Title:

Continuous Integration Platform and Guidelines

Multi-cloud Execution-ware for Large-scale Optimized Data-Intensive Computing

H2020-ICT-2016-2017
Leadership in Enabling and Industrial Technologies;
Information and Communication Technologies

Grant Agreement No.:
731664

Duration:
1 December 2016
30 November 2019
www.melodic.cloud

Deliverable reference:
D5.05

Date:
31 May 2018

Responsible partner:
7bulls

Editor(s):
Michał Semczuk

Author(s)
Paweł Skrzypek

Approved by:
Tomasz Przeździęk

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables.html

Abstract/Executive summary

Cloud computing conquers the market of software products and services. In the fast-paced business environment, more and more companies see the cloud as an opportunity to stay competitive, to improve their IT operations and to decrease the costs. At the same time, as much as the business models change, a shift to the cloud requires a different approach to software development and delivery.

Continuous Integration/Continuous Delivery becomes a natural choice for cloud software development as it enables much faster testing, building, and deployment of the applications. It accelerates the process of software delivery and, foremost, thanks to integrated and automated process, all the versions are always built in the same way hence avoiding a risk of an error.

This document describes how Continuous Integration/ Continuous Delivery is used for development and maintenance of the Melodic project.

This project has received funding from
the European Union's Horizon 2020 research
and innovation programme under grant agreement No 731664

Document	
Period Covered	M12-18
Deliverable No.	D5.05
Deliverable Title	Continuous Integration Platform and Guidelines
Editor(s)	Michał Semczuk
Author(s)	Paweł Skrzypek
Reviewer(s)	Kyriakos Kritikos, Volker Foth
Work Package No.	5
Work Package Title	Integration and security
Lead Beneficiary	7bulls
Distribution	PU
Version	Final
Draft/Final	Final
Total No. of Pages	27

This project has received funding from the European Union’s Horizon 2020
research and innovation programme under grant agreement No 731664

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Index of Figures

Index of Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1 Introduction

The purpose of this document is to describe Continuous Integration/Continuous Delivery platform delivered in the project as well as supply guidelines for its usage.
This document is dedicated, among others, to development and testing teams, architects and all of those interested in the continuous delivery and evolution of big, multi-cloud systems.
The Continuous Integration/Continuous Delivery platform for the Melodic project was developed according to collected requirements which are also described in the deliverable.

1.1 PaaSage[1] on OW2[2]

The predecessor of Melodic – PaaSage was designed to provide support for the provisioning and deployment of dynamically adaptive cloud-based applications in a cloud technology-neutral way on top of multiple and heterogeneous cloud infrastructures. This enables reduction of the time and cost necessary to adapt the applications to different cloud providers, hence minimizing vendor lock-in scenarios. To reach this objective, the project drew on the benefits of the model-based approach to abstract away technical details. PaaSage has the ability to reason about application reconfiguration as well as conduct this reconfiguration across multiple clouds when the set of user-supplied requirements is compromised at runtime. The PaaSage codebase is hosted on the OW2 platform, where OW2 represents an independent community dedicated to the development of an open source industry-grade enterprise computing infrastructure software, including middleware, application platforms and cloud computing technologies. As a successor, Melodic benefits from this and is also synchronized with OW2. All the branches from its production environment are being contemporised with OW2 after each release. This allows to deliver the respective Melodic results on top of the PaaSage ones and enables the OW2 community to capitalise on them.
PaaSage used a simple Continuous Integration/Continuous Delivery platform based on the Jenkins[3] build server and various source code repositories. Melodic has extended and improved that approach to provide a complete Continuous Integration/Continuous Delivery platform, with a single repository for source code, an artifact repository, a docker images repository and a build/deployment server. Thanks to that, a state of the art software development environment has been built.

