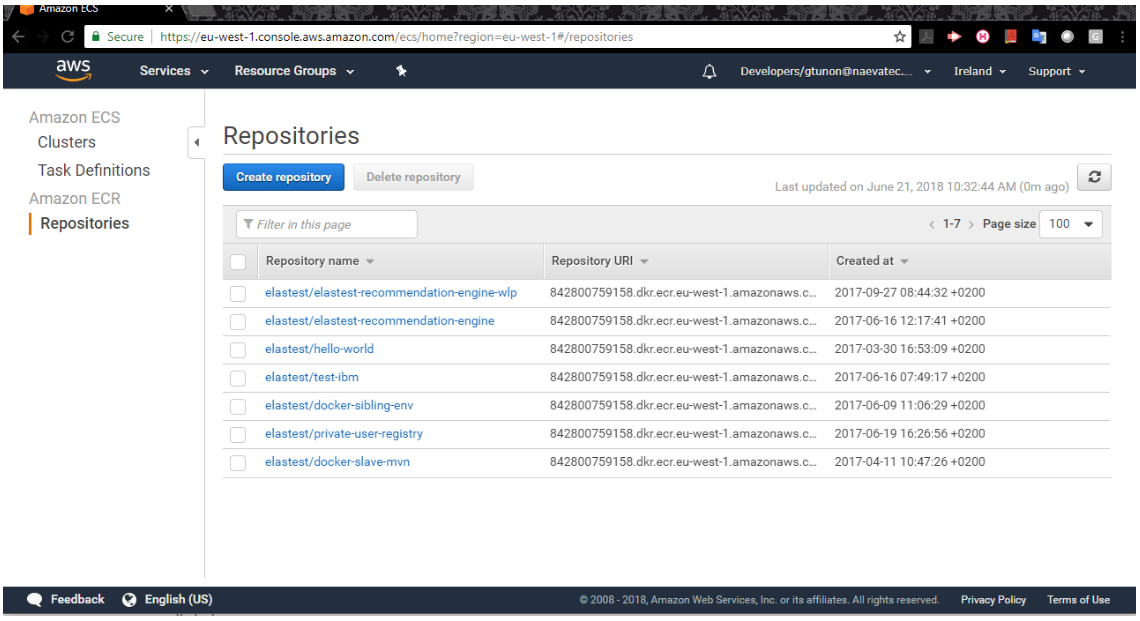
3.2.1.6 *Amazon EC2 Container Registry*

As explained in section *3.2.1.4 Nexus Repository Manager (OSS).* There is one component that is licensed under a proprietary license, but it should be available for the consortium. With this objective the Amazon EC2 Container Registry [7] has been chosen.

The access to the repository is completely private and it will be nominative even for read-only purposes, this will grant some control over the Docker images that are shared there. The access to the repository will be solicited with the User-registry application that will automatically check the access with the user GitHub account. Any user without a GitHub account or not member of the ElasTest organization in Jenkins will not be allowed to access to the repositories.

Amazon EC2 Container Registry transfers images over HTTPS and automatically encrypts your images at rest.

Full documentation on how to publish ElasTest private Docker Images have been provided to all the partners. And the procedure description can be found in section **4.6 Releasing.**



https://eu-west-1.console.aws.amazon.com/ecs/home?region=eu-west-1#/repositories

Amazon ECS
Clusters
Task Definitions
Amazon ECR
Repositories

Repositories

Last updated on June 21, 2018 10:32:44 AM (0m ago)

Repository name	Repository URI	Created at
elastest/elastest-recommendation-engine-wlp	842800759158.dkr.ecr.eu-west-1.amazonaws.c...	2017-09-27 08:44:32 +0200
elastest/elastest-recommendation-engine	842800759158.dkr.ecr.eu-west-1.amazonaws.c...	2017-06-16 12:17:41 +0200
elastest/hello-world	842800759158.dkr.ecr.eu-west-1.amazonaws.c...	2017-03-30 16:53:09 +0200
elastest/test-ibm	842800759158.dkr.ecr.eu-west-1.amazonaws.c...	2017-06-16 07:49:17 +0200
elastest/docker-sibling-env	842800759158.dkr.ecr.eu-west-1.amazonaws.c...	2017-06-09 11:06:29 +0200
elastest/private-user-registry	842800759158.dkr.ecr.eu-west-1.amazonaws.c...	2017-06-19 16:26:56 +0200
elastest/docker-slave-mvn	842800759158.dkr.ecr.eu-west-1.amazonaws.c...	2017-04-11 10:47:26 +0200

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Figure 7. Amazon ElasTest ECR

3.2.1.7 Private User Registry

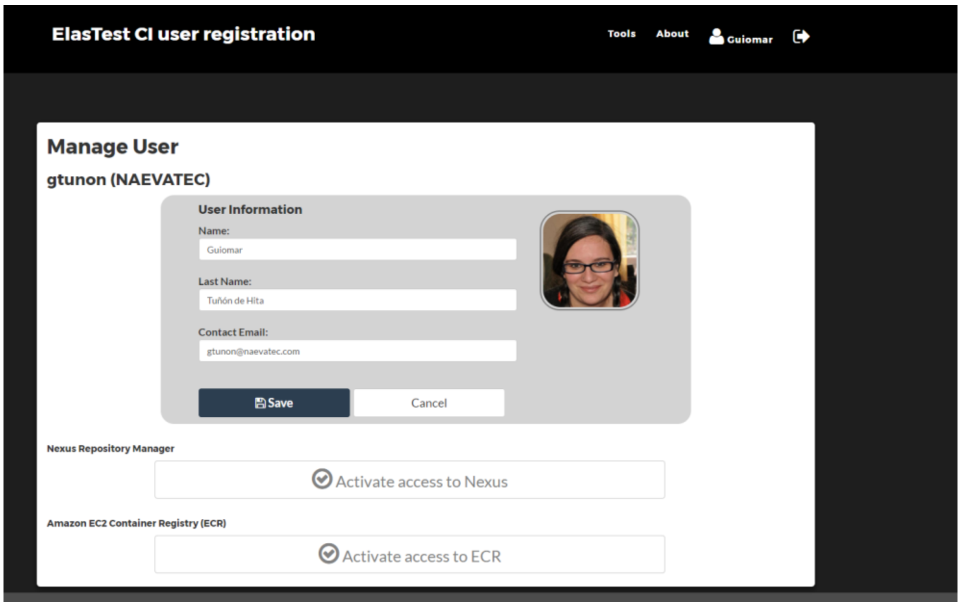
Some of the tools included in the environment don't provide GitHub Oauth2 integration (e.g. AWS ECR and Nexus Repository Manager OSS). In order to make the access unique, nominal and easy to manage and monitor, NaevaTec has developed a small application that manages the authorization of those tools. This application aims to ease the management of credentials for the private repositories, this way manual intervention isn't needed.

The Private User Registry is an application developed by NaevaTec, in the context of the ElasTest Project in order to enable developers from the consortium a way to consume private maven/gradle artifacts and private Docker images.

The application will verify the user GitHub account for checking his/her involvement in the project and his/her membership to one of the partners of the consortium. Once the user is validated she/he will be able to request credentials to access ElasTest private repositories.

The credentials provided by the application will only allow the user to read information from these repositories, as the publication of images and artifacts is done exclusively from the CI server (Jenkins). These credentials will enable developers to work in a local environment, for example to test the integration between components.

The application relays in the GitHub login, so any user who want to get credentials should be registered in GitHub.


ElasTest CI user registration

Tools About Guiomar

Manage User

gtunon (NAEVATEC)

User Information

Name: Guiomar

Last Name: Tuñón de Hita

Contact Email: gtunon@naevatec.com

Save Cancel

Nexus Repository Manager

Activate access to Nexus

Amazon EC2 Container Registry (ECR)

Activate access to ECR

Figure 8. Private User Registry - manage user access interface

Once registered the user will be able to manage his/her credentials (activate, deactivate, refresh) or edit/delete the user.

The information provided by the user is kept in a NaevaTec Database according to the following objectives:

- To keep track of the credentials used to access the repository.
- To enable the user to create, revoke and refresh credentials associated to his/her user name.

The database is completely private, only NaevaTec administrators related to the ElasTest project will have access to the database.

3.2.1.8 ElasTest

ElasTest is the tool developed within this project context, and it is used in two contexts on the CI / CV System:

- As object of the tests, the platform that should be tested before it can be released.
- As part of the tools for testing the SW, the platform that is used for testing.

Because we use this tools in two different contexts we have deploy the ElasTest platform twice, and we have aliased them as Nightly and Stable.

3.2.1.8.1 Nightly

ElasTest Nightly is a nightly updated live ElasTest instance, running with the aim of providing the latest ElasTest version of every component so end-to-end integrated tests can be run.

This ElasTest instance aims to provide all partners a place to test their own components on a production-like environment. Partners can access the ElasTest Nightly UI to do manual testing, check the look and feel, and of course run automated tests with Jenkins jobs.

ElasTest Nightly has been deployed on AWS following the specification shared in the ElasTest Community Web [11] validating in this way the correctness of the documentation.

3.2.1.8.2 ElasTest Stable

Since March 2018 we have also a stable version of ElasTest running that is being used for testing the ElasTest Nightly with ElasTest. This is a milestone – MS5 – set out in the DoA by end 2018, and the tasks to reach this milestone is in progress.

3.2.2 Add-ons and auxiliary tools.

Name	Type	License	Self-hosted	Access	Description
Codecov [9]	Cobertura reports analyser	OSS (Apache 2.2)	No	Public	Codecov provides highly integrated tools to group, merge, archive and compare coverage reports. Whether your team is comparing changes in a pull request or reviewing a single commit, Codecov will improve the code review workflow and quality.
SonarCloud [62]	Code review tool	OSS (LGPL-3.0)	No	Public	Analyse the quality of your source code to detect bugs, vulnerabilities and code smells throughout the development process.
ElasTest Jenkins Library [59]	Jenkins library for manage ElasTest	OSS (Eclipse)	--	Public	Developed groovy library to be used within Jenkins to help to launch ElasTest and manage ElasTest nodes. Developed in the context of the ElasTest project.

Table 2. CI environment auxiliary tools and add-ons.

3.2.2.1 Codecov.

3.2.2.1.1 What is code coverage?

Code coverage [61] is a measurement used to express which lines of code were executed by a test suite. We use three primary terms to describe each line executed:

- **hit** indicates that the source code was executed by the test suite.
- **partial** indicates that the source code was not fully executed by the test suite; there are remaining branches that were not executed.
- **miss** indicates that the source code was not executed by the test suite.

Coverage is the ratio of **hits / (sum of hit + partial + miss)**.

Phrased simply, code coverage provides a visual measurement of what source code is being executed by a test suite. This information indicates to the software developer where they should write new tests in the effort to achieve higher coverage.

Testing source code helps to prevent bugs and syntax errors by executing each line with a known variable and cross-checking it with an expected output.

3.2.2.1.2 What does Codecov bring to the table?

Code coverage tools incentivize developers to write tests and increase coverage. During the process of writing tests, the developer may discover new bugs or syntax issues in the source code that are important to resolve before distributing the application.

Codecov [9] takes coverage to the next level. Unlike open source and paid products, Codecov focuses on integration and promoting healthy pull requests. Codecov delivers -or "injects"- coverage metrics directly into the modern workflow to promote more code coverage, especially in pull requests where new features and bug fixes commonly occur.

3.2.2.2 SonarCloud.

SonarCloud [62] is a service operated by SonarSource [65], the company that develops and promotes open-source code quality products SonarQube [63] and SonarLint [64].

SonarQube provides the capability to not only show health of an application but also to highlight issues newly introduced. With a Quality Gate in place, you can fix the leak and therefore improve code quality systematically.

SonarLint gives you feedback on new bugs and quality issues injected into your code.

At the moment just, some components use SonarCloud to assure its quality and a proper procedure is expected to be designed so more of ElasTest can make use of the features it provides.

3.2.2.3 ElasTest Jenkins Library.

The ElasTest-library has been developed by NaevaTec in order to ease the integrated end-to-end tests of every component on Jenkins.

This library has 3 main objectives:

- Encapsulate all the main stages related to the ElasTest platform lifecycle management such as commissioning and decommissioning of ElasTest.
- Configure the type of ElasTest instance that should be run (authentication, mode, exclusive or shared instance, etc.) Making the jobs agnostic to the details. It is very useful on the current stage of the project as the ElasTest platform is still changing quite regularly.
- Enable the jobs to be run against a clean exclusive ElasTest platform launched for the specific job, and to run the same job against the nightly ElasTest or other

ElasTest running on another server. This can be configured by the job itself but also it can be forced by the CI admin, without changing the job definition (*Jenkinsfile*).

The library is in constant change, adapting itself to the changes of the ElasTest platform and reducing the impact that those changes have on the components jobs. In addition, it has the ability to grow-up to satisfy new requirements from the components. It has been designed to be as backwards compatible as possible with previous versions of ElasTest platform and with previous versions of the library itself.

The library is available as an open source resource on the ElasTest GitHub [59]. The following methods are provided by the library in order to configure and customize the ElasTest platform that would be used on the end-to-end tests.

3.2.2.3.1 Configuration methods:

setVerbose(boolean)
Overrides the Verbose configuration parameter value (false) of the Library

setVersion(String)
Overrides the ElasTest Version configuration parameter value (“latest”) of the Library

setERE(String)
To select if and which version of the ERE component will be deployed and used on the running ElasTest instance

setContext(value)
Initializes the context that should be applied for the execution of certain steps and commands.

setMode(String)
Initialization of the parameter Lite just for personalization, for normal execution leave it empty. For lite execution initialize with ‘--lite’

setShared(boolean)
Overrides the ElasTest Shared configuration parameter value (false) of the Library

setAuthenticatedElastest(boolean)
Overrides the default ElasTest Authentication parameter configuration parameter value (false) of the Library. If true the launched ElasTest would be initialized with basic authentication.

setTestLink(boolean)

Overrides the default ElasTest TestLink parameter configuration parameter value (false) of the Library. The TestLink component will be launched on the new instance.

3.2.2.3.2 Support functionalities for end-to-end testing:

String getIp()
Executed after the ElasTest start returns the IP address where the main component (ETM) is listening

String getPort()
Executed after the ElasTest start returns the port where the main component (ETM) is listening

String getEtmUrl()
Executed after the ElasTest start returns the full http connection URL where the ETM is listening (Example: http://172.0.18.10:8091)

Object getElastestMethods()
Returns an object that can execute the methods implemented by the library in the user defined stages.

String getElasTestUser()
Returns the user for the ElasTest Basic Authentication if the ElasTest is authenticated

String getElasTestPassword()
Returns the password for the ElasTest Basic Authentication if the ElasTest is authenticated

3.2.2.3.3 ElasTest management functionalities:

boolean startElastest()
Executes the ElasTest platform with the options configured in the library (version, lite)

boolean elastestIsRunning()
Return (in the moment) while ElasTest is already running. Checking if ElasTest Platform container is running and if ETM is running too.

boolean waitElasTest()
Waits up to 900s to ElasTest to start up. Returns true if it has started, and false otherwise.

boolean elastestIsStuck()

This is a variation of the waitElasTest that detects if any the platform component has been launched but the rest of the platform hasn't.

connect2ElastestNetwork()
After the ElasTest platform is launched as a Docker container, if this method is executed inside another container the method adds it with the ElasTest network.

stopElastest()
Stop the ElasTest platform and all the components.

getApi()
After the ElasTest Platform is started, this method initializes the connection information for the ElasTest

pullERE(optional version)
In case the ERE is expected to be launched the pullERE method will pull the defined version of the component if selected and the latest if void.

pullERE(optional version)
In case the ERE is expected to be launched the pullERE method will pull the defined version of the component if selected and the latest if void.

testRemoteElastest()
If the library is configured to connect with an external ElasTest (nightly mainly) the library would try to connect with it instead of "startElastest".

3.2.2.3.4 Pipeline encapsulation:

pipeline(body)
This method receives as parameter the user defined stages of the pipeline and encapsulates them between the necessary logic to select the kind of ElasTest execution and the management logic of the ElasTest. Resulting in a more complete pipeline with the start ElasTest and release ElasTest stages added to the user defined stages.

3.3 Security and User Access

3.3.1 Security

The CI environment has been designed to grant a great level of security by applying the following rules:

- **Minimize doors to the environment:**

The accesses to the instances where the tools are running are only accessible from NaevaTec VPN. By default, all the ports are closed for inbound connections, and only SSH port is open for NaevaTec VPN and it is secured by SSH2 and private key, with no user/password enabled. Ports 80 and 447 for the world. All the connections through the port 80 (HTTP) are redirected to port 447 (HTTPS).

- **Nominative access to resources:**
  All the not public resources are behind a log-in process. All the log-ins are nominative. And the accesses to the resources are logged and can be audited in case of secret revelations.

3.3.2 User Access

The CI environment and procedures allows all the members of the consortium to access the whole set of tools in a secured way with different access profiles, which depend on the roles they have within the project.

We have designed a GitHub Oauth2 based centralized login for most of the tools access. The groups of access are mainly maintained on the GitHub so the same access to the code management is replicated to most of the CI environment tools.

Each user will be probably assigned to more than one group, and that will enable different scopes within the CI environment.

Two groups for each Partner:

<u><Partner>-admin</u>

- **Users:** Only selected users from the partner designed to admin CI reserved scope.
- **CI functionalities:** Full access to functionalities within the partner reserved scope.

<u><Partner></u>

- **Users:** All the users from the partner that will be using the CI.
- **CI functionalities:** <Partner>-admin will enable the features within its reserved scope that these users can use.

Two groups for each Component:

<u><Component>-admin</u>

- **Users:** Only selected users from the partners working on the component, designed to admin CI reserved scope. Most probably users from the partner who is leading the WP or the related task.
- **CI functionalities:** Full access to functionalities within URJC reserves scope.

<u><Component></u>

- **Users:** All the users from the group of partners working on this component that will be using the CI.

- **CI functionalities:** <Component>-admin will enable the features within its reserved scope that these users can use.