[1] https://paasage.ercim.eu/
[2] https://www.ow2.org/
[3] https://jenkins.io/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1.2 Structure of the document

A short description of the structure of this document is supplied below.
Chapter 2: Continuous Integration/Continuous Delivery Platform – elaborates the definition of terms used in the document, especially explaining what a Continuous Integration/Continuous Delivery platform is as well as what processes and components are involved in it.
Chapter 3: Review of currently available Continuous Integration/Continuous Delivery platforms – provides a short review on Continuous Integration/Continuous Delivery platforms available on the current IT market.
Chapter 4: Continuous Integration platform for Melodic – supplies a detailed description, including the main functionalities, of the built-from-scratch platform.
Chapter 5: Continuous Integration Platform guidelines – supplies a short manual on how the implemented platform can be utilised in the context of continuous integration.
Chapter 6: Summary – summarizes the content of this document and supplies directions for future work.

2 Continuous Integration/Continuous Delivery Platform

The implementation of a fully automated Continuous Integration/Continuous Delivery class solution significantly speeds up the process of software delivery and, above all, allows its repetition. Further, the delivered software version is always being built and implemented in the same, fully automatic way which minimizes the risk of making mistakes. On the other hand, Continuous Integration class solutions, which are responsible for building and testing software using unit tests, have been used for some time also in on-premise environments or in relation to a bare-metal infrastructure. Meanwhile, the adoption of Continuous Delivery solutions has become the norm in the cloud domain due to their simplicity and user-intuitiveness based on the cloud model. In traditional IT environments, it is usually not possible to create an infrastructure in a programmatic manner, as this approach is more difficult and complex, which contrasts the necessity of the Continuous Deployment process.
In parallel with the development of the cloud computing model, several other significant trends can be observed, including[4]:

- dissemination of the concept of a software-defined infrastructure - defined as a system where software controls computing hardware without significant human intervention;

[4] http://www.7bulls.com/wp-content/uploads/2017/12/multiCloud_article_7bulls.pdf

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- striving to shorten the average software delivery time by using agile methodologies and approaches which allow for producing and presenting results faster as well as enable finding potential bugs earlier;
- automating the software testing and implementation processes (this helps to lower the effort of testers and developers) and
- DevOps[5][6][7] - a relationship between people, processes and products that enables to continuously deliver revised and improved products to end users

Continuous Integration is a software development practice where members of a team integrate their work frequently. Usually, each person integrates at least daily, thus leading to multiple integrations even per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

A typical Continuous Integration/Continuous Delivery solution allows to execute a certain process, called Continuous Integration/Continuous Delivery process, comprising the following steps:

- Software build – from a specific branch of the version control system, the software is built by using a certain build tool (usually Maven[8] or Gradle[9]).
- Verification of the software – through the analysis of static code quality by SonarQube[10] or any other analogous software.
- Deployment on the test environment – automatic release of the software for the selected test environment.
- Performing unit tests – on the developed software, running in the test environment, tests are carried out to verify the behaviour of both methods and key software components.
- Performing automatic acceptance tests – which includes the following kinds of tests:
    - functional,
    - performance,
    - safety
- Placing the developed software in the repository – such a repository constitutes a dedicated storage for source code, artifacts and docker images.
- Implementation on selected target environment – automatic deployment of the developed software on the production environment, after successful acceptance tests.

Below, we describe the differences and similarities between Continuous Integration, Continuous Delivery and Continuous Integration/Continuous Delivery solutions:

Continuous Integration: short-lived feature branches, the team is merging to master branch multiple times per day, fully automated build and test process which gives feedback within 10 minutes; deployment is manual.

[5] https://www.gartner.com/it-glossary/devops
[6] https://aws.amazon.com/devops/what-is-devops/
[7] https://www.webopedia.com/TERM/D/devops_development_operations.html
[8] http://maven.apache.org/
[9] https://gradle.org/
[10] https://www.sonarqube.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Continuous Delivery: Continuous Integration + the entire software release process is automated, it may be composed of multiple stages, and deployment to production is manual.
Continuous Integration/Continuous Delivery platform: Continuous Integration plus Continuous Delivery and fully automated deployment to production in one.
In the following parts of this chapter, we present the architecture of a typical Continuous Integration/Continuous Delivery platform and then we analyse the main components of this architecture and their interactions.