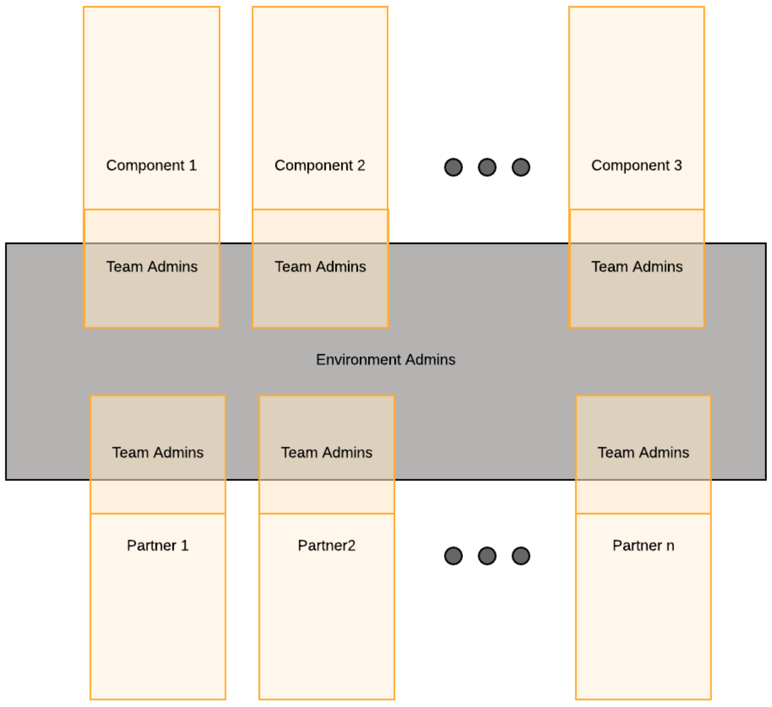


Figure 9. Authorization management: Teams

As presented in the Figure 9. We have defined Teams for each Partner and each component.

- Any consortium user will be part of its company team and also part of any of the teams where she/he is collaborating.
- Each organization can name any number of administrators of his/her group. And they will have administrator rights over that team.
- Each component leader will be administrator of the component team they are leading. And the leader can grant any user administrator rights over her/his managed group.
- Then there is a nominative group (with no associated team on GitHub) that have administrative access to all the teams, and to the GitHub organization.
- In addition to the GitHub Teams, some users would be selected as administrators of selected tools with independence of the partner and components he/she belongs to.

The rest of the applications in the environment will rely on the GitHub OAuth2 login and retrieve the user teams to grant the appropriate access to the tool to each user. By default, Partners teams are used for granting access to the tools and then components to add the appropriate level of access to the user.

The user access configuration allows to open the contribution to the project to people outside the consortium while maintaining the tools and the access to the sensitive tools and applications restricted to the Consortium.

The following figure shows a diagram of a Jenkins user login.

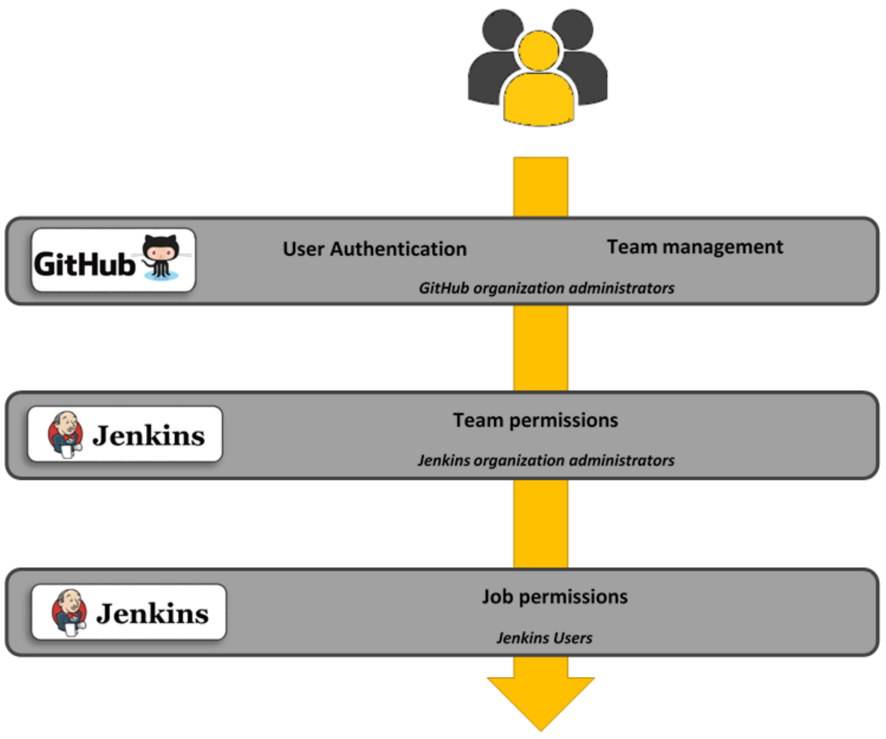


Figure 10. Jenkins user login

The user would be required to give each of the linked applications read permissions over his/her GitHub account. Those permissions can be revoked from her/his GitHub account at any moment.

3.4 Maintenance

The CI environment is regularly updated for maintaining the set of tools in a stable, secure and updated state.

The maintenance has been scheduled to be carried every three months with a full upgrade to the latest stable version of each of the hosted tools. Also, Exceptional maintenances have been taken into account for critical or important bugs or security issues. [See 4.7 Maintenance procedures]. Each of the actuations held on the environment are documented and these documents are kept for future reference and available for all the partners to read.

At the moment the following actuations have been run/ scheduled in the environment:

Date	Name	Cause	Status	Description
03/07/2017	20170703 - Maintenance Window	Scheduled	Done	OS, Jenkins, Nginx and Docker updates.
02/10/2017	20171002 - Maintenance Window	Scheduled	Done	OS, Jenkins, Nginx, Docker, Docker

				Compose, aws-cli updates.
20/11/2017	20171120 - Maintenance Window - Exceptional	Security	Done	OS, Jenkins updates.
22/12/2017	20171222 - Maintenance Window - Exceptional	Security	Done	OS, Jenkins updates.
16/01/2018	20180116 - Maintenance Window	Scheduled	Done with issues	OS, Jenkins, Nginx, Docker, Docker Compose, aws-cli updates.
26/02/2018	20180226 - Maintenance Window - Exceptional	Security	Done	OS, Jenkins updates.
02/04/2018	20180402 - Maintenance Window	Scheduled	Done	OS, Jenkins, Nginx, Docker, Docker Compose, aws-cli updates.
02/07/2018	20180702 - Maintenance Window	Scheduled	Delayed	OS, Jenkins, Nginx, Docker, Docker Compose, aws-cli updates.
01/10/2018	20181001 - Maintenance Window	Scheduled	Scheduled	OS, Jenkins, Nginx, Docker, Docker Compose, aws-cli updates.
10/01/2019	20190110 - Maintenance Window	Scheduled	Scheduled	OS, Jenkins, Nginx, Docker, Docker Compose, aws-cli updates.
01/04/2019	20190401 - Maintenance Window	Scheduled	Scheduled	OS, Jenkins, Nginx, Docker, Docker Compose, aws-cli updates.
01/07/2019	20190701 - Maintenance Window	Scheduled	Scheduled	OS, Jenkins, Nginx, Docker, Docker Compose, aws-cli updates.
07/10/2019	20191007 - Maintenance Window	Scheduled	Scheduled	OS, Jenkins, Nginx, Docker, Docker Compose, aws-cli updates.

Table 3. Maintenance schedule.

4 Methodology and Procedures

The following section describes all the procedures defined to interact with the environment and tools and maintaining them.

4.1 Methodology and Procedures

4.1.1 Basic Rules and Best practices.

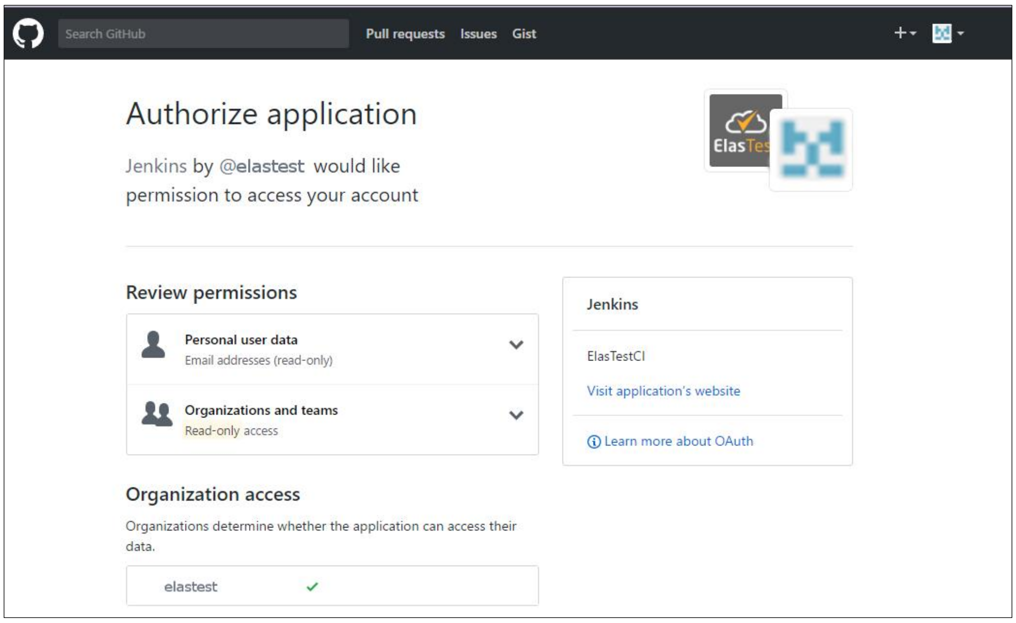
The following rules apply to all the CI environment users:

- Actively protect her/his access data to private repositories
- Not sharing his/her access data with any other person, especially with those external to the consortium.
- Communicate to the administrators any suspicion of misbehaviour or data access loss.
- Grant the SW published by Jenkins under her/his control to behave as expected.
- Not to publish any "dangerous" or "malicious" artifact.

4.1.2 Jenkins: Login

Users within the ElasTest Organization in GitHub that are members of any of the partner's teams would be granted access. Anybody inside the ElasTest Organization that isn't member at least of one of the partners' team won't be able to access.

1. Go to https://ci.elastest.io/jenkins
2. Authorize the application in your GitHub Account



Authorize application

Jenkins by @elastest would like permission to access your account

Review permissions

Personal user data
Email addresses (read-only)

Organizations and teams
Read-only access

Organization access

Organizations determine whether the application can access their data.

elastest ✓

Jenkins

ElasTestCI

Visit application's website

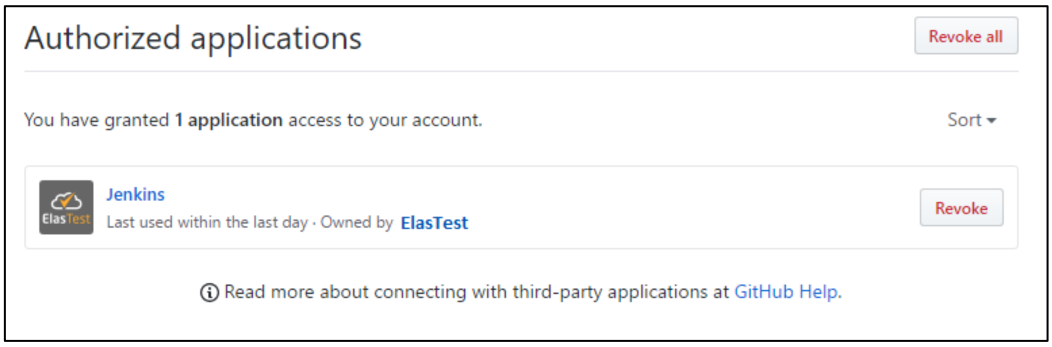
Learn more about OAuth

Figure 11. Authorize Jenkins GitHub

Users can revoke the authorization at any moment:

1. Go to https://github.com/settings/applications

2. Select Jenkins and press revoke:


Authorized applications

Revoke all

You have granted **1 application** access to your account.

Sort

Jenkins
Last used within the last day · Owned by **ElasTest**

Revoke

Read more about connecting with third-party applications at GitHub Help.

Figure 12. Revoke Jenkins GitHub

4.1.3 Jenkins: Tagging slaves and jobs

Whenever a slave is registered Jenkins offers the possibility of setting a set of labels/tags that defines the slave. So, jobs can restrict their execution to a specific slave or set of slaves that provide them with the necessary environment.

As it is a good practice to use the tags "wisely". The following rules are recommended for tagging slaves and jobs.

4.1.3.1 Slaves

All the slaves should be tagged so they will be assigned jobs that they are capable of executing. At least, there should be one tag for each of the following:

- **OS:** general, distribution, distro+version (e.g.: Linux ubuntu14.04)
- **Tools /SW:** one tag for each tool/SW installed on the AMI with version if it applies. (e.g.: java8, mvn3.3.9)

Also, when a slave is too sensitive or it is too specific for a job, it can be tagged with a name or token and the jobs can configure the aforementioned tag so they will be executed in that slave specifically.

4.1.3.2 Jobs

In order to launch the job on the correct Instance the Flag *"Restrict where you can run this project"* should be checked. This will enable a field where tags can be used in logical predicates to set the kind of instance this job needs to run.

The following rules apply to compose the predicate:

- Set the OS, as restrictive as it is necessary. Examples:
  - *Linux* - no specific distribution is necessary
  - *ubuntu14.04* - this specific distribution is needed.
  - *Linux && !ubuntu* - this job needs a Linux but it cannot be Ubuntu, but all other distributions will be valid...

- Set of tools needed. Examples
    - *(OS predicate) && (maven && java8)*
- Specific label of a node.

As the predicate is written the user will see the number of slaves or cloud resources that satisfies the predicate in that moment. If there isn't any slaves or cloud resources that satisfies the predicate and there isn't any restriction that can be overseen system administration will look for a solution – probably create and define a new type of instance that would act as a Jenkins Slave –.

4.1.4 Private User Registry: Login and Register

In the main page the user should click on the Sign in with GitHub button located on the top right corner.

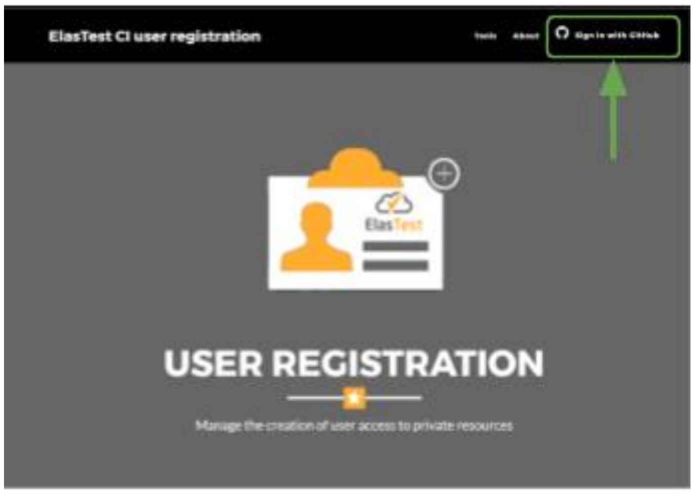
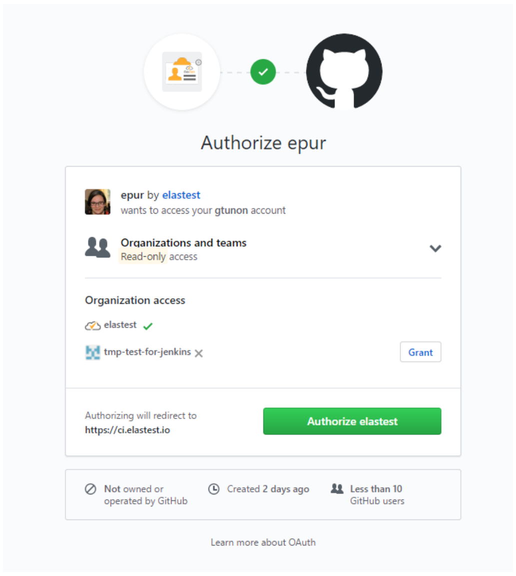

ElasTest CI user registration
Tools
About
Sign in with GitHub
ElasTest
USER REGISTRATION
Manage the creation of user access to private resources

Figure 13. User Registration home

Along the first attempt to access to this application, GitHub will request the user permission to enable ElasTest User Registry application in order to gather the user data from his GitHub account. The mechanism requests access to the user public information and his membership within organizations and teams (read-only). If the user is member of more than one organization she/he can chose only the ElasTest organization, as the application won't be reading other information.


Authorize epur

epur by elastest
wants to access your gtunon account

Organizations and teams
Read-only access

Organization access
elastest ✓
tmp-test-for-jenkins ✗ Grant

Authorizing will redirect to
https://ci.elastest.io
Authorize elastest

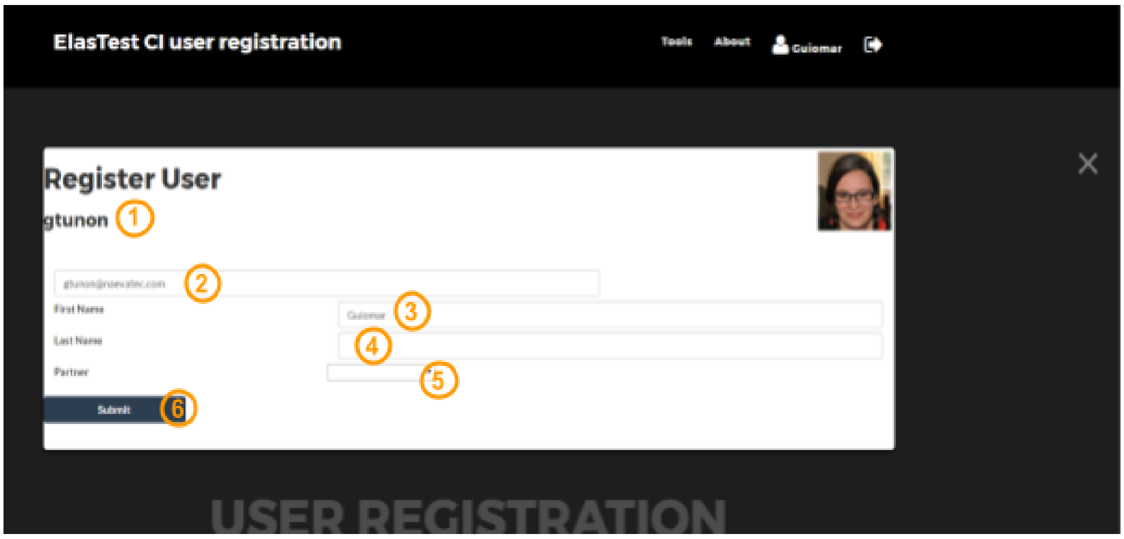
Not owned or operated by GitHub
Created 2 days ago
Less than 10 GitHub users

Learn more about OAuth

Figure 14. User Registration. GitHub Authorization

If the user is member of the ElasTest organization and belongs to at least one of the Consortium Teams (ATOS, CNR, FOKUS, IBM, IMDEA, NaevaTec, REL, TUB 5G, TUB IIoT, URJC, ZHAW), he/she will be able to register and then generate his/her credentials.

The first time the user is logged in, a form will pop-up. Filling this form will grant the administrators to gather the information needed to provide the user with the requested credentials.


ElasTest CI user registration
Tools About Guiomar

Register User
gtunon 1

gtunon@naevatec.com 2
First Name: Guiomar 3
Last Name: 4
Partner: 5
Submit 6

USER REGISTRATION

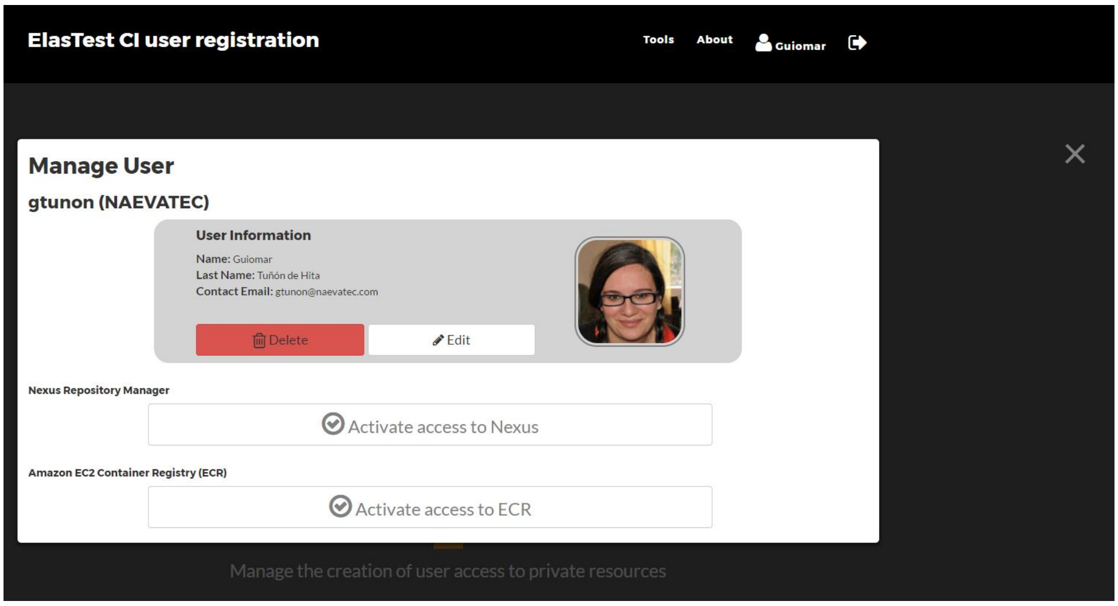
Figure 15. User Registration form

1. **User Name.** Retrieved from GitHub and it can't be modified from this form.
2. **Email.** Retrieved from GitHub and could be overwritten with a different email address provided by the user. This information will be used to send the user credentials.
3. **First Name.** This field is retrieved from GitHub and could be overwritten from this form and should contain the user's REAL first Name.
4. **Last Name.** Not pre-initialized and must be filled by the user's REAL last name.
5. **Partner.** To which partner does the user belongs to. (Can't be changed once submitted).

6. **Submit.** To proceed with the registration

4.1.5 Private User Registry: Manage User,

Once registered, the user will be able to manage his/her information, and his/her credentials. Also, the user will be able to delete him/her-self from the application



Manage User

gtunon (NAEVATEC)

User Information

Name: Guiomar

Last Name: Tuñón de Hita

Contact Email: gtunon@naevatec.com

Nexus Repository Manager

Amazon EC2 Container Registry (ECR)

Manage the creation of user access to private resources

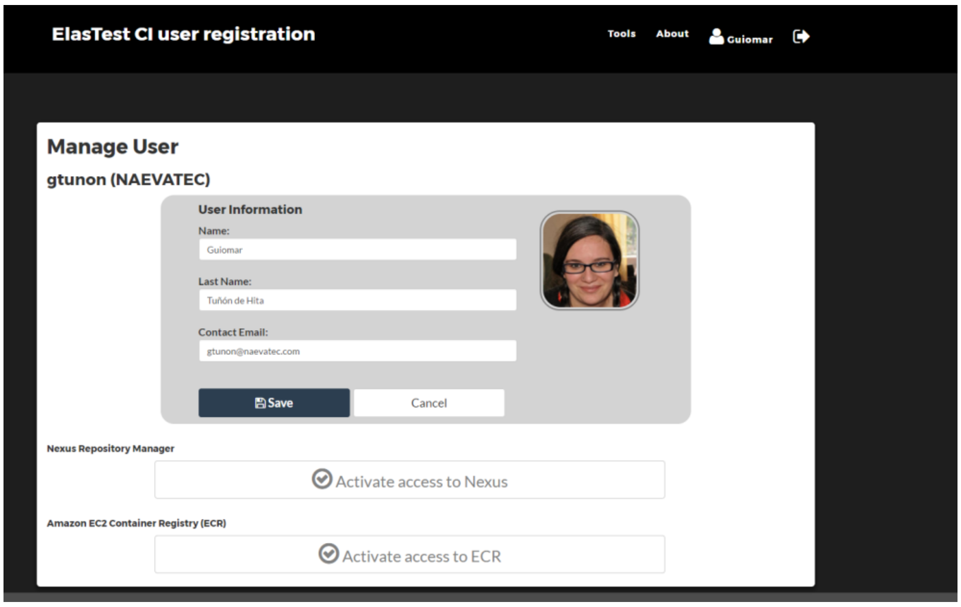
Figure 16. User Registration. User Management

4.1.5.1 Delete User.

Deleting a user will prompt a confirmation dialog. Once it is confirmed the following actions will take place:

- If the user has active credentials they will be revoked.
- The data kept in the application database will be deleted, no information will be kept (except the logs of his/her activity within the tools).

4.1.5.2 *Edit User Information.*



Manage User
gtunon (NAEVATEC)

User Information
Name: Guiomar
Last Name: Tuñón de Hita
Contact Email: gtunon@naevatec.com

Nexus Repository Manager

Amazon EC2 Container Registry (ECR)

Figure 17. User Registration. User Information edition

The fields that can be edited are:

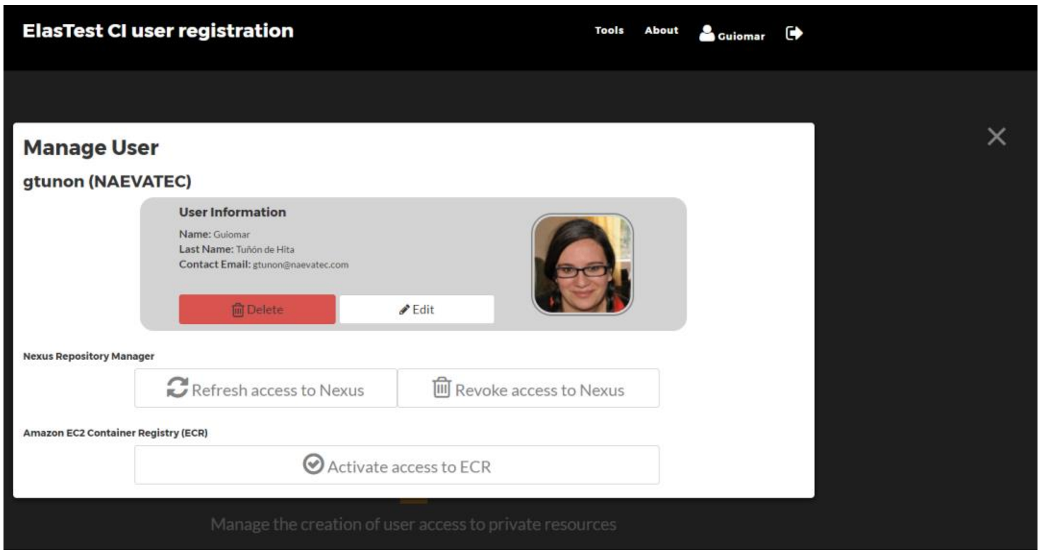
- Name
- Last Name
- Email.

4.1.5.3 *Requesting access to Nexus Repository.*

The nexus repository is used as a private software Artifact repository where those components that shouldn't be published outside the consortium should be published.

Activate the access to Nexus will create a user inside the nexus with the same username as GitHub, and a secure password will be sent to the provided email.

Once the user has an active credential for the nexus the options to refresh and delete the credentials will be enabled.


ElasTest CI user registration

Tools About Guiomar

Manage User

gtunon (NAEVATEC)

User Information

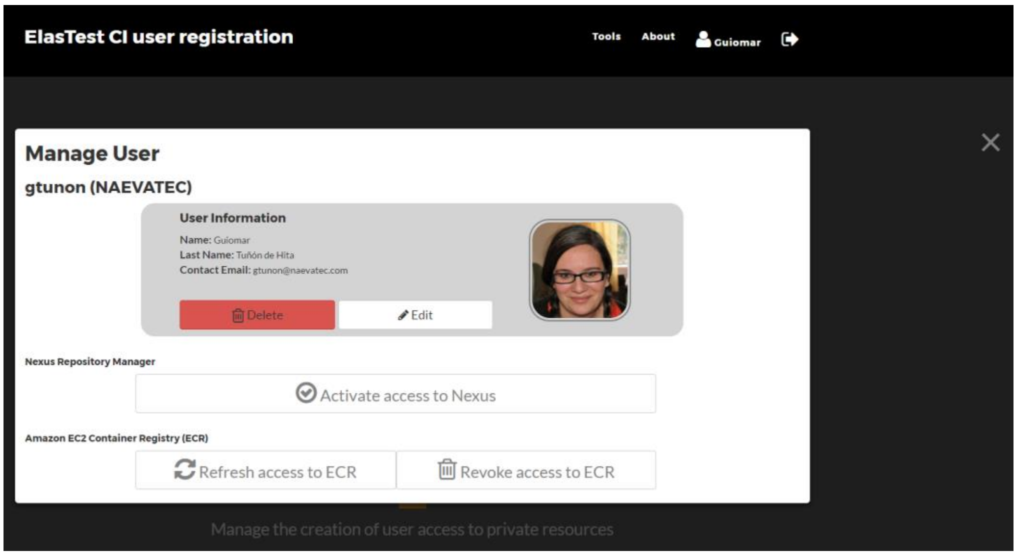
Name: Guiomar
Last Name: Tuñón de Hita
Contact Email: gtunon@naevatec.com

Figure 18. User Registration. Access to Nexus activated

4.1.5.4 *Requesting access to ECR*

ECR is a private Docker container registry where those images that shouldn't be published outside the consortium should be published.

Activate the access to Nexus will create the credentials to access the Docker registry. The credentials will be sent to the provided email. Once the user has an active credential for the ECR the options to refresh and delete the credentials will be enabled.


ElasTest CI user registration

Tools About Guiomar

Manage User

gtunon (NAEVATEC)

User Information

Name: Guiomar
Last Name: Tuñón de Hita
Contact Email: gtunon@naevatec.com

Figure 19. User Registration. Access to AWS ECR activated

4.1.5.5 *Refresh access to Nexus*

Refreshing the access to Nexus will revoke the previous credentials and a new password will be sent to the user.

4.1.5.6 Refresh access to ECR

Refreshing the access to ECR will revoke the previous credentials and new credentials will be sent to the user.

4.1.5.7 Revoke access to Nexus

The revocation of the access to Nexus will disable the user in the Nexus. But it won't delete the user so we can keep track of the previous actions done by the user.

4.1.5.8 Revoke access to AWS ECR

The revocation of the access to ECR will disable the user in the ECR and the credentials provided previously will be revoked. But it won't delete the user so we can keep track of the previous actions done by the user.

4.2 Development environment configuration for AWS ECR

The following procedure should be followed in order to access any of the Docker images hosted in the private AWS ECR.

1. Register for an AWS ECR private registry account. (See section 4.1.4 Private User Registry: Login and Register). Once you have registered for ECR you will receive credentials by email
2. Install AWS CLI tools for your platform using the AWS CLI Installation guide provided by Amazon.
3. Configure AWS by running `aws configure` with the following parameters:
   - AWS Access Key ID - *provided in the email received in step 2.*
   - AWS Secret Access Key - *provided in the email received in step 2.*
   - Default region name - eu-west-1
   - Default output format - json
4. Login to the private ECR registry by running `$(aws ecr get-login --no-include-email)`

4.3 Jenkins Jobs Naming

As a component owner, you can have as many jobs in Jenkins as you want. There is a Jenkins folder per each component. The more common jobs are the following:

- **Compile jobs:** These jobs are executed in every commit that lands to master branch (directly or after a merge or rebase). They execute unitary and integration tests and if there are no errors, it publishes development artifacts to binary repositories. In the case of platform components, they create and publish the **main component image** in DockerHub. The names of these jobs are simply the acronym of the component. For example: EDM, EPM, ETM, among others.

- **End to end test jobs:** These jobs execute end-to-end tests against Docker images generated in the compile job. They are executed also in every commit, after the compile jobs. In the future, if these tests spend too much time to execute (and use too much computing resources), it will be considered executing them once a day, at night or even with fewer periodicity -every two days or even weekly-. The names of these jobs are the acronym of the component followed by “e2e”. For example: edm-e2e, epm-e2e, etm-e2e, etc.

- **Docker images jobs:** As complementary component images and test images change less often than main component images, it is not necessary to recreate them in every commit of the main repository. For that reason, it can be useful to have Jenkins jobs tailored specifically to the creation of **complementary component images and test images**. To reduce the number of jobs to maintain, it is ok to have only one job per component that creates all complementary component images and test images of that component. But it will be also accepted to have a job per complementary in case more fine-grained control over image creation is needed. Those jobs can be executed manually by the component developer or can be executed on every commit that affects to files on that image. A crucial aspect is to follow the name format for the jobs. If only a job for all images exists, the name of that job is the acronym followed by “-comp-images”. For example: edm-comp-images, epm-comp-images, etc.… In other case each subcomponent job should be named according to the image name, for example: eim-logstash, edm-mysql, etc.

In summary, we will have the following Jenkins jobs in every component folder:

1. Compile job: <component_acronym>
2. End to end job: <component_acronym>-e2e
3. Docker images job:
    - Multiple images: <component_acronym>-comp-images
    - One image: <component_acronym>-<complement>

4.4 Development

- <u>Source code management:</u>

All the ElasTest repositories must be contained in the ElasTest GitHub organization: https://github.com/elastest. Inside this organization, there are two types of repositories:

- Internal repositories, i.e. repositories created directly in the ElasTest organization.
- Forked repositories, i.e. repositories originally created in a different organization, but forked inside the ElasTest GitHub organization.

Each of the repositories would be managed and administered by the component leader. Leaders have the freedom of add people to the contributors and select the best branching and merging directives that would fit better for the managed component.

As defined in the Grant Agreement document, the license of the public repositories should be one of the following: LGPL, Apache 2, or MIT. For the sake of simplicity, we selected Apache 2.0 [10] license for every public software artifact of ElasTest, except those that pertain to the background of a partner.

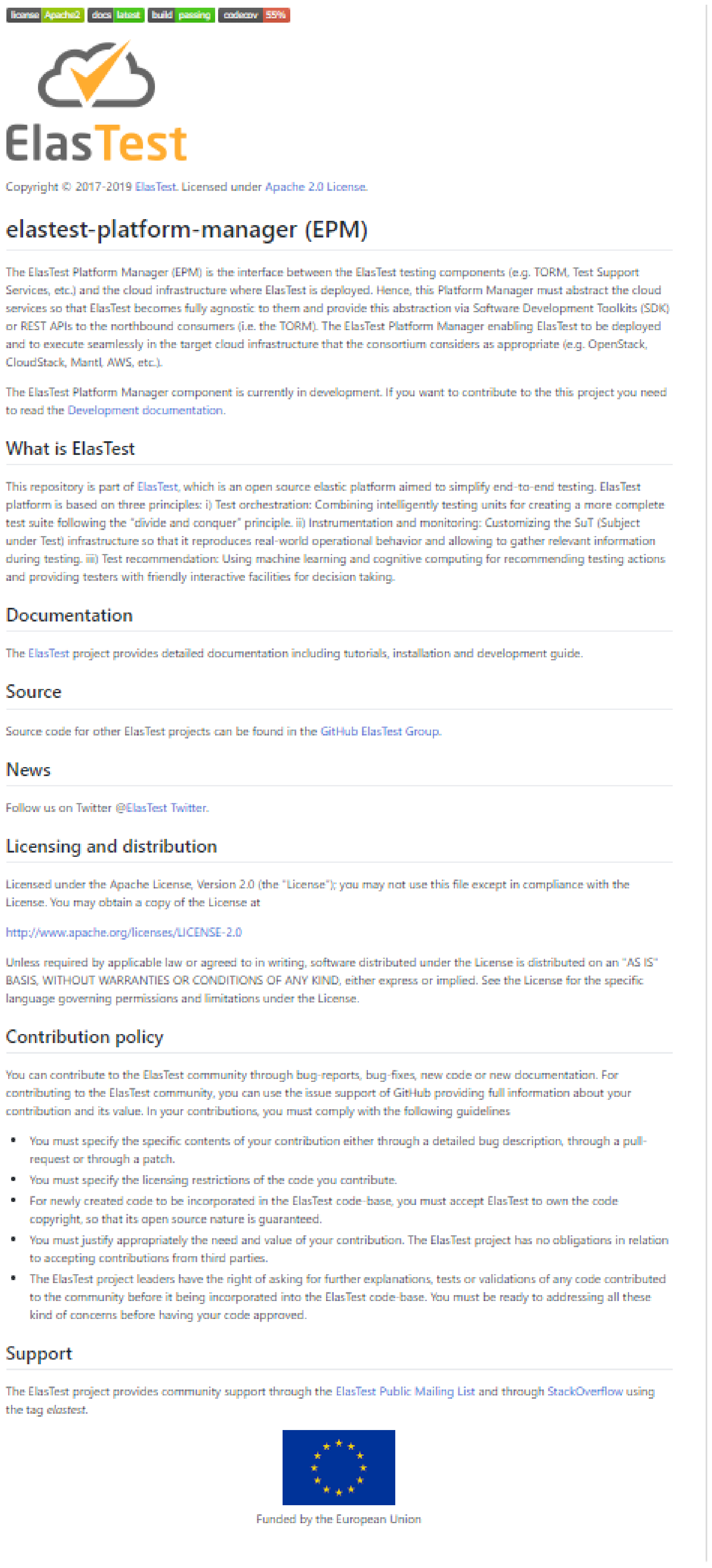
Each repository should contain a README file, written in Markdown format, in the root with the default template (See [27]) defined for the project. At least the following information should be present in each README file:

- Badges
- Header (ElasTest logo, copyright, license)
- Name of repository and description
- Brief description of ElasTest
- License and distribution information
- Contribution policy
- Support

In addition to the README file each repository that belongs to an ElasTest component should have a folder for the user documentation.