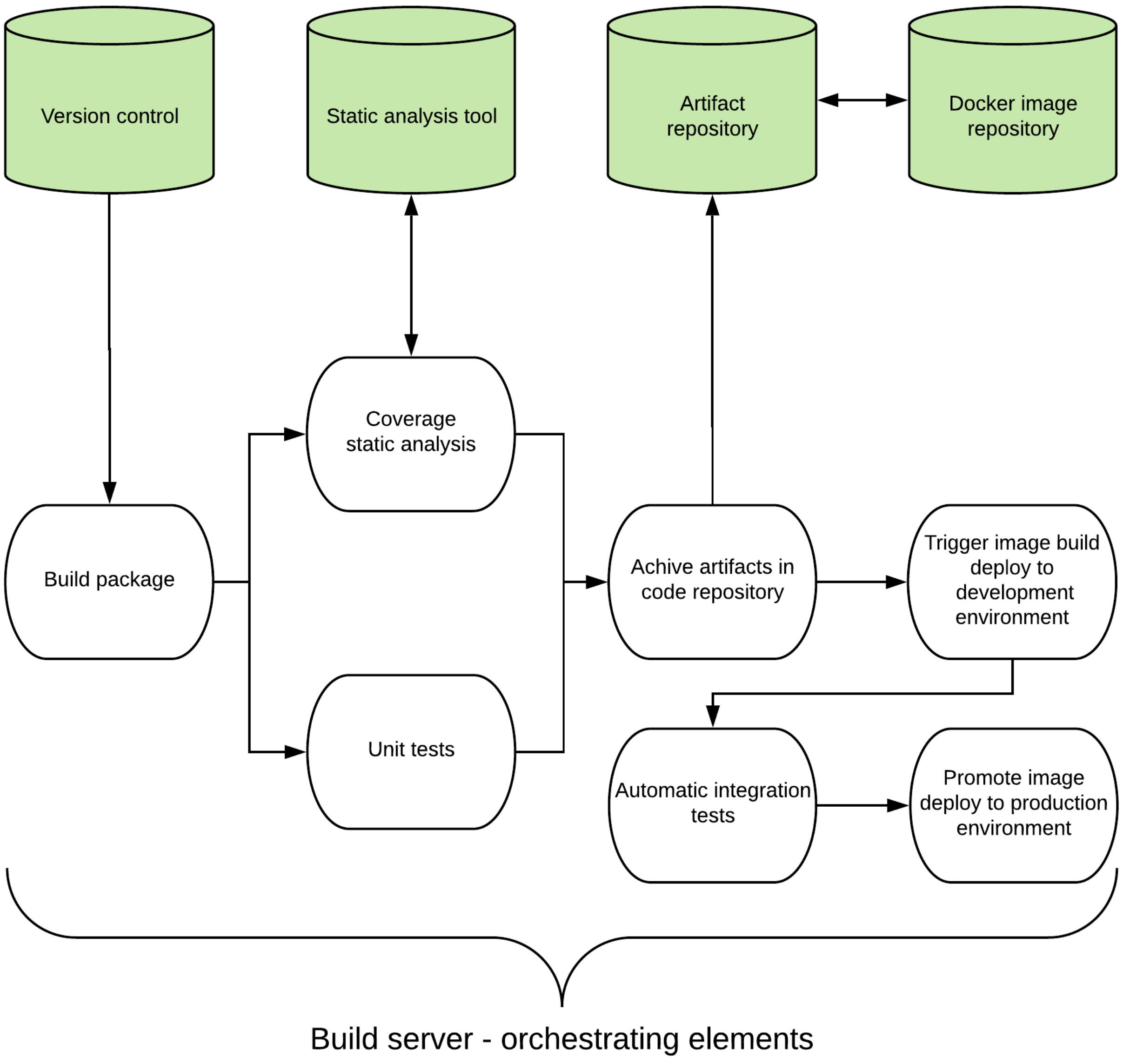
Version control
Static analysis tool
Artifact repository
Docker image repository
Coverage static analysis
Build package
Achive artifacts in code repository
Trigger image build deploy to development environment
Unit tests
Automatic integration tests
Promote image deploy to production environment
Build server - orchestrating elements

Figure 1 Typical architecture of a Continuous Integration/Continuous Delivery solution

2.1 Source code repository

A repository supports the management of source code and related documents, images and other elements that are part of a software project. It includes version-control software which supplies a database maintaining all revisions to an application realised by all the developers involved in its development. This allows developers to revert to a previous version, if necessary, as well as to review the changes made from one version to another.

2.2 Artifact repository

Such a repository enables to store a collection of binary software artifacts and metadata stored in a user-defined directory structure which is used by clients (such as, e.g., Maven) to retrieve binaries during a building process.
While the process of development is in progress, various kinds of artifacts can be created. These can be for example:

- Binary code – the simplest form of computer code or programming data. It is represented entirely by a binary system of digits consisting of a string of consecutive zeros and ones. Binary code is often associated with machine code in that binary sets can be combined to form raw code, which is interpreted by a computer or other piece of hardware[11].
- Compiled application – when programmers create software programs, they write the source code in a specific programming language. These source code files are saved in a text-based, human-readable format, which can be opened and edited by programmers. However, the source code cannot be run directly by the computer. For the code to be computer-processable, it must be converted from source code (a high-level language) into machine code (a low-level language). This process is referred to as code compilation[12].
- Deployable packages – packages that enable "one-click" deployment of code stored in the repository.

Maven, for example, stores artifacts in xml files which are called Project Object Models (POMs). The best repository for Maven is Nexus, which can proxy the main repository and can cache artifacts retrieved from remote repositories on a local server. Although source code control systems could be also used to store binary files, they have been designed most of the cases to be more optimal in the management of text files. In this sense, they can be used as a simple storage mechanism, if most of the releases are text-based and there is no need to store a large amount of binary data. Apart from Nexus, the following artifact repositories can be used instead:

- Cloudsmith Package[13]

[11] https://www.techopedia.com/definition/17052/binary-code
[12] https://techterms.com/definition/compile
[13] https://cloudsmith.io/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- Pulp[14]
- AWS ECR[15]
- Archiva[16]
- Artifactory[17]

It is better to use artifact repositories that are designed to store all kinds of files, including text and binary files, but as the other files are less important it highly recommended to store a source code which is about 99% of all the files. This can include, among others, compressed source codes, build results, and items, such as docker containers. In addition, such repositories usually not only store such artifacts but also help to manage them by offering various additional features, such as:

- Version control support – this enables to store metadata related to artifacts to facilitate their management, including their build date, their version number and their abbreviations.
- Access control – configure who can read, publish, overwrite, modify and download the artifacts stored.
- Promotion – snapshot artifacts with a short storage period on a server can be stored „near to developers" on a separate, „near-live" servers, repository, where only deployable artifacts appear. This should also include support for different version channels and transfer of artifacts between them.

2.3 Docker images repository

A docker image is a snapshot, or template, from whereof new containers can be started. It is a representation of a filesystem plus libraries for a given operating system (OS). A new image can be created by executing a set of commands contained in a Dockerfile.
A docker image repository is a place where docker images are stored, compared to the image registry which is a collection of pointers to these images. An image registry gathers resources about public, such as docker hub, and private repositories as well as about how to set up and manage docker image repositories.