The badges expected to be found in each of the repositories are the following:

- License.
- Documentation.
- Build Status.
- Code Coverage.
- Quality.



license Apache2 docs latest build passing codecov 55%

ElasTest

Copyright © 2017-2019 ElasTest. Licensed under Apache 2.0 License.

elastest-platform-manager (EPM)

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

The ElasTest Platform Manager component is currently in development. If you want to contribute to the this project you need to read the Development documentation.

What is ElasTest

This repository is part of ElasTest, which is an open source elastic platform aimed to simplify end-to-end testing. ElasTest platform is based on three principles: i) Test orchestration: Combining intelligently testing units for creating a more complete test suite following the "divide and conquer" principle. ii) Instrumentation and monitoring: Customizing the SuT (Subject under Test) infrastructure so that it reproduces real-world operational behavior and allowing to gather relevant information during testing. iii) Test recommendation: Using machine learning and cognitive computing for recommending testing actions and providing testers with friendly interactive facilities for decision taking.

Documentation

The ElasTest project provides detailed documentation including tutorials, installation and development guide.

Source

Source code for other ElasTest projects can be found in the GitHub ElasTest Group.

News

Follow us on Twitter @ElasTest Twitter.

Licensing and distribution

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Contribution policy

You can contribute to the ElasTest community through bug-reports, bug-fixes, new code or new documentation. For contributing to the ElasTest community, you can use the issue support of GitHub providing full information about your contribution and its value. In your contributions, you must comply with the following guidelines

- You must specify the specific contents of your contribution either through a detailed bug description, through a pull-request or through a patch.
- You must specify the licensing restrictions of the code you contribute.
- For newly created code to be incorporated in the ElasTest code-base, you must accept ElasTest to own the code copyright, so that its open source nature is guaranteed.
- You must justify appropriately the need and value of your contribution. The ElasTest project has no obligations in relation to accepting contributions from third parties.
- The ElasTest project leaders have the right of asking for further explanations, tests or validations of any code contributed to the community before it being incorporated into the ElasTest code-base. You must be ready to addressing all these kind of concerns before having your code approved.

Support

The ElasTest project provides community support through the ElasTest Public Mailing List and through StackOverflow using the tag *elastest*.

Funded by the European Union

Figure 20. Example of README.md

4.5 Testing

4.5.1 Unitary and integration (component)

All the components are expected to have a complete suite of unitary tests. The suite should reach about a 70% test coverage for mature components whether those components where research is more prominent are expected to have weaker test coverage at this point.

Test coverage changes rapidly with the different iterations on the development process but as an idea at the moment of writing this document the test coverage of each component is the following:

Component	Test Coverage
elastest-torm	43,71%
elastest-user-emulator-service	73,74%
elastest-monitoring-service	95,45%
elastest-instrumentation-manager	7,21%
elastest-monitoring-platform	69,91%
elastest-cost-engine	79,70%
elastest-platform-manager	55,18%
elastest-data-manager	90,91%
elastest-service-manager	84,64%
elastest-bigdata-service	92,31%
elastest-device-emulator-service	34,31%

Table 4. Code Coverage. Date: 21-06-2018

To report the test coverage, we have selected CodeCov.io tool that provides a good graphic interface for the public repositories of GitHub and provides a good insight of the improvements made with each commit to the test coverage.

In order to publish the test coverage into CodeCov.io the following steps should be as follow:

1) Generate the Cobertura reports in a Jenkins Job.
2) Use the provided Tokens (as Jenkins Constants) for the appropriate repository, to upload the reports to CodeCov.io
3) Add badge for code Coverage in the GitHub documentation.

Jenkins jobs examples for the configuration of Cobertura reports and codecov.io integration have been provided for the partners to follow. You can find more information in A3. Cobertura Reports and CodeCov.io integration procedure.

4.5.2 End-to-end tests.

We have devised an end-to-end test plan that have 3 main stages:

1. **Component end-to-end:** Components provide end-to-end tests that ensure the behaviour of the component and all the services that it makes use of. This kind of tests are usually held against the component API and can be launch against the component running as a part of the ElasTest platform, or against an isolated instance of the component if applies. This kind of test aren't applicable to all the components, see section 4.5.2.4.
2. **Platform end-to-end, traditional tools:** Most components have defined their own end-to-end tests that test their behaviour within the ElasTest platform through the GUI. These tests reproduce use cases that would make use of the component tested. The assertion clauses are focused on the component tested. Only components that have no GUI are excluded from these end-to-end tests. This way once run all the component end-to-end tests we have an idea of the actual behaviour of the whole platform. These tests are run by Jenkins jobs nightly.
3. **Platform end-to-end, ElasTest:** All the Jenkins jobs of the stage 2 are being converted on TJobs of for ElasTest. These TJobs would be run on a Stable ElasTest. This is a project milestone planned for the end of 2018.

4.5.2.2 ElasTest-library

In order to use the described library in section 3.2.2.3 a procedure have been created, in order to simplify and unify the style of the end-to-end tests jobs that run over a full ElasTest.

The code and basic documentation can be found in the following repository: https://github.com/elastest/ci-elastest-jenkins-lib and can be used in Jenkins pipelines with the following reference: **`@Library('ciElastestConnect') _`**

The library has been developed with the basic functionality provided by the elastest/platform component, but other common useful functionalities can be added at any time.

- The library can be configured to launch different configurations of the ElasTest:
- Normal / experimental / experimental-lite
- One per job / one shared for all the jobs…
- Specific version of the ElasTest

4.5.2.2.1 Library initialization

First of all, the library should be instantiated. The name the library has been registered with in the Jenkins is `'ciElastestConnect'`

```
@Library('ciElastestConnect') _
```

After the declaration of the library it should be configured to behave as we expect. We have two kind of configuration parameters: mandatory and optional.

- **Mandatory:**
    - Context:

```
//initialization of the context for the library
elastest_lib.setContext(this)
```

- **Optional:**
    - Mode
        - Values: “normal” [default], "experimental", "experimental-lite"
    - Verbose

```
//uncomment this if you want to use a lite or augmented version of
elastest
//elastest_lib.setMode(“experimental”)
```

- Version

```
//uncomment this if you want a more detailed log
//elastest_lib.setVerbose(true)
```

- Shared
    - The shared parameter is recommended to be used with the default value.

More configuration options are described in section *A5. ElasTest Jenkins Library API.*

Lastly in the library set up we need to instantiate an ElasTest object to be able to execute the methods provided by it within our stages.

```
//initialization of the runnable object to access elastest methods
//inside the pipeline
def elastest = elastest_lib.getElastestMethods()
```

4.5.2.2.2 End-to-end stages

In order to define our custom stages, it is important to notice that no node declaration is accepted (the library will take care of it).

Once marked that, the stages should be declared inside of the `elastest_lib.pipeline({...})` method. Example:

```
//definition of nodes is managed by the library. Just declare the
//steps of your pipeline inside the elastest_lib.pipeline({...})
elastest_lib.pipeline({
    stage('Conf tests')
        echo 'Test configuration'

    stage(‘E2E tests')
```

```
        echo 'Here your tests'
})
```

This definition of stages will lead to a pipeline like this one:

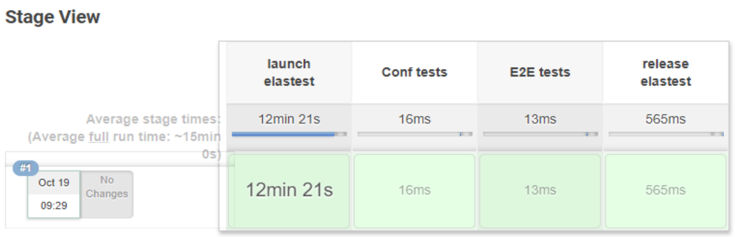


Figure 21. Jenkins ElasTest-library. Pipeline

Stages “launch elastest” and “release elastest” are added by the library.

Inside the user defined stages, the following rules apply:

- **Plugins and common stages:**
    - As the execution of the commands is done inside the library the context is different, and some objects or stages are unavailable. The invoked object or stage will fail with the next error:

```
java.lang.NullPointerException: Cannot invoke method image() on null object
        at org.codehaus.groovy.runtime.NullObject.invokeMethod(NullObject.java:91)
        at org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.call(PogoMetaClassSite.java:48)
        at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray.java:48)
        at org.codehaus.groovy.runtime.callsite.NullCallSite.call(NullCallSite.java:35)
        at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCall(CallSiteArray.java:48)
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:113)
```

    - This error can be solved by adding elastest.ctx to the object or the stage for example:

```
        docker.image('...') //fails
        elastest.ctx.docker.image('...')//works
```

- **Inside a custom container:**
    - When the job launches a custom container and when from inside that container, the job tries to connect with the ElasTest in any way curl, Selenium, etc. The launched ElasTest wouldn’t be reachable.
    - The solution is to execute inside the container and before trying to connect with the ElasTest, the following library method:

```
        elastest.connect2ElastestNetwork()
```

4.5.2.3 Jenkins end-to-end tests job structure

In order to compare the jobs execution and have a “meta-job” that can run parallel end-to-end test jobs and be able to compare stages the end-to-end job pipelines should be similar. The proposed stages are the following:

1. **"Launch elastest":** Library Stage. It will launch the ElasTest instance or check the nightly Instance.
2. **"Configuration:** User stage. Here the entire configuration previous to the tests execution will take place. From launching a specific container and connecting it to ElasTest, to load tests parameters.
3. **"E2E tests":** User stage. Here only the execution of the test should take place.
4. **"Results":** User stage. The results can be published here.
5. **"Release elastest":** Library Stage. It will release and delete the ElasTest Instance or doing nothing in case the shared library is used.

We recommend to add all the proposed stages and fit your steps in the proposed stages, without any addition or deletion, in case one stage is empty it should be filled with an *echo* step.

4.5.2.4 End-to-end tests per component

Each component can be tested as a black box, to assert the validation of the provided APIs, and expected behaviour. Each component can define its own method of validation and test execution scheduled. These tests aren't mandatory so they are a recommendation, as a first step towards the Integrated UI end-to-end tests.

4.5.2.4.1 ElasTest Big Data Service (EBS)

The ElasTest Big-Data Service consists of an Apache Spark [40] cluster, the necessary connectors for persistence services (EDM) and a REST API to provide detailed health information for the cluster.

The API tests are written in Python using the requests library and fully check the functionality of the cluster. An abstract explanation of the steps is given below:

1. Create a TJob (described below).
2. Execute the TJob
3. Probe for results.
4. Exit (successfully or unsuccessfully).

While the test performs a thorough check of ElasTest API functionality for interacting with EBS, the specific TJob is specifically created to perform an end-to-end test on the EBS service by going through the whole lifecycle of a Spark application. More specifically the job will:

1. Git clone a demo Spark application from ElasTest demo projects [41].
2. Build the application using maven.
3. Download a big text file [42] from the internet and save it to HDFS (EDM).
4. Submit the application to the cluster. This is a word-count application that will return the number of occurrences of every word on the specified file and save the results back in HDFS.
5. Pull the result file back out from HDFS.
6. Perform a printout on the result file.

4.5.2.4.2 ElasTest Cost Engine (ECE)

The ElasTest Cost Modelling Engine (ECE) has been implemented in Java, using Spring-boot [37] and Thymeleaf [38] templating framework. The unit tests are written as Junit 4 [33] tests and code coverage are assessed using Cobertura [34] reports. The project itself is developed as a Maven [35] project and the tests can be executed locally as well as on the ElasTest CI server [36]. The unit tests cover the following functionality in ECE:

- REST driver functions
- Controller functions which controls the UI rendering of cost analysis requests

In the next iteration of ECE, online accounting and billing mode will be added, and JUnit tests will be enhanced to cover the new features in order to maintain or increase the code coverage of the module.

4.5.2.4.3 ElasTest Data Manager (EDM)

The ElasTest Data Manager (EDM) is a bundled set of persistence services in a single component. All tests (with or without ElasTest GUI) must utilize other components in order to test EDM functionality, as most parts of EDM are not API nor GUI directly accessible, only via other services. The only components that can be directly end-to-end tested are cloudcmd (via ElasTest Platform UI) and Kibana (via ElasTest Platform UI).

4.5.2.4.3.1 Hadoop/Alluxio

Hadoop and Alluxio are tested via EBS end-to-end test *4.5.2.5.1 ElasTest Big Data Service (EBS)*. The job functionality described in the relevant section is specifically designed to perform different operations on Hadoop, in order to verify both components work.

4.5.2.4.4 ElasTest Device Emulator Service (EDS)

The goal of EDS end-to-end tests is to invoke the TSS EDS via ETM, use the facilities offered by EDS in particular the facilities offered by service eds-base to initiate sensors and actuators. Furthermore, the initiated sensors and actuators can be wired at the application level where the end-to-end test is written.

The tests are executed in the ElasTest CI server [36] and are defined by a Jenkinsfile [44].

The EDS provides a test [45] that uses the API facilities of the ETM to invoke and use eds-base in the steps mentioned below:

1. Open ElasTest using a HTTP request with authentication parameters that is available on the machine running Jenkins.
2. Create a new project with name EDSE2E via http request to ETM.
3. Send a request to the ElasTest Platform to facilitate new TJob with the following details:
    a. TJob name as demotjob.

   b. Commands required to run the end-to-end test.
   c. ESM string to indicate initiation of required TSS. In this case only component EDS.
4. TJob execution process -
   a. Wait until the TSS is available.
   b. After TSS is made available from ETM, execute the TJob until its completion.
   c. A successful exit of the test indicates the TJob was successful in execution else the TJob execution is marked as Fail.

4.5.2.4.4.1 Test Application for the EDS tests

The test application interacts with service eds-base offered by EDS to request components and wire them. The components offered at present are:

- Temperature sensor: This is a sensor which provides sensor values at a fixed interval and can be turned on or off.
- Simple actuator: This actuator receives a value at its input which it will forward to the output after a fixed delay. This actuator can be turned on or off.

The sequence of actions carried out by the test application are as follows:

1. Register the test application with eds-base.
2. Request for a temperature sensor.
3. Request for a simple actuator.
4. Turn on the temperature sensor and fix the reporting interval to 5 seconds.
5. Turn on the simple actuator and fix the reporting delay to 3 seconds.
6. Wire the output of the temperature sensor to the input of the simple actuator. Wire the simple actuator output to the test application.
7. Run this loop for 60 seconds. After this terminate the application with a success.

4.5.2.4.5 ElasTest Instrumentation Manager (EIM)

The ElasTest Instrumentation Manager (EIM) has been implemented in Java, as a JAX-RS RESTful API Service [47] using a MySQL database to store data (MongoDB is also supported). JUnit 4 [33] has been chosen as testing framework for the component. Tests suite is automatically executed by the ElasTest CI server [36], the tests are based on several topics in order to cover these topics:

- Data storage layer.
- Data model layer.
- REST API performance.

4.5.2.4.6 ElasTest Monitoring Platform (EMP)

The ElasTest Monitoring Platform (EMP) is written in Java using Spring framework [18] as a RESTful service. The unit tests are written as Junit 4 [33] tests and Cobertura [34] reports are used to assess the code coverage. The EMP uses Maven [35] as its build environment and tests can be executed locally as well as on the ElasTest CI server [36]. The unit tests in EMP cover various capabilities for correctness including (but not limited to) the following areas:

- Database interactions
- Time series data manipulations
- Kafka communications and topic management
- Controller functionalities
- API service
- Health check functionalities
- In memory cache behaviour

4.5.2.4.7 ElasTest Monitoring Service (EMS)

In order to test the ElasTest Monitoring Service (EMS) into the ElasTest Jenkins, a Jenkins file has been provided [26].

The test starts launching an instance of ElasTest and its URL retrieved in order to access the different functionalities exposed by the EMS. The test flow is specified in a bash script [28] and it realizes the following actions:

1. An EMS instance is started using the ESM REST API.
2. Once the EMS is up and running, its URL is retrieved through the ESP API.
3. An empty project then is created also the ESM REST API.
4. A SuT is created inside the project from the previous step. This SuT runs a dummy Docker [29] listing all the processes running on the container.
5. A TJob is created in the project and linked to the dummy SuT using also the ETM REST API. The T-job is defined as a metricbeat Docker Image [30], configured to send beats to the EMS whose URL was fetched in step 2.
6. The TJob is executed using the ETM REST API.
7. The EMS health-check is continuously queried to retrieve the amount of evens processed. When this number differs from 0, the test finishes successfully. If 125 seconds have passed and EMS still hasn't processed any events, the test finishes with FAILURE.

4.5.2.4.8 ElasTest Orchestrator Engine (EOE)

The ElasTest Orchestrator Engine is still on first stages of design and development, so component end-to-end tests doesn't apply.

4.5.2.4.9 ElasTest Platform Manager (EPM)

The ElasTest Platform Manage doesn't have direct interaction with the user. Its behaviour is tested on every one of the other components' tests. So, no specific tests are needed for the EPM.