2.4 Build server

Build server, also known as the Continuous Integration server, is a centralised, stable and reliable environment for building distributed programming projects.
The build server builds code from a specified branch (usually from the master branch). The changes to the master branch are added only through pull requests; it is not allowed to make a

[14] https://pulpproject.org/
[15] https://aws.amazon.com/ecr/
[16] http://archiva.apache.org/index.cgi
[17] https://jfrog.com/artifactory/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

direct commit to that branch. This increases predictability, forcing control over the different sources and enables signalling of problems and rapid notification of programmers in case of conflicts or missing dependencies. This helps, for instance, to ensure that the same dynamic link library is used in all compilations and does not lead to a failure during quality testing.

2.5 Automatic test execution

Automated software testing is the process in which testing tools pre-execute application test scripts before they are put into production. There are many tools that can be used to perform this, however, based on our experience in software development, we would recommend:

- Functional testing, as described in D5.06 "Test Strategy and Environment" [1], can be supported with SOAPUI[18], a very popular open-source tool for testing Web Services. It supports both SOAP and RESTful Web Services and can be installed on many operating systems
- JMeter[19] for load and performance testing. It is designed to measure and analyse the performance of applications by allowing to build and execute test plans. One of the advantages of using JMeter is the ability to set up a load test in a distributed manner [2].

The most important goals are automation and completeness. For the latter, a minimum set of (representative) test scripts shall be determined such that the complete coverage of the application behaviour is guaranteed. If unit testing consumes a significant portion of the quality assurance team (QA) resources, this process can be a good candidate for automation. Automated testing tools allow to perform tests, report results and compare such results with those of previous tests. Tests carried out with these tools can be run repeatedly, at any time.
This, of course, does not mean that there is no need of performing some manual tests, but they are "project specific" and the process is in details described in D5.06 deliverable "Test Strategy and Environment" [1]. For manual testing, we might take into consideration tools like TestLink [20] or HP QC[21].

2.6 Deployment server

A deployment server is a system that enables to distribute applications, configurations, and other resources to different instances (e.g., VMs, containers, etc.) in the fastest possible way.
Deployment server usually provides such features:

[18] https://www.soapui.org/
[19] https://jmeter.apache.org/
[20] http://testlink.org/
[21] https://software.microfocus.com/en-us/products/quality-center-quality-management/overview

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- **Executing device configuration rules** to automatically configure programs to work.
- **Executing deployment rules** that deploy packages.
- **Fault tolerance** automatically distributes the load among available servers if one server is not available.
- **Load balancing and scalability** automatically distributes the load among running Deployment Servers when more than one is configured.

2.7 Pipeline

A pipeline is a tool which enables the execution of the whole Continuous Integration/Continuous Delivery process by attempting to integrate various tools. In a pipeline, there is a set of elements (tools or other kinds of software) which can be triggered based on events and can produce an output that can be consumed by the next element.
A pipeline breaks down the software delivery process into stages. Each stage verifies the quality of a new feature from a different angle to guarantee the new functionality at the end of the entire pipeline and prevent errors from affecting users.

2.8 Reasons for developing the platform

Main reasons for developing a Continuous Integration/Continuous Delivery platform:

1. Continuous Integration reduces the risks associated with integration at the very end of the project – errors, incompatibility of interfaces, difficulty to estimate time to assemble the whole solution.
2. Continuous Integration makes it easy to fix errors – quick detection makes it easier to locate the cause and enables to more rapidly correct errors – developers know what has been recently modified and which version has worked correctly.
3. Continuous Integration protects against unexpected situations resulting from differences between the development and production environment (e.g., in case different runtime environments of the same language have been employed in these two environments - production and development).
4. Continuous Integration enables the demonstration of the application as well as consultation with the client at any time thanks to the constant availability of the last working version of the software on a pre-production environment, as a kind of new environment which enables to show some software features to the client and retrieve feedback from him/her.
5. Continuous Integration facilitates refactoring as after each change, everyone can quickly check if the program has a stable and robust operation (i.e., it is being built and tested properly), just like before the change.