4.5.2.4.10 ElasTest Recommendation Engine (ERE).

The ElasTest Recommendation Engine is mainly used through the ElasTest GUI. This first set of tests are avoided, focusing all the effort in the GUI end-to-end tests – See section 4.5.2.5.10 –.

4.5.2.4.11 ElasTest Security Service (ESS)

The ESS end-to-end test is implemented in Python using the request libraries and can be found in the ESS GitHub repository [31]. These tests have been automated in an ElasTest Jenkins job following the recommendations [32].

The test does the following actions in order to check the validity of the ESS functionality:

1. Start the ElasTest Platform.
2. Call the ETM API for getting the status of the running services to check it the ElasTest platform is accessible.
3. Launches an instance of ESS by calling the ETM API for launching a service.
4. A new project called “E2E test ESS” is created, also through the ETM API.
5. A new TJob (“E2E TJob”) is created under the “E2E tests ESS” project. This TJob is based on the Docker image dockernash/ess-e2e [66] that has a python script which makes http request via the 8080 port of a configurable IP address. This IP address is set, as an argument, at run-time. In the test we setup the IP address to be the ESS Man-in-the-middle proxy service running at port 8080.
6. The newly created TJob is run by calling the corresponding ETM API, and it would validate whether the HTTP connection information is produced as output by the ESS.

Steps 5 and 6 tests whether the man in the middle proxy service of ESS is working properly and can be used by TJobs. Step 6 validates also whether the ESS can access the connection information which is necessary for conducting further analysis.

4.5.2.4.12 ElasTest Service Manager (ESM)

The ElasTest Service Manager isn’t shown in the ElasTest GUI currently. But it has a health check endpoint that is being considered to be used on the ElasTest platform to inform the user of the current status of the component. Until this approach is further studied no end-to-end test will be provided.

4.5.2.4.13 ElasTest Tests Manager (ETM)

The ElasTest Tests Manager (ETM) is a web application implemented with Java 8 [51] and Angular 4 [52] as main technologies and build as a Spring Boot application [14]. To test the ElasTest backend, JUnit 5 [15] is used as testing framework and the currently existing tests are available on the ETM GitHub repository [53]. These tests are focused on check the ElasTest operation and for that make use the ETM's API. The functionalities covered by these tests are:

- Project Management
- TJobs Management
- TJobs Execution Management
- SUTs Management

These tests are executed by Jenkins [36] and together with the unit tests, after each code integration, allow us to get to know that ElasTest is still working.

4.5.2.4.14 ElasTest User impersonation Service (EUS)

Regarding the end-to-end tests, two different tests have been implemented:

- **SeleniumE2ETest** [19]. This test starts a standalone instance of EUS, and it is used to feed a Selenium's RemoteWebDriver object. Two different types of browsers (Chrome and Firefox) are used to navigate to a test SUT, in this case Wikipedia. All in all, this test aims to verify that the basic operation of EUS (i.e. the capability of providing browsers following a SaaS approach) is working properly.
- **TimeOutTest** [20]. The EUS, internally implements a timeout for browser sessions. This timeout is reached when no interactions with the session is performed in a given time (60 seconds by default). This test is aimed to verify this feature. To do that, first the timeout is reduced to 5 seconds. Then, test forces to wait longer than that timeout, in this case 10 seconds, verifying that the proper exception (WebDriverException) is thrown by EUS.

These tests, together with the unit and integration ones, are executed by the ElasTest Jenkins server. Concretely, these tests are launched when any new patch is committed to the EUS GitHub repository. The job is labelled as "eus" in Jenkins [36].

4.5.2.5 Integrated UI end-to-end tests

ElasTest is a set of components that should be tested as whole and following user paths through the API. For these tests we have designed 3 steps that should be covered for a complete validation of ElasTest itself:

- **End-to-end component tests on a whole ElasTest (Nightly ElasTest)**. Each of the teams should test its component but from the user point of view i.e. accessing the functionality of the component through the web UI. This step would produce a set of jobs, where each of these jobs would focus in a specific component.
- **End-to-end user flows (Nightly ElasTest).** Main use cases of the ElasTest platform as a whole should be developed and tested. This would produce a set of tests that would model use cases, which implies more than one component, so integration issues can be found.

- **ElasTest tests ElasTest.** One instance stable Instance of ElasTest would make use of the tests and jobs produced on the previous stages and would be run over a SuT that is a newer version of ElasTest.

We have reached maturity in the first step where most of the components have provided a set of Jenkins jobs that test each component integrated in a common ElasTest instance. But there is still work to be done in order to improve the already developed tests and to add new tests that would increase the quality of the ElasTest platform as a whole.

4.5.2.5.1 ElasTest Big Data Service (EBS)

EBS is tested using the ElasTest GUI, using Selenium [43] via Python. This test will create a TJob, utilizing the UI controls of ElasTest platform. The purpose is to prove that EBS can be fully managed via the graphical elements of ElasTest.

The TJob will behave as the not UI end-to-end TJob described for the EBS:

1. Git clone a demo Spark application from ElasTest demo projects [41].
2. Build the application using maven.
3. Download a big text file [42] from the internet and save it to HDFS (EDM).
4. Submit the application to the cluster. This is a word-count application that will return the number of occurrences of every word on the specified file and save the results back in HDFS.
5. Pull the result file back out from HDFS.
6. Perform a printout on the result file.

The tests will check if the results of the TJob are the expected ones.

4.5.2.5.2 ElasTest Cost Engine (ECE)

ECE has been integrated with the ElasTest dashboard, Jenkins job ece-e2e-test [39] performs the end-to-end test to test the correctness of the integration between the overall ElasTest dashboard and the cost engine. The tests have been developed using JUnit 4 [33] and the web integration has been tested using Selenium drivers. The following describes the structure of the test:

- **invokeEce**: This test verifies whether the ECE UI element is available within an iframe of the main ElasTest. The test in the setup phase activates the Test-Engines link from the main navigation panel in the dashboard. It then checks if the ECE engine is present in the list of engines, and then starts the engine by pressing the ‘Start Engine’ button. The test then verifies that specific HTML DIV element in the ECE view is present or not. Presence confirms that the ECE UI is integrated and working as expected in ElasTest dashboard. The JUnit assertion tests whether a known ECE UI element has been displayed or not.

The ECE future development with include ability to do actual cost calculations for all valid executions for the selected TJob. The end-to-end test with the ElasTest GUI will be evolved to test additional visual elements which will be incorporated in the ECE UI.

4.5.2.5.3 ElasTest Data Manager (EDM)

EDM components that can be specifically tested via the platform GUI are described here.

4.5.2.5.3.1 Cerebro/Elasticsearch/Kibana

Cerebro is a web admin tool for Elasticsearch. By performing a UI test of Cerebro, Elasticsearch functionality can also be tested. Unfortunately, Cerebro is not yet accessible from ElasTest dashboard, so end-to-end test cannot be applied yet. The same applies for Kibana.

4.5.2.5.3.2 Cloud Commander

Cloud commander (cloudcmd) is a web-based file manager for managing files stored in HDFS. It is tested via EBS E2E UI test – See 4.5.2.5.1 –, since that test also produces data inside HDFS.

4.5.2.5.4 ElasTest Device Emulator Service (EDS)

The goal of EDS end-to-end tests is to invoke the TSS EDS via ETM, use the facilities offered by EDS in particular the facilities offered by service eds-base to initiate sensors and actuators. Furthermore, the initiated sensors and actuators can be wired at the application level where the end-to-end test is written.

The EDS Integrated and UI Test [46] uses Selenium framework using the Chrome web driver [43] to interact with the ElasTest dashboard in a typical way how user can initiate and run tests. This test is carried out using the Selenium framework using the Chrome web driver in the headless mode.

1. Open ElasTest nightly using a HTTP request with authentication with authentication parameters that is available on the machine running Jenkins.
2. Click the main menu button, followed by "New Project" button. Fill in the project name as "EDSE2E_UI" and press save button.
3. Click on "New TJob" button. In here fill the name of the TJob as "sampletjob". In the field image name, fill in "elastest/eds-base". In the field of SUT, select “None” and press save. Fill in the commands needed to execute the TJob. In the options containing required TSS for the TJob, cross mark the TSS EDS and press save button.
4. Press on the start button of the execution of the TJob.
5. Monitor the dashboard status, for the word "Finished". Until it's obtained keep looping.

6. Once the TJob has completed execution, check the dashboard status again. If SUCCESS is in the dashboard, then TJob exited with a result success else it was a failure.

A complete description of the application that has been used to test the EDS on this test has been provided in section *4.5.2.4.4.1 Test Application for the EDS tests.*

4.5.2.5.5 ElasTest Instrumentation Manager (EIM)

EIM is also tested using the ElasTest dashboard, a web application. To test it, a Jenkins job [48] has been created in the ElasTest CI Server [36]. This job runs automatically EIM end-to-end tests that are implemented using JUNIT 4 [33] and Selenium [43].

- **ObservabilityTest:** this test verifies if a new SuT is correctly monitored by EIM. To achieve this goal, the test follows these steps:
  - Create new project
  - Create SuT outside ElasTest platform based on EIM SuT image [49]
  - Create new SuT in the created project. The SuT have these details:
    - IP address
    - user
    - private key for the user
    - Deployed outside ElasTest option selected
    - Instrumented by ElasTest option selected
    - Log paths: destinations that are monitored by EIM, in order that if any file of these folders changes, a notification is sent.
  - Create new TJob associated to the created SuT, with these details:
    - Associate the TJob to the created SuT
    - Environment Docker image [50]
    - Commands: download a git repository and launch tests of the project.
      - `git clone https://github.com/elastest/demo-projects`
      - `cd demo-projects/unit-java-test`
      - `mvn -B test`
    - Uncheck All-in-one chart check in Metrics and logs section
  - Launch the TJob:
    - When execution starts, click on Open Monitoring Config. button and check SuT node on metrics sections in order that monitoring data appears on screen.
    - The monitoring data appears in the several graphics of the screen:

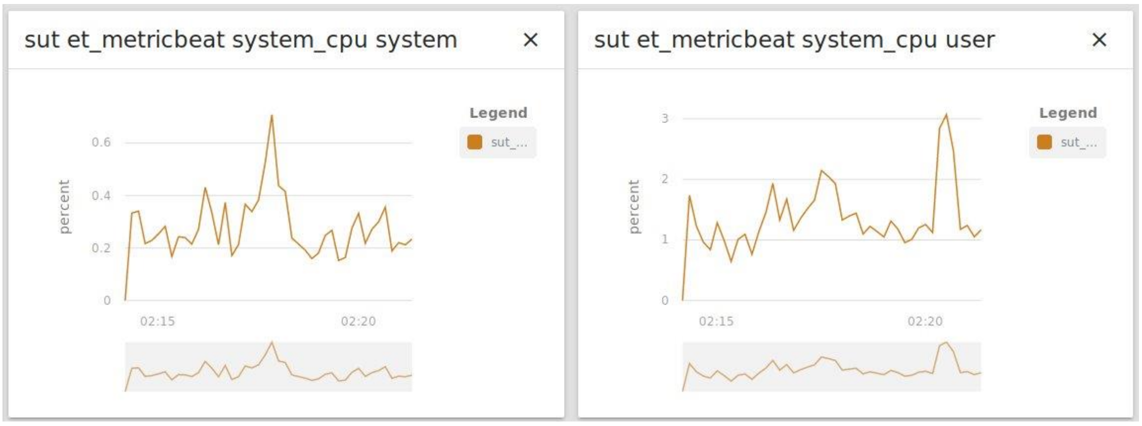


Figure 22. Graphics of EIM execution

- Test is marked as successful when graphics appears on the screen

4.5.2.5.6 ElasTest Monitoring Platform (EMP)

EMP has been integrated with the ElasTest GUI and the e2e-test [67] is the principal mode to test the correctness of the UI integration. In the ElasTest CI server, Components e2e -> elastest-platform-monitoring -> emp-e2e-test performs this integration test. It uses JUnit together with Selenium [43] to invoke various elements of the GUI to assess availability of known web-elements to assert whether the EMP + ElasTest GUI integration still remains valid at that point in time or not. The workflow implemented as part of this test is briefly described next:

- **EMP End2End Tests:** This test is divided into 3 parts, setup, "*invokeEmp*", and clean-up phases. Once the latest ElasTest platform is ready, the Selenium driver via the latest Chrome image, accesses the web UI, and simulates a click the sidebar element to switch to the EMP view. As the UI is accessed for the first time after fresh creation by this end-to-end test, the EMP view will show the login screen of Grafana which is the visualization framework used by EMP. The test validates the login process and relies on the well-known behaviour for this particular version of Grafana used in the EMP setup. The test proceeds to check the login by inserting the username / password in appropriate fields and checking the error popup in case the login fails. This workflow if succeeds confirms the availability of Grafana view in ElasTest GUI and thus confirming the integration with EMP UI elements.

This test is executed by the ElasTest CI server [36].

In future, it is planned to make the EMP administration UI also available on the ElasTest GUI, once this integration is achieved, the EMP end-to-end tests will also be expanded to include elements from the administration UI such as creation of new monitoring spaces, series, health-checks, etc.

4.5.2.5.7 ElasTest Monitoring Service (EMS)

ElasTest Monitoring Service has yet to develop the tests using the UI of ElasTest. But it has been already planned 2 main tests for the integrated end-to-end.

EMS dummy SuT:

It would follow steps described in 4.5.2.4.7 where main functionality of the EMS is tested, without minimum user personalization.

EMS events subscription:

The idea for this test is using the newest EMS capabilities to get feedback to the TJob and define test actions accordingly to this feedback. The general layout for the test will be the following:

1. Create a project with a SuT which runs a custom Docker image (e.g. “imdeasoftware/ems-e2e_0-sut”). This SuT would emit events to the EMS.
2. Create a TJob for this SuT which will subscribe to *interesting* events. These *interesting* events would be configurable dependant on the interest of the TJob or SuT.
3. The EMS will be able to guide the behaviour of the SuT resembling the expected layout of a real use case.

We believe that a well-thought-out testing using these tools will leverage the failure discoveries of certain kinds of bugs which are usually hard to unveil (e.g. race conditions).

4.5.2.5.8 ElasTest Orchestrator Engine (EOE)

The ElasTest Orchestrator Engine is still on first stages of design and development, so component end-to-end tests doesn’t apply.

4.5.2.5.9 ElasTest Platform Manager (EPM)

The ElasTest Platform Manage doesn’t have direct interaction with the user. Its behaviour is tested on every one of the other components’ tests. So, no specific tests are needed for the EPM.

4.5.2.5.10 ElasTest Recommendation Engine (ERE)

ElasTest Recommendation Engine implemented 2 main tests that will cover the full functionality of the component.

Test 1. Asking recommender about suitable tests for reuse.

Prerequisite: Sample model available for queries. This does not need to be performed as a part of the test – sample model is available for the user once Docker container is started.

1. Open the ElasTest dashboard UI.
2. Switch to Test Engines Dashboard.
3. Start ERE container.
4. Log into ERE.
5. Open 'New Recommendation' wizard.
6. Type in query in natural language and specify confidence threshold.
7. Run the query.
8. Verify the returned result.

Test 2. Pre-processing user data.

Prerequisite: (a) Platform is running in experimental mode. EDM component operational. (b) Sample data loaded to Alluxio. This is performed as a pre-test action, using Alluxio REST API.

1. Open the ElasTest dashboard UI.
2. Switch to Test Engines Dashboard.
3. Start ERE container.
4. Log into ERE.
5. Open Admin Dashboard.
6. Open Pre-process tab.
7. Select Alluxio option.
8. Launch pre-processing action.
9. Verify that the action completed with no errors.

4.5.2.5.11 ElasTest Security Service (ESS)

The current integration of the ESS on the ElasTest UI is scheduled to be changed so end-to-end tests using the GUI haven't been developed yet, but they will be implemented using a Jenkins job that does the following:

1. Download ElasTest, launches the ElasTest Platform and gets the IP address of the ElasTest platform (hereinafter referred to as *ELASTEST_IP*).
2. Starts the container for the end-to-end test and executes a python program that contains the end to end tests. During the execution of this program, the value *ELASTEST_IP* is passed as an argument to it.

This program does the following tests:

Test 1: To test whether the ElasTest GUI loads correctly, the program does the following

Opens a Firefox web browser, visits *http://ELASTEST_IP:37000* (ElasTest GUI) and waits until the GUI loads completely.

Test 2: To check whether a new project and a TJob can be created using the ElasTest GUI

1. Click on the "Projects" option in the side panel of the ElasTest dashboard.

2. Click on the "New Project" icon that appears, enter the value "ESS e2e test project" in the input field with placeholder value "Project Name" and click the "Save" button.
3. Click on the <div> with value "ESS e2e test project" to load the project-specific page.
4. Click on the "New TJob" button.
5. Enter the TJob name for the field with placeholder "TJob name", select the value "None" from the drop-down list to select the SuT, provide the value "*dockernash/test-TJob-ess*" [68] to the input field with place holder value" Environment Docker image", tick the checkbox corresponding to "ESS" and click the "Save" button. In this way a new TJob is created. This TJob contains a python script that makes HTTP requests to the domain google.com through the ESS man in the middle proxy. This python script refers to the ESS man in the middle proxy through the ESS environment variable provided by ETM.

Test 3: To test whether a TJob can be analysed by ESS

1. Click on the "Run TJob" button corresponding to the newly created TJob.
2. Wait until the status of the TJob execution changes to "Success" or "Fail" or "Error".
3. If it was "Success", then check whether a new iframe with the ESS GUI has been loaded.
4. Wait until the ESS GUI displays the text "Results".
5. Check whether "google.com" is displayed under the list "Insecure http connections" in the ESS GUI. This indicates that the TJob was able to make connections via ESS and ESS was able to correctly identify the insecure http connection made to the domain google.com.

This end-to-end test will test a unique path that corresponds to the successful execution of a TJob via the ESS. Further tests can be added to test the behaviour during erroneous situations.

4.5.2.5.12 ElasTest Service Manager (ESM)

The ElasTest Service Manager isn't shown in the ElasTest GUI currently. But it has a health check endpoint that is being considered to be used on the ElasTest platform to inform the user of the current status of the component. Until this approach is further studied no end-to-end test will be provided.

4.5.2.5.13 ElasTest Tests Manager (ETM)

The ETM provides ElasTest with a graphical user interface (GUI) and also the ETM must be tested from this. For that purpose, JUnit 5 [15] and Selenium [43] are the selected technologies to implement the tests. As we mentioned above, JUnit 5 is the used as a testing framework and Selenium provides the capability to control a browser from the tests.

The Test Manager end-to-end tests implemented for ElasTest are in the ETM GitHub [54] repository and are described below:

- **EtmUnitTestE2eTest.** This test uses the ElasTest GUI to create the test basic infrastructure (project + TJob) and configures the TJob to execute a simple jUnit test without SUT.
- **EtmRestApiE2eTest.** This test uses the ElasTest GUI to create the test basic infrastructure (project + TJob + SuT) and configures the TJob to use the created SuT and test its API.
- **EtmWebappE2eTest.** This test uses the ElasTest GUI to create the test basic infrastructure (project + TJob + SuT + EUS) and configures the TJob to use the created SuT and test it using its GUI.
- **EtmOpenViduWebRTCE2eTest**. This test uses the ElasTest GUI to create the test basic infrastructure (project + TJob + SuT + EUS) and configures the TJob to use the created SuT and test it using its GUI. This test check correct operating of the media transmission using the browsers provided by the EUS TSS.
- **EtmTestLinkE2eTest.** This test uses the ElasTest GUI to check the status of a TestLink test created previously on ElasTest and below executes this test on ElasTest and verify that these have finished correctly.

4.5.2.5.14 ElasTest User impersonation Service (EUS)

EUS is also tested using the ElasTest GUI (web application). To that aim, the project "e2e-test" located in the EUS GitHub repository has been created. This project contains two JUnit 5 [15] tests using Selenium by means of the JUnit 5's extension Selenium-jupiter [22]:

- **EusSupportServiceE2eTest** [23]. This test uses EUS as a Test Support Service through the ElasTest GUI.
- **EusTJobE2eTest** [24]. This test uses EUS in a TJob through the ElasTest GUI. The configuration of the TJob is done automatically by the test using an existing test repository in GitHub [25].

Both tests are executed by the ElasTest Jenkins when a new patch in the EUS source code is committed.

4.6 Releasing

For the releasing phase we have designed some procedures, as well as a set of rules that developers and publishers should follow.

4.6.1 Maven/Gradle Artifacts.

4.6.1.1 Basic rules and best practices

- All public artifacts (snapshots and releases) will be available for everybody (consortium and outside the consortium).
- Private artifacts (snapshots and releases) will be only available for people from the consortium, under nominal users linked with GitHub login.
- All the artifacts public and private must follow the requirements for Central publishing (even if they are not to be published in Central).
- It is mandatory to use stable versions of Maven 3 since other versions of Apache Maven including all versions of Maven 2 and Maven 1 have reached their end of life therefore they are no longer supported.

4.6.1.2 Procedure

For public artifacts these procedures are MUSTs, as artifacts that not satisfy Central requirements won't be synchronized and won't be accessed. For private artifacts these procedures are recommendations.

- **Javadoc and Sources**

  Projects with packaging other than `pom` have to supply JAR files that contain Javadoc and sources. This allows the consumers of your components to automatic access to Javadoc and sources for browsing as well as for display and navigation e.g. in their IDE.

  The naming convention for these files following the Maven repository format uses the classifiers `javadoc` and `sources` and assembles the names with `artifactId` and `version` with packaging `jar`.

- **Sign Files with GPG/PGP**

  All files deployed need to be signed with GPG/PGP and an `.asc` file containing the signature must be included for each file.

- **Sufficient Metadata**

  - *Correct Coordinates*

    The project coordinates, also known as **GAV**, coordinates determine the location of your project in the repository. The values are:

    - `groupId`: `io.elastest.`<subgroup> (if applies)
    - `artifactId`: the unique name of the component
    - `version`: <semantic versioning[12]> <`-SNAPSHOT`> (when applies -this won't be released to Central-)

- *Project Name, Description and URL*

  For some human readable information about the project and a pointer to the ElasTest website:

  - `name:` ${project.groupId}:${project.artifactId}
  - `Description`: description of the artifact.
  - `URL`: GitHub URL or if more descriptive web page has been made for the module/component.

- *License Information*

  The license definition by default as specified during the early stages of the project public components should follow Apache License 2.0 [10].

- *Developer Information*

  The developers section will provide developers information. When specific developer information isn't relevant or partners don't want to provide personal information, it can be set as follows:

```
<developers>
	<developer>
		<id>elastest.io</id>
		<name>-elastest.io Community</name>
		<organization>Elastest.io</organization>
		<organizationUrl>http://elastest.io</organizationUrl>
	</developer>
</developers>
```

- *SCM Information*

  The connection to the GitHub. This part can be skipped by private artifacts.

```
<scm>
	<connection>
		scm:git:git://github.com/elastest/<repo>.git
	</connection>
	<developerConnection>
		scm:git:ssh://github.com:elastest/<repo>.git
	</developerConnection>
	<url>
		http://github.com/elastest/<repo>/tree/master
	</url>
</scm>
```

- **Specific ElasTest artifact Requirements**

  As it has been explained all public artifacts must meet the previous requirements. In the case of the private ones these are recommended, obviously `-sources.jar` and `scm` specification should be left out.

  In addition to the previous indication the following rules should be considered as best practices, and recommended for all the artifacts:

- *groupId:*
  - For artifacts belonging to a specific architecture component use: io.elastest.<architecture-component-identificator>
  - Other artifacts can make their own group as long as it doesn't interfere with the architecture groups.
- *artifactId:* use identifiable names.
- Use *Nexus Staging Maven plugin* for both public and private repositories are Sonatype Nexus software.
- Define version of plugins and dependencies as properties.

4.6.1.3 Configuration for consuming private Maven artifacts.

Private Maven artifacts are released onto our own Nexus Artifact repository. In order to consume them it is important to have a strict policy so they aren't distributed outside the consortium.

Therefore, we have configured the Nexus to be accessed only under nominal user-names linked with the GitHub accounts. As this feature isn't built on the repository itself it is delegated to the Private User Registry – see 4.1.5.5 –.

Developers would use the credentials provided by the Private User Registry on their maven settings.xml.

In order to being able to use it in the development environment, the user should modify its maven settings.xml to provide the definition of the server:

```
<servers>
	<server>
		<id>nexus-private-releases</id>
		<username>github-user</username>
		<password>provided-password</password>
	</server>
	<server>
		<id>nexus-private-snapshots</id>
		<username>github-user</username>
		<password>provided-password</password>
	</server>
</servers>
```

And then configure the project's pom so it makes use of that repository:

```
<repository>
	<id>nexus-private-releases</id>
	<name>Elastest Private Maven</name>
	<url>http://ci.elastest.io/nexus/repository/maven-snapshots/</url>
	<layout>default</layout>
	<releases>
		<enabled>true</enabled>
		<updatePolicy>never</updatePolicy>
	</releases>
	<snapshots>
		<enabled>false</enabled>
	</snapshots>
```

```
</repository>
<repository>
      <id>nexus-private-snapshots</id>
      <name>Elastest Private Maven</name>
      <url>http://ci.elastest.io/nexus/repository/maven-releases/</url>
      <layout>default</layout>
      <releases>
            <enabled>true</enabled>
            <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
            <enabled>false</enabled>
      </snapshots>
</repository>
```

4.6.2 Docker Images.

We have some different kind of Docker images within the ElasTest project:

- **Public ElasTest components images:** These are the public components of ElasTest prepared to be launched and executed by the ElasTest platform (ElasTest ToolBox).
- **Private ElasTest components images:** These are the private components of ElasTest prepared to be launched and executed by the ElasTest platform (ElasTest ToolBox) if they are available in the system.
- **Environmental images:** These images are used by many of the Jenkins jobs to provide the necessary environment for the job to execute the steps needed to reach its goal.

4.6.2.1 How to create component images

The images will be pushed by a Jenkins job created with pipeline plugin[1].

- **Previous actions:**

  Check the repository you want to push into is already created, if it isn't contact the administrators at elastest@naevatec.com

  Name the image with the naming convention (<repository>/<image-name>:<tag>) taking into account that for public images the repository is "elastest" and for the private images the repository has this format: "<amazonId>.dkr.ecr.eu-west-1.amazonaws.com/elastest"

- **Jenkins job:**

  Inside the job it is necessary to login into the "elastestci" DockerHub account or the AWS ECR. The credentials are registered inside the Jenkins Server.

  We can push the image by: `<image>.push`

  Example:

[1] The Docker plugin won't work outside the pipelines with the current configuration of Jenkins slaves

```
def myimage = docker.build "<reposistory>/<image-name>:<tag>”
withCredentials([[ $class: 'UsernamePasswordMultiBinding',
      credentialsId: <credentials_id>,
      usernameVariable: 'USERNAME',
      passwordVariable: 'PASSWORD']]){
            sh 'Docker login -u "$USERNAME" -p "$PASSWORD"'
            myimage.push()
      }
```

- **Confirmation:**

  If the job result is SUCCESS, check the <repository>/<image-name>:<tag> and you should see the Last Updated row as seconds ago.

  Try from any place to pull the image and run it. It is as simple as execute `Docker pull elastest/<component-acronym>:<tag>` in any machine with Docker installed and configured. Log for the private repositories with the nominative credentials provided by the *Private User Registry*, no login is necessary for pulling the public images.

4.6.2.2 How to create Environmental Images

Images used in CI are considered as “global” because can be used in the jobs of any component. For that reason, they are managed by Jenkins admins. There is a GitHub repository that have Dockerfiles (and other needed files) to create such images. That repository is https://github.com/elastest/ci-images. In order to create a new CI Image please follow the procedure:

1. Create a new pull request that contains a new folder named after the image.
2. Provide inside the folder all the necessary files for building the image and a README.md.
3. Contact the Jenkins admin (elastest@naevatec.com) to generate the DockerHub repository and configure the autobuild from DockerHub.
4. To do some modification to the image, just create a new PR with the changes.

4.6.3 Documentation.

All the public SW released should have enough documentation associated to guide the user through its usage. Also, it should point clearly to the ElasTest’s GitHub repositories and to the ElasTest Community page.

4.6.3.1 Docker Images descriptions

4.6.3.1.1 Short Description

- 100 characters maximum
- Short descriptions should provide a general idea of the image.
  - For component main images: The full name of the component should be used.
    - **Example:**

eus: ElasTest User Emulator Service provides browsers for manual and automated interaction.

- For services or subcomponents:
  - Full name of the service or subcomponent should be used or commercial name.
  - Full name of the parent component, if max characters limit is reached, component acronym can be used instead.
  - **Example:**

    eus-novnc: Alpine X11 server and HTML5 VNC client with autofocus for the ElasTest User Emulator Service.

4.6.3.1.2 Long Description

- 25.000 character maximum
- Should be provided as a file *.md on where the Dockerfile is placed (GitHub repository). The file can be called readme.md, DockerDescription.md or <Docker-ComponentAcronym>.md and can be placed in the same folder where the Dockerfile is placed or in a specific folder.
- The document should contain these sections:
  - Badges
  - ElasTest Header
  - Name of the Image
  - What is this? Description of the image
  - Quick Reference
  - Where to get help.
  - Where to file issues.
  - Maintained by.
  - Published image artifact details: (image metadata, transfer size, etc.).
  - Source of this description:
  - Supported Docker versions.
  - What's on this image (not always necessary dependent on the content of the image)
  - How to use the image
- A template for the *.md document can be found in.

4.7 Maintenance.

The maintenance for the whole hosted CI environment has been developed and documented, in order to be carried regularly.

Maintenance action will follow these steps:

4.7.1 Request to admin.

When any CI environment user detects any SW malfunctioning, is aware of security updates or finds a new tool or SW to be added to the environment; should notify the CI environment admins (elastest@naevatec.com) with the issue describing:

- Tool or tools affected.
- Reason: Security / malfunction
- Action to be taken (if known).
- Priority:
    - Critical: to be solved as soon as possible.
    - Important: can't wait for the next maintenance window.
    - Normal: to be solved with the next maintenance window.
    - Low: good to have. But otherwise not blocking, not necessary.

CI environment admin would attend to all the requests, confirming its priority and when more information is required, they would contact the user for clarification.

4.7.2 Plan and document the maintenance.

The CI environment admins will evaluate the impact of the actions to be held, fill the Maintenance Window Template (See *A4. Maintenance Window Procedure Template*) and schedule the intervention of the CI environment.

4.7.3 Execution.

On the scheduled time window, the procedure will be executed following all the instructions filled in the maintenance window document. A standard maintenance window will consist in the following steps:

- Start of the procedure & communication:
- Back Up
- Upgrade
- Test and confirmation
- Rollback
- Communication of results

5 References

[1] Jenkins – Build great thinks at any scale- https://jenkins.io

[2] DockerHub - Dev-test pipeline automation, 100,000+ free apps, public and private registries - https://hub.docker.com/

[3] OSSRH – The Central Repository: Serving Open Source Components Since 2002 - http://central.sonatype.org/

[4] Nexus Repository Manager OSS - The world's only repository manager with FREE support for popular formats. - https://www.sonatype.com/nexus-repository-oss

[5] Private User Registry - Manage the creation of user access to private resource - https://ci.elastest.io/user-registry/

[6] ElasTest - An elastic platform to ease end to end testing – https://elastest.io

[7] Amazon ECR - Amazon Elastic Container Registry - https://aws.amazon.com/ecr/?nc1=h_ls

[8] GitHub - development platform inspired by the way you work. - https://github.com/

[9] CodeCov.io - Enhancing development workflows and improving code quality. - https://codecov.io/

[10] Apache 2.0 License - http://www.apache.org/licenses/LICENSE-2.0

[11] ElasTest AWS Deployment documentation - http://elastest.io/docs/deploying/aws/

[12] Semantic Versioning - https://semver.org/

[13] AWS Client Installation guide - http://docs.aws.amazon.com/cli/latest/userguide/installing.html

[14] Spring-Boot - https://projects.spring.io/spring-boot/

[15] JUnit 5 - http://junit.org/junit5/

[16] EUS GitHub repository - https://github.com/elastest/elastest-user-emulator-service/tree/master/eus

[17] Mockito - http://site.mockito.org/

[18] Spring Framework - https://projects.spring.io/spring-framework/

[19] Selenium EUS E2E Test - https://github.com/elastest/elastest-user-emulator-service/blob/master/eus/src/test/java/io/elastest/eus/test/e2e/SeleniumE2ETest.java

[20] TimeOutTest - https://github.com/elastest/elastest-user-emulator-service/blob/master/eus/src/test/java/io/elastest/eus/test/e2e/TimeoutTest.java

[21] EUS E2E GitHub repository - https://github.com/elastest/elastest-user-emulator-service/tree/master/e2e-test

[22] Selenium-jupiter - https://bonigarcia.github.io/Selenium-jupiter/

[23] EusSupportServiceE2eTest - https://github.com/elastest/elastest-user-emulator-service/blob/master/e2e-test/src/test/java/io/elastest/eus/test/e2e/EusSupportServiceE2eTest.java

[24] EusTJobE2eTest - https://github.com/elastest/elastest-user-emulator-service/blob/master/e2e-test/src/test/java/io/elastest/eus/test/e2e/EusTJobE2eTest.java

[25] Test TJob for E2E test - https://github.com/elastest/elastest-user-emulator-service/tree/master/TJob-test

[26] ElasTest E2E EUS job - https://ci.elastest.io/jenkins/job/elastest-user-emulator-service/job/eus-e2e-test/

[27] EMS Jenkinsfile for e2e testing - https://github.com/elastest/elastest-monitoring-service/blob/master/e2e-test/Jenkinsfile

[28] EMS tests bash script - https://github.com/elastest/elastest-monitoring-service/blob/master/e2e-test/start-instance.sh

[29] Dummy Docker Image - https://hub.docker.com/r/williamyeh/dummy/

[30] Metricbeats Docker Image - http://docker.elastic.co/beats/metricbeat:5.4.0

[31] ESS e2e component test - https://github.com/elastest/elastest-security-service/blob/master/e2e-test/e2etest.py
[32] ESS e2e component Jenkins file - https://github.com/elastest/elastest-security-service/blob/master/e2e-test/Jenkinsfile
[33] JUnit 4 - https://junit.org/junit4/
[34] Cobertura - http://cobertura.github.io/cobertura/
[35] Maven - https://maven.apache.org/
[36] ElasTest CI Server - https://ci.elastest.io/jenkins/
[37] Spring-boot - https://projects.spring.io/spring-boot/
[38] Thymeleaf - https://www.thymeleaf.org/
[39] ECE e2e Jenkins job - https://ci.elastest.io/jenkins/view/Component-e2e/job/elastest-cost-engine/job/ece-e2e-test/
[40] Apache Spark - https://spark.apache.orgç
[41] EBS ElasTest demo projects - https://github.com/elastest/demo-projects/tree/master/ebs-test
[42] Project Gutenberg, “The Adventures of Sherlock Holmes” - https://norvig.com/big.txt
[43] Selenium - http://www.seleniumhq.org/projects/webdriver/
[44] EDS Jenkins file - https://github.com/elastest/elastest-device-emulator-service/blob/master/demo/e2e_tests/Jenkinsfile
[45] EDS API test - https://github.com/elastest/elastest-device-emulator-service/blob/master/demo/e2e_tests/e2e_test_api.py
[46] EDS UI test - https://github.com/elastest/elastest-device-emulator-service/blob/master/demo/e2e_tests/e2e_test_ui.py
[47] JAX-RS - https://github.com/jax-rs
[48] EIM e2e Job - https://ci.elastest.io/jenkins/job/elastest-instrumentation-manager/job/eim-e2e/
[49] ElasTest EIM SuT image - https://hub.docker.com/r/elastest/eim-sut/
[50] ElasTest ETM alpine git java image - https://hub.docker.com/r/elastest/test-etm-alpinegitjava/
[51] Java8 - http://www.oracle.com/technetwork/java/javase/overview/index.html
[52] Angular 4 - https://angular.io/
[53] ETM GitHub Repository - https://github.com/elastest/elastest-torm/tree/master/elastest-torm
[54] Torm E2E tests GitHub - https://github.com/elastest/elastest-torm/tree/master/e2e-test
[55] Atlassian Bamboo - https://www.atlassian.com/software/bamboo
[56] CruiseControl - http://cruisecontrol.sourceforge.net/
[57] Team Foundation Server - https://www.visualstudio.com/tfs/
[58] Travis CI - https://travis-ci.org/
[59] ElasTest Jenkins Library - https://github.com/elastest/ci-elastest-jenkins-lib
[60] ElasTest Toolbox Github - https://github.com/elastest/elastest-toolbox/tree/master/
[61] About code coverage - https://docs.codecov.io/docs/about-code-coverage
[62] SonarCloud - https://about.sonarcloud.io/