6. Continuous Integration removes from developers the obligation to perform many repetitive, activities.

3 Review of currently available Continuous Integration/Continuous Delivery platforms

This section of the document presents a short review of those Continuous Integration/Continuous Delivery platforms that are the most popular and widely used by developers (now and in the past). As there is no such thing as complete Continuous Integration/Continuous Delivery solution available on the market, we will analyse key elements of the typical solution. These are the most advanced tools available on the market with functionality and features superior to their competitors. Also, some other useful tools dedicated to the Continuous Integration/Continuous Delivery process are described here as they enable to make this process more efficient.
The most popular Continuous Integration/Continuous Delivery solutions are those built on Jenkins[22] or solutions which belong to the Atlassian stack[23].

3.1 Criteria for choosing the Continuous Integration/Continuous Delivery platform

This section of the document focuses on criteria used to choose the best suitable platform and also show a comparison of tools mostly used by developers.

3.1.1 Criteria used for choosing project's platform

Features that could well characterise a prominent Continuous Integration/Continuous Delivery environment are the following:

1. One, common code repository, containing everything you need to build and run the system.
2. Automatic build support – no pop-ups and need to conduct manual activities e.g., file copying.
3. Automatic tests while building – even when the program is compiled and can be run, this does not mean that everything works perfectly. In this respect, it is not only

[22] https://jenkins.io/
[23] https://www.atlassian.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

necessary to perform automatic testing but also to check both the test results as well as the percentage of test coverage of a certain code.

4. Support for everyday submission and merging of code by developers. Integration means also communication – programmers see each other's work and this is how they can all know what is the status and progress of their work.
5. Code inspections – code reviews carried out by other programmers. The basic value is the exchange of knowledge which can lead to code improvement.
6. Build process triggered by every code change in the repository and immediate response to failures.
7. Complex history log – allows getting back in time to see any change.
8. Built-in solution for continuous software delivery – provides a complete process at every stage: from code creation to implementation. Such solution does not need to configure the server for continuous integration and user management system or to synchronize repositories.
9. Search with code recognition – Semantic search with code recognition analyses the code syntax and organizes the results according to the code compatibility with the search criteria.
10. Pre-integration with other elements of Continuous Integration/Continuous Delivery platform – allows for integration with other elements of the whole platform in an "out-of-the-box" manner, which means there is no need to install special plugins or some extra coding.

The above criteria will be used to evaluate every part of the complete Continuous Integration/Continuous Delivery platform

3.2 Source code repositories Git, SVN, BitBucket

3.2.1 Git

Git[24] – a distributed version control system. It was created by Linus Torvalds[25] as a tool supporting the development of the Linux[26] kernel[27]. Git is an open source software and has been released under the GNU GPL[28] version 2. The first version of the Git tool was released on April 7, 2005 to

[24] https://git-scm.com/
[25] https://en.wikipedia.org/wiki/Linus_Torvalds
[26] https://www.linux.com/
[27] https://www.kernel.org/
[28] https://opensource.org/licenses/gpl-3.0.html

replace the BitKeeper[29] version control system, previously used in the development of Linux[30], which was not open source.
The main features of Git include:

- Good support for a branched software development process – several algorithms for combining changes from two branches are available, as well as the ability to add custom algorithms.
- Off-line work – each programmer can have his/her own copy of the repository, to which he/she can save changes without connecting to the network; then, changes can be exchanged between local and remote repositories.
- Support for various existing network protocols – data can be exchanged via HTTP(S), FTP, rsync, and SSH.
- Effective work with large projects – The Git system, according to Torvalds' assurances, and also according to the Mozilla foundation[31] tests, is some orders of magnitude faster than some competing solutions[32].
- Each revision is a picture of the entire project – unlike other version control systems, Git does not remember just changes between subsequent revisions, but their complete images. This can be beneficial when the opportunity to rapidly take one revision and enforce it is rather better than waiting for the system to first reproduce it and then be able to exploit it.