[63] SonarQube - https://www.sonarqube.org/
[64] SonarLint - https://www.sonarlint.org/
[65] SonarSource - https://www.sonarsource.com/
[66] ESS end-to-end test image - https://hub.docker.com/r/dockernash/ess-e2e/
[67] ElasTest EMP E2E job - https://ci.elastest.io/jenkins/view/Component-e2e/job/elastest-platform-monitoring/job/emp-e2e-test/
[68] ESS TJob Docker image - https://hub.docker.com/r/dockernash/test-TJob-ess/

ANNEXES

A1. GitHub Readme.md Template.

```
[![License badge](https://img.shields.io/badge/license-Apache2-
orange.svg)](http://www.apache.org/licenses/LICENSE-2.0)
[![Documentation badge](https://img.shields.io/badge/docs-latest-
brightgreen.svg)](http://elastest.io/docs/)

[![][ElasTest Logo]][ElasTest]

Copyright © 2017-2019 [<member>]. Licensed under
[Apache 2.0 License].

<repository_name>
=================

<repository_description>

What is ElasTest?
-----------------

This repository is part of [ElasTest], which is a flexible open source
testing platform aimed to simplify the end-to-end testing processes for
different types of applications, including web and mobile, among others.

The objective of ElasTest is to provide advance testing capabilities aimed
to increase the scalability, robustness, security and quality of
experience of large distributed systems. All in all, ElasTest will make
any software development team capable of delivering software faster and
with fewer defects.

Documentation
-------------
The ElasTest project provides detailed [documentation][ElasTest Doc]
including tutorials, installation and development guide.

Source
------
Source code for other ElasTest projects can be found in the [GitHub
ElasTest Group].

News
----
Check the [ElasTest Blog] and follow us on Twitter [@elastestio][ElasTest
Twitter].

Issue tracker
-------------

Issues and bug reports should be posted to the [GitHub ElasTest
Bugtracker].

Licensing and distribution
--------------------------
```

```
Licensed under the Apache License, Version 2.0 (the "License"); you may
not use this file except in compliance with the License. You may obtain a
copy of the License at   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
License for the specific language governing permissions and limitations
under the License.

Contribution policy
-------------------
You can contribute to the ElasTest community through bug-reports, bug-
fixes, new code or new documentation. For contributing to the ElasTest
community, you can use GitHub, providing full information about your
contribution and its value. In your contributions, you must comply with
the following guidelines
* You must specify the specific contents of your contribution either
through a detailed bug description, through a pull-request or through a
patch.
* You must specify the licensing restrictions of the code you contribute.
* For newly created code to be incorporated in the ElasTest code-base, you
must accept ElasTest to own the code copyright, so that its open source
nature is guaranteed.
* You must justify appropriately the need and value of your contribution.
The   ElasTest project has no obligations in relation to accepting
contributions from third parties.
* The ElasTest project leaders have the right of asking for further
explanations, tests or validations of any code contributed to the
community before it being incorporated into the ElasTest code-base. You
must be ready to addressing all these kind of concerns before having your
code approved.

Support
-------

The ElasTest project provides community support through the [ElasTest
Public
Mailing List] and through [StackOverflow] using the tag *elastest*.

<p align="center">
  <img src="http://elastest.io/images/logos_elastest/ue_logo-
small.png"><br>
  Funded by the European Union
</p>

[Apache 2.0 License]: http://www.apache.org/licenses/LICENSE-2.0
[ElasTest]: http://elastest.io/
[ElasTest Blog]: http://elastest.io/blog/
[ElasTest Doc]: http://elastest.io/docs/
[ElasTest Logo]: http://elastest.io/images/logos_elastest/elastest-logo-
gray-small.png
[ElasTest Public Mailing List]:
https://groups.google.com/forum/#!forum/elastest-users
[ElasTest Twitter]: https://twitter.com/elastestio
[GitHub ElasTest Group]: https://github.com/elastest
[GitHub ElasTest Bugtracker]: https://github.com/elastest/bugtracker
```

```
[StackOverflow]: http://stackoverflow.com/questions/tagged/elastest
[<member>]: <member_url>
```

A2. Docker Image long description template

```
<!--
************************************************************************

ELASTEST - Template for Docker Images README

************************************************************************ -
->
<!-- badges -->

[![License badge](https://img.shields.io/badge/license-Apache2-
orange.svg)](http://www.apache.org/licenses/LICENSE-2.0)
[![Docker
badge](https://img.shields.io/docker/pulls/elastest/etm.svg)](https://hub.
docker.com/r/elastests/etm/)

<!-- Elastest logo -->
[![][ElasTest Logo]][ElasTest]

Copyright © 2017-2019 [ElasTest]. Licensed under [Apache 2.0 License].

elastest/\<image\>
==============================

# What is this? (TITLE of the image and short description)

===============================

# Supported tags and respective `Dockerfile` links
-      [`1.0`, `1.1`, `1`
(*1.0/Dockerfile*)](https://github.com/elastest/environments)
-      [`2.0`, `2.1`, `2.2`, `2`
(*2.0/Dockerfile*)](https://github.com/elastest/environments)

# Quick reference

-      **Where to get help**:
       [the ElastTest mailing list](), [the Elastest Slack](), or [Stack
Overflow]()

-      **Where to file issues**:
       Issues and bug reports should be posted to the [GitHub ElasTest
Bugtracker].

-      **Maintained by**:
       [the ElasTest community](https://github.com/elastest)

-      **Published image artifact details**:
       (image metadata, transfer size, etc).
```

```
-       **Source of this description**:
        [docs repo's `template/` directory](https://github.com/elastest/ci-
images/edit/master/doc-templates/template.md)
([history](https://github.com/elastest/ci-images/commits/master/doc-
templates))

-       **Supported Docker versions**:
        [the latest
release](https://github.com/docker/docker/releases/latest) (down to
17.03.1 on a best-effort basis)

# What's on this image?
<!-- tools and purpouse -->

# How to use this image

## Dependencies (other containers or tools)

## Integration with other containers or tools)

[Apache 2.0 License]: http://www.apache.org/licenses/LICENSE-2.0
[ElasTest]: http://elastest.io/
[ElasTest Logo]: http://elastest.io/images/logos_elastest/elastest-logo-
gray-small.png
[ElasTest Twitter]: https://twitter.com/elastestio
[GitHub ElasTest Group]: https://github.com/elastest
[GitHub ElasTest Bugtracker]: https://github.com/elastest/bugtracker
```

A3. Cobertura Reports and CodeCov.io integration procedure.

A3.1. Integration

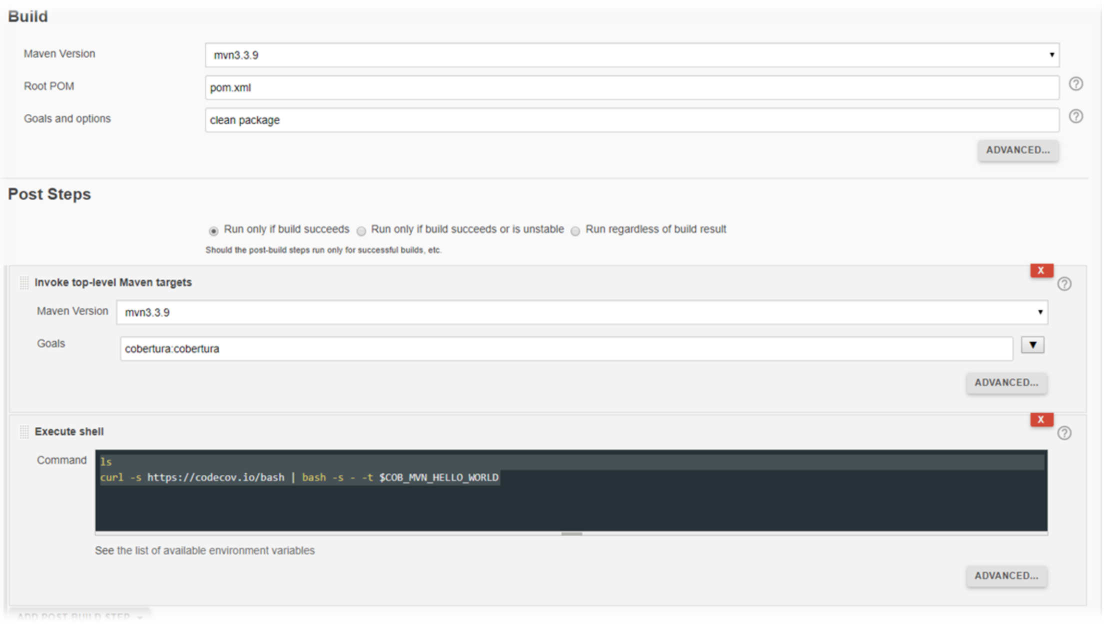
Codecov can be integrated in the Jenkins Jobs. There are two example jobs in the ElasTest Jenkins to show how to use it with maven:

- Hello-world-Docker-pipeline:

```
node('Docker'){
    stage "Container Prep"
        echo("the node is up")
        def mycontainer = docker.image('elastest/docker-
siblings:latest')
        mycontainer.pull() // make sure we have the latest available
                           // from Docker Hub
        mycontainer.inside {
            git 'https://github.com/elastest/mvn-hello-world.git'

            stage "Tests"
                echo ("Starting maven tests")
                sh 'mvn clean test'
                step([$class: 'JUnitResultArchiver', testResults:
'**/target/surefire-reports/TEST-*.xml'])

            stage "Package"
```

```
                    echo ("Packaging")
                    sh 'mvn package -DskipTests'

               stage "Cobertura"
                    sh 'mvn cobertura:cobertura'
                    sh('cd src && git rev-parse HEAD > GIT_COMMIT')
                         git_commit=readFile('src/GIT_COMMIT')

                    sh 'export GIT_COMMIT=$git_commit'

                    sh 'export GIT_BRANCH=master'
                    def codecovArgs = "-v -t $COB_MVN_HELLO_WORLD"

                    echo "$codecovArgs"

                    def exitCode = sh(
                         returnStatus: true,
                         script: "curl -s
https://raw.githubusercontent.com/codecov/codecov-bash/master/codecov
| bash -s - $codecovArgs")
                         //script: " pip install --user codecov && codecov
-v -t $COB_MVN_HELLO_WORLD")
                         if (exitCode != 0) {
                              echo( exitCode +': Failed to upload code
coverage to codecov')
                         }
               }

          stage "Archive artifacts"
                    archiveArtifacts artifacts: 'target/*.jar'
}
```

- <u>mvn–hello-world:</u>



Build

Maven Version	mvn3.3.9
Root POM	pom.xml
Goals and options	clean package

Post Steps

[✓ Run only if build succeeds] Run only if build succeeds or is unstable Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Invoke top-level Maven targets

Maven Version	mvn3.3.9
Goals	cobertura:cobertura

Execute shell

Command

```
ls
curl -s https://codecov.io/bash | bash -s - -t $COB_MVN_HELLO_WORLD
```

See the list of available environment variables

Figure 23. Test coverage job configuration

For other kind of projects there is more information at https://docs.codecov.io/docs/supported-languages and ElasTest environment are available to solve any doubts or problems.

In order to upload to Cobertura reports to codecov.io from Jenkins is necessary to use a unique token for project (GitHub repo). All the tokens have been already registered in Jenkins.

The tokens have the following Name Structure:

```
COB_<component_Acronym>_TOKEN
```

For example:

```
COB_EUS_TOKEN
```

These tokens can be used just as any environment variable:

```
$ curl -s https://codecov.io/bash | bash -s - -t $COB_EUS_TOKEN
```

A3.2. Other Useful integration

A3.2.1.Badges:

```
[![codecov](https://codecov.io/gh/elastest/elastest-user-emulator-
service/branch/master/graph/badge.svg)](https://codecov.io/gh/elastest
/elastest-user-emulator-service)
```

This badge can be retrieved from the https://codecov.io/gh/elastest/<component-repo>/settings/badge. In this page you can retrieve the badge markdown for GitHub.

Only repositories that haven't upload any report to codecov.io will show the unknown status.

A4. Maintenance Window Procedure Template

A4.1. General Information

Affected tools / SW

Template to be filled on each Maintenance Window one row per tool.

SW to be upgraded	Old version	New version	Documentation	Location

<name of the tool>	Currently installed tool	Proposed version	Link to the official documentation of the proposed version	host / container in host / image

Motivation

[Scheduled maintenance window]

[Critical request]

- ¿Who has requested it?
- ¿Why has been defined as critical?

Risks

- [Docker] The images that use the host Docker could fail until their own Docker is upgraded.
- [Jenkins] Some Jobs may need to be reconfigured to work.
- ...

Contact information.

	Partner	User (Name)	Email
Requester			
Upgrade responsible	NaevaTec		
Notify to	ALL	WP6	elastest-wp6@googlegroups.com

Upgrade Plan

	Date and Time	Duration*
Start	**{_DATE_} 07:15 am**	**2 h**
Init communication	{_DATE_} 07:15 am	5 min
Back Up	{_DATE_} 07:20 am	15 min
Upgrade	{_DATE_}07:35 am	1 h
Test and confirmation	{_DATE_} 08:35 am	25 min
Rollback	{_DATE_} 09:00 am	10 min
Communication of results	{_DATE_} 09:15 am	5 min
End of the Upgrade	**{_DATE_} 09:15 am**	

A4.2. Procedure

A4.2.1.Notification

WP6 users will be notified through the mailing list at the beginning of the upgrade procedure. Expected maintenance time will be reminded in this email.

A4.2.2. System shutdown.

A4.2.2.a Main Instance.

The administrators will check the CI environment is available for the upgrade.

- No jobs are being executed
- No user processes are executing.

If there are slaves up, they will be stopped.

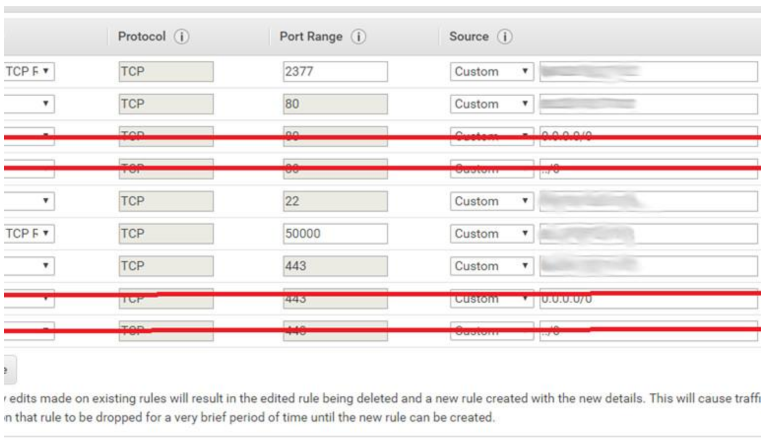
The security group will disable public http access to the instance only access from NaevaTec will be accepted. ()

A4.2.2.b Slaves.

No actions required

A4.2.2.c ElasTest Nightly.

The security group will disable public http access to the instance only access from NaevaTec will be accepted. (Figure 24. AWS disable inbound rules)



	Protocol	Port Range	Source	
TCP F	TCP	2377	Custom	[redacted]
	TCP	80	Custom	[redacted]
	~~TCP~~	~~80~~	~~Custom~~	~~0.0.0.0/0~~
	~~TCP~~	~~80~~	~~Custom~~	~~::/0~~
	TCP	22	Custom	[redacted]
TCP F	TCP	50000	Custom	[redacted]
	TCP	443	Custom	[redacted]
	~~TCP~~	~~443~~	~~Custom~~	~~0.0.0.0/0~~
	~~TCP~~	~~443~~	~~Custom~~	~~::/0~~

edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffi
n that rule to be dropped for a very brief period of time until the new rule can be created.

Figure 24. AWS disable inbound rules

A4.2.3.Back Up

A4.2.3.a Main Instance.

[1] Push user Registry Image:

```
$ $(aws ecr get-login --no-include-email)
$ cd ci-containersEnviroment/private-user-registry
$ docker ps #get the id of the private-user-registry container
$ docker commit <private-user-registry_id> 842800759158.dkr.ecr.eu-west-
1.amazonaws.com/elastest/private-user-registry:AAAAMMDD
$ Docker push 842800759158.dkr.ecr.eu-west-
1.amazonaws.com/elastest/private-user-registry:AAAAMMDD
```

[2] Create Snapshot (Figure 25. AWS EC2. Create Image)

- Select instance: elastest-ci
- AMI Backup -> Image / Create Image. (Figure 26. AWS EC2. Configuration of the Image)
  - **name:** elastestci_AAAAMMDD
  - **description:** AAAAMMDD_Maintenance_Window

[3] Wait until Image status is: available. (Figure 27. AWS EC2. Available Image)

A4.2.3.b Slaves.

No actions required

A4.2.3.c ElasTest Nightly.

[1] Create Snapshot: (Figure 25. AWS EC2. Create Image)

- Select instance: Secure-ElastestStack-v001
- AMI Backup -> Image / Create Image. (Figure 26. AWS EC2. Configuration of the Image)
  - **name:** nigthly_AAAAMMDD
  - **description:** AAAAMMDD_Maintenance_Window

Wait until Image status is: available. (Figure 27. AWS EC2. Available Image)

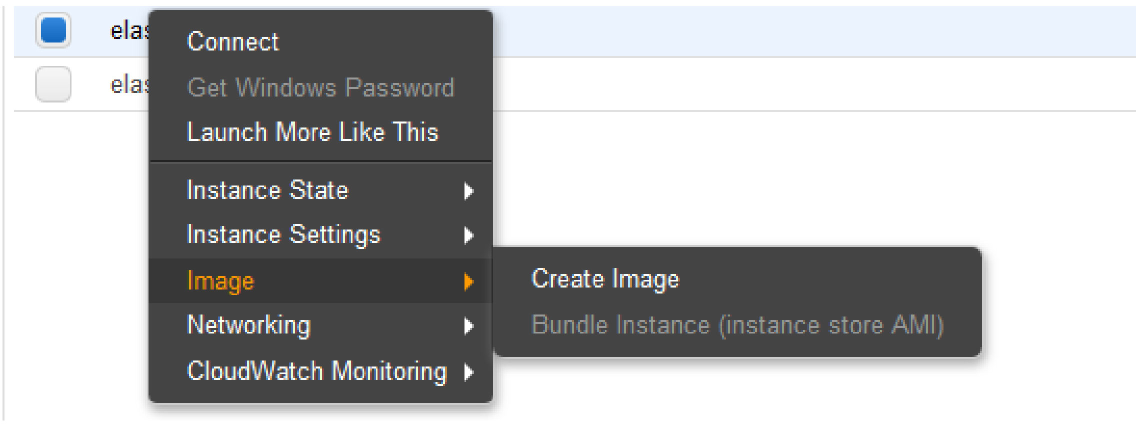
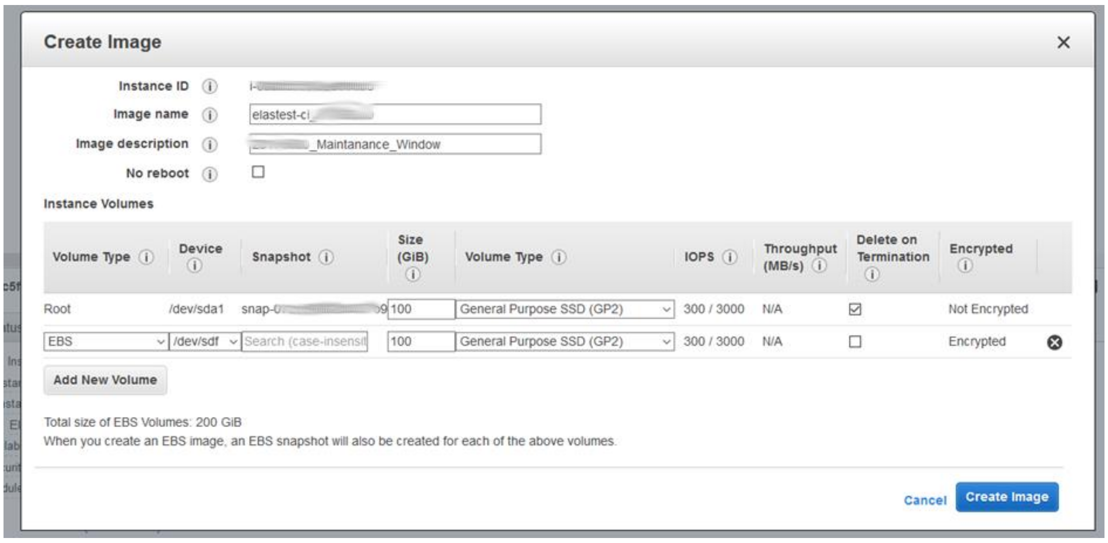


Figure 25. AWS EC2. Create Image



Create Image

Instance ID	i-
Image name	elastest-ci_
Image description	_Maintanance_Window
No reboot	☐

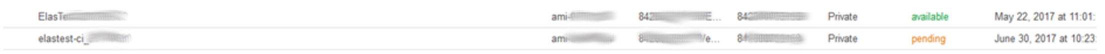
Instance Volumes

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snap-0	100	General Purpose SSD (GP2)	300 / 3000	N/A	☑	Not Encrypted
EBS	/dev/sdf	Search (case-insensit	100	General Purpose SSD (GP2)	300 / 3000	N/A	☐	Encrypted

Add New Volume

Total size of EBS Volumes: 200 GiB
When you create an EBS image, an EBS snapshot will also be created for each of the above volumes.

Figure 26. AWS EC2. Configuration of the Image



ElasTe	ami-	842 E...	842	Private	available	May 22, 2017 at 11:01:
elastest-ci_	ami	84 /e...	84	Private	pending	June 30, 2017 at 10:23

Figure 27. AWS EC2. Available Image

A4.2.4. Upgrade

A4.2.4.a Main Instance.

- **Kernel (5min):**
  - Update and upgrade

```
$ sudo apt update
$ sudo apt upgrade
```

  - If unused packages:

```
$ sudo apt autoremove
```

  - Reboot

```
$ sudo reboot
```

- **Docker:**
  - All the containers will be stopped.

```
$ Docker stop $(Docker ps -a -q)
```

  - Docker Images will be cleared.

```
$ Docker rmi -f $(Docker images -q)
```

  - Docker will be upgraded in the host with:

```
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
$ sudo apt update
$ sudo apt-get install docker-ce=<DOCKER_NEW_VERSION>
```

- **Docker Compose:**
  - Run this command to download the latest version of Docker Compose:

```
$ sudo curl -L
https://github.com/docker/compose/releases/download/<docker-
compose_NEW_VERSION>/docker-compose-`uname -s`-`uname -m` -o
/usr/local/bin/Docker-compose
```

 - Apply executable permissions to the binary:

```
$ sudo chmod +x /usr/local/bin/Docker-compose
```

 - Test the installation.

```
$ Docker-compose --version
```

- **AWS cli**
  - uninstall old version

```
$ sudo apt-get remove awscli
```

  - install new version

```
$ sudo curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o
"awscli-bundle.zip" &&  sudo  unzip awscli-bundle.zip
&&  sudo   ./awscli-bundle/install -i /var/lib/aws -b /usr/bin/aws
```

- **User-Registry**
  - Log in in aws ecr:

```
$ $(aws ecr get-login --no-include-email)
```

  - Modify docker-compose.yml to retrieve backup instead of clean image.
  - Start container:

```
$ Docker-compose up -d
```

- **Jenkins: <JENKINS_NEW_VERSION>**
  - Retrieve Dockerfile and setup from GitHub

```
$ git pull
```

  - remove related containers that could be stuck

```
$ Docker-compose rm
```

  - Start and build containers:

```
$ . ./env/generate_docker_env.sh
$ Docker-compose up --build -d
```

  - Plugins and jobs will be upgraded (after nginx start).

- **Nexus**
  - remove related containers that could be stuck

```
$ Docker-compose rm
```

  - The nexus Image will be built and started

```
$ Docker-compose up --build -d
```

- **Nginx:**
  - remove related containers that could be stuck

```
$ Docker-compose rm
```

- Nginx Image will be upgraded to <NGINX_NEW_VERSION> and the container rebuilt and restarted

- **Jenkins plugins and jobs**
  - Update all Jenkins plugins

A4.2.4.b Slaves.

- **Launch Slaves AMI:**
  - Select launch instance: elastest-slave-basic-AMI
  - Apply the steps 2- 5 into the instance

- **Kernel (5min):**
  - Update and upgrade

```
$ sudo apt update
$ sudo apt upgrade
```

  - If unused packages:

```
$ sudo apt autoremove
```

  - Reboot

```
$ sudo reboot
```

- **Docker:**
  - All the containers will be stopped.

```
$ Docker stop $(Docker ps -a -q)
```

  - Docker Images will be cleared.

```
$ Docker rmi -f $(Docker images -q)
```

  - Docker will be upgraded in the host with:

```
$ sudo add-apt-repository “deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
$ sudo apt update
$ sudo apt-get install docker-ce=<DOCKER_NEW_VERSION>
```

- **Docker Compose:**
  - Run this command to download the latest version of Docker Compose:

```
$ sudo curl -L
https://github.com/docker/compose/releases/download/<docker-
compose_NEW_VERSION>/docker-compose-`uname -s`-`uname -m` -o
/usr/local/bin/Docker-compose
```

  - Apply executable permissions to the binary:

```
$ sudo chmod +x /usr/local/bin/Docker-compose
```

  - Test the installation.

```
$ Docker-compose --version
```

- **AWS cli**
  - uninstall old version

```
$ sudo apt-get remove awscli
```

- install new version

```
$ sudo curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o
"awscli-bundle.zip" &&  sudo  unzip awscli-bundle.zip
&&  sudo   ./awscli-bundle/install -i /var/lib/aws -b /usr/bin/aws
```

- **AMI creation:**
  - Select instance: elastest-slave-basic-AMI : (Figure 25. AWS EC2. Create Image)
  - Image / Create Image. (Figure 26. AWS EC2. Configuration of the Image)
    - **name:** elastest-slave-basic-AMI-v<new_version>
    - **description:** AAAAMMDD_Maintenance_Window

- **Jenkins**
  - In the Jenkins substitute old AMI with new AMI

A4.2.4.c ElasTest Nightly.

- **Kernel (5min):**
  - Update and upgrade

```
$ sudo apt update
$ sudo apt upgrade
```

  - If unused packages:

```
$ sudo apt autoremove
```

  - Reboot

```
$ sudo reboot
```

- **Docker:**
  - All the containers will be stopped.

```
$ Docker stop $(Docker ps -a -q)
```

  - Docker Images will be cleared.

```
$ Docker rmi -f $(Docker images -q)
```

  - Docker will be upgraded in the host with:

```
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
$ sudo apt update
$ sudo apt-get install docker-ce=<DOCKER_NEW_VERSION>
```

- **Docker Compose:**
  - Run this command to download the latest version of Docker Compose:

```
$ sudo curl -L
https://github.com/docker/compose/releases/download/<docker-
compose_NEW_VERSION>/docker-compose-`uname -s`-`uname -m` -o
/usr/local/bin/Docker-compose
```

  - Apply executable permissions to the binary:

```
$ sudo chmod +x /usr/local/bin/Docker-compose
```

- Test the installation.

```
$ Docker-compose --version
```

- **AWS cli**
  - uninstall old version

```
$ sudo apt-get remove awscli
```

  - install new version

```
$ sudo curl "https://s3.amazonaws.com/aws-cli/awscli-
bundle.zip" -o "awscli-bundle.zip" &&  sudo unzip awscli-
bundle.zip &&  sudo   ./awscli-bundle/install -i /var/lib/aws
-b /usr/bin/aws
```

A4.2.5. Test and Confirmation

[Test-Jenkins-01] Login in Jenkins
[Test-Jenkins-02] Run basic jobs.
[Test-Jenkins-02-01] Run job hello-world/mvn-hello-world
[Test-Jenkins-02-02] Run job hello-world/hello-world-Docker-image-pipeline
[Test-Jenkins-03] Plugins?

[Test-Nexus-01] Web interface (Login and query)
[Test-Nexus-02] Publish artifact: run Jenkins job: hello-world/private-mvn-release
[Test-Nexus-03] Retrieve artifact.

[Test-UserRegistry-01] Log In
[Test-UserRegistry-02] Regenerate access.

[Test-DockerSibling-01] Run job hello-world/hello-world-Docker-image-pipeline.
[Test-DockerSibling-02] Run job hello-world/pipeline-Docker-privateRegistry

[Test-AMI-01] After AMI update test reboot and Docker TCP ports (if not check Docker tcp procedure)

A4.2.6.Roll Back

A4.2.6.a Main Instance.

- The instance of the AWS EC2 will be switched off. (elastest-ci)
- A new instance of the AWS EC2 will be launched with the backed-up AMI with the same configuration and IP as the old one.
- Elastic IP will be assigned to the rolled back instance.
- The Docker Images will be rolled back (the Dockerfile recovered and the images recreated with the old values)

A4.2.6.b Slaves.

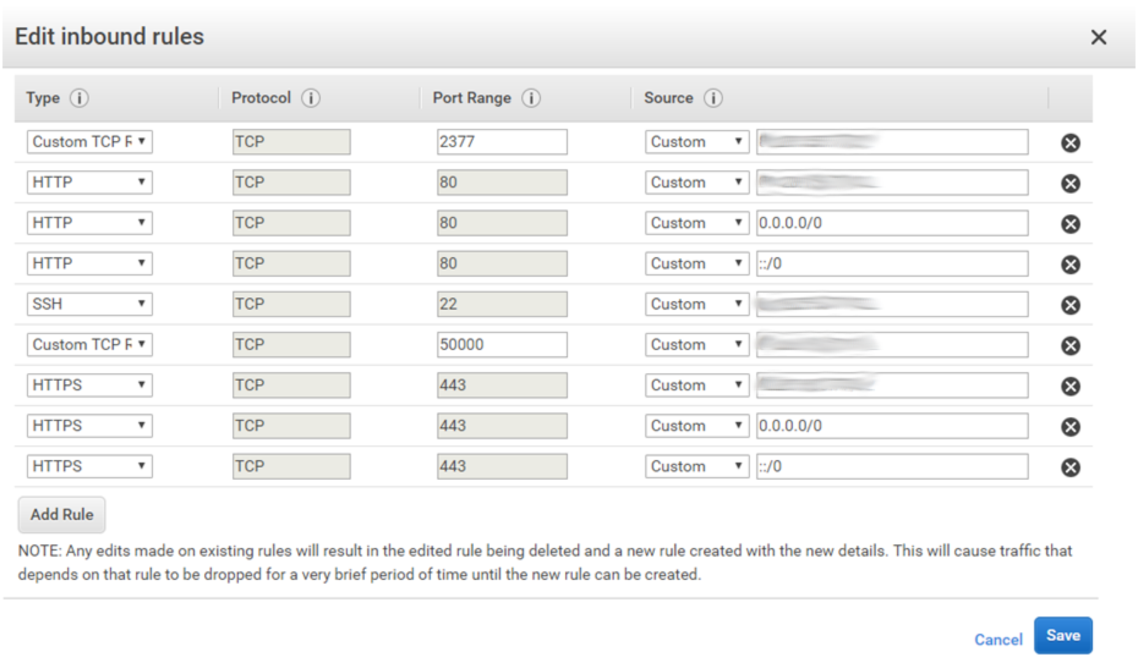
- New image wouldn't be saved so no extra actions required

A4.2.6.c ElasTest Nightly:

- The instance of the AWS EC2 will be switched off. (Secure-ElastestStack-v001)
- A new instance of the AWS EC2 will be launched with the backed-up AMI with the same configuration and IP as the old one.
- Elastic IP will be assigned to the rolled back instance.

A4.2.7. Open System and Result Notification.

- The instances will be configured to accept external requests



Edit inbound rules

Type	Protocol	Port Range	Source	
Custom TCP R	TCP	2377	Custom	[redacted]
HTTP	TCP	80	Custom	[redacted]
HTTP	TCP	80	Custom	0.0.0.0/0
HTTP	TCP	80	Custom	::/0
SSH	TCP	22	Custom	[redacted]
Custom TCP R	TCP	50000	Custom	[redacted]
HTTPS	TCP	443	Custom	[redacted]
HTTPS	TCP	443	Custom	0.0.0.0/0
HTTPS	TCP	443	Custom	::/0

Add Rule

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

Cancel Save

Figure 28. AWS enable inbound rules

A4.3. Results

A4.3.1.Table of results

Phase	Result	Time and duration
Back UP	SUCCESS / FAILURE / WARN	
Upgrade	SUCCESS / FAILURE / WARN	
Test and Confirmation	SUCCESS / FAILURE / WARN	
Rollback	NOT RUN / SUCCESS / FAILURE / WARN	

A4.3.2. Actions to be executed after upgrade

A4.3.2.a Main Instance

A4.3.2.b Slaves

A4.3.2.c ElasTest Nightly

A4.4. Logs

<if applies>

A4.5. Issues

Any issue that is detected and is suspected to be related to the upgrade should be registered here.

A5. ElasTest Jenkins Library API.

A5.1. Configuration methods:

setVerbose(boolean)
Overrides the Verbose configuration parameter value (false) of the Library

setVersion(String)
Overrides the Elastest Version configuration parameter value (“latest”) of the Library

setERE(String)
To select if and which version of the ERE component will be deployed and used on the running ElasTest instance

setContext(value)
Initializes the context that should be applied for the execution of certain steps and commands.

setMode(String)
Initialization of the parameter Lite just for personalization, for normal execution leave it empty. For lite execution initialize with ‘--lite’

setShared(boolean)
Overrides the ElasTest Shared configuration parameter value (false) of the Library

setAuthenticatedElastest(boolean)

Overrides the default ElasTest Authentication parameter configuration parameter value (false) of the Library. If true the launched ElasTest would be initialized with basic authentication.

setTestLink(boolean)
Overrides the default ElasTest TestLink parameter configuration parameter value (false) of the Library. The TestLink component will be launched on the new instance.

A5.2. Support functionalities for end-to-end testing:

String getIp()
Executed after the ElasTest start returns the IP address where the main component (ETM) is listening

String getPort()
Executed after the ElasTest start returns the port where the main component (ETM) is listening

String getEtmUrl()
Executed after the ElasTest start returns the full http connection URL where the ETM is listening (Example: http://172.0.18.10:8091)

Object getElastestMethods()
Returns an object that can execute the methods implemented by the library in the user defined stages.

String getElasTestUser()
Returns the user for the ElasTest Basic Authentication if the ElasTest is authenticated

String getElasTestPassword()
Returns the password for the ElasTest Basic Authentication if the ElasTest is authenticated

A5.3. ElasTest management functionalities:

boolean startElastest()
Executes the ElasTest platform with the options configured in the library (version, lite)

boolean elastestIsRunning()
Return (in the moment) while ElasTest is already running. Checking if elastest Platform container is running and if ETM is running too.

boolean waitElasTest()
Waits up to 900s to ElasTest to start up. Returns true if it has started, and false otherwise.

boolean elastestIsStuck()
This is a variation of the waitElasTest that detects if any the platform component has been launched but the rest of the platform hasn't.

connect2ElastestNetwork()
After the ElasTest platform is launched as a Docker container, if this method is executed inside another container the method adds it with the ElasTest network.

stopElastest()
Stop the ElasTest platform and all the components.

getApi()
After the ElasTest Platform is started, this method initializes the connection information for the ElasTest

pullERE(optional version)
In case the ERE is expected to be launched the pullERE method will pull the defined version of the component if selected and the latest if void.

pullERE(optional version)
In case the ERE is expected to be launched the pullERE method will pull the defined version of the component if selected and the latest if void.

testRemoteElastest()
If the library is configured to connect with an external ElasTest (nightly mainly) the library would try to connect with it instead of "startElastest".

A5.4. Pipeline encapsulation:

pipeline(body)
This method receives as parameter the user defined stages of the pipeline and encapsulates them between the necessary logic to select the kind of ElasTest execution and the management logic of the ElasTest. Resulting in a more complete pipeline with the start ElasTest and release ElasTest stages added to the user defined stages.