3.2.2 SVN

Subversion[33] (also known as SVN) – a version control system that was created to replace CVS[34]. By assumption, SVN is in most cases functionally compatible with its predecessor. However, compatibility was abandoned due to the need to introduce new features. SVN is a free and open software under the Apache license[35].
SVN main features include:

- History of directory and file name changes – The lack of history of introduced changes of catalogue names was one of the most frequently criticized CVS defects. Subversion saves not only the contents of the file and information about whether the file exists but also the location of the file in directories, its copies, as well as its renaming. It also allows the properties of a given file or directory, like its executable flags, to be remembered.
- Changes are atomic transactions – Changes to several files or directories only take effect if all modifications have been successfully completed. This was another improvement over

[29] http://www.bitkeeper.org/
[30] https://www.linux.org/articles/
[31] https://www.mozilla.org/en-US/foundation/
[32] http://jultika.oulu.fi/files/nbnfioulu-201503311195.pdf
[33] http://subversion.apache.org/
[34] http://cvs.nongnu.org/
[35] http://www.apache.org/

CVS which was enforcing the partial changes of a certain atomic transaction unit. In particular, in CVS, such a situation was possible as some of the files could be updated and some not, e.g., in the event of a network connection failure.

- Ability to use the Apache Server – Subversion can use an HTTP-based protocol. In particular, it exploits WebDAV/DeltaV for network communication, as well as the Apache web server for providing access to the network on the repository site. This gives Subversion an advantage over CVS and introduces important functions, such as:
  - user authentication and authorization,
  - compression of transmitted data,
  - basic access to the repository for free.
- Available stand-alone server – Subversion allows accessing the repository through a dedicated server, independent of the (Apache) http server. The main benefit and feature is that such a server runs as a dedicated service or a separate daemon. It offers basic user authentication and authorization. It also allows creating encrypted connections.
- Quick creation of branches and tags – In contrast to CVS, where the addition of branches and tags for organisations could be time-consuming, in SVN these operations are based on fast copying – where copies take up a small, fixed space.

3.2.3 BitBucket

BitBucket[36] – it is a web-based hosting solution for projects that desire to use a revision control system. Developed for professional teams, BitBucket enables users to code, manage, and collaborate on Git projects.
Its main features include:

- Git repository hosting – includes a git repository
- Build integration – easily shows build results with a simple pass or fail icon
- Pull requests – facilitates code reviews by other developers, resulting in higher quality code and an opportunity to share knowledge amongst the developers.
- Diff views – enables to see what is changed between two versions of code which is under review.
- Branch permissions – provides granular access control to branches for the dev team, ensuring the right people can make the right changes to your code.
- JIRA software integration – enables to create a branch from within a Jira Software issue and set up triggers to transition between status codes when reviewing or merging code.

[36] https://bitbucket.org/

Table 1 Source code repositories - scoring

Req. Id		Git	SVN	BitBucket
1	One and common code repository	X	X	X
2	Automatic build support	X	X	X
3	Automatic tests while building	N/A	N/A	N/A
4	Support for everyday submission and merging of code	X	X	X
5	Code inspections	X	X	X
6	Build process triggered by very code change	X	X	X
7	Complex history log	X	X	X
8	Built-in solution for continuous software delivery	-	-	X
9	Search with code recognition	-	-	X
10	Pre-integration with other elements of Continuous Integration/Continuous Delivery platform	-	-	X
	Summary	6	6	9

As shown in above

Table 1, Bitbucket seems to prevail as it covers almost all the criteria.

3.3 Artifact repository solutions

Nexus[37] – Nexus is a tool to manage the artifacts of software required for development. If a software is being compiled, the build process can draw dependent artifacts from the Nexus repository and can publish the built artifacts on this repository, creating a new way to share artifacts even across organisations. The central repository always serves as a convenience for programmers, but this

[37] http://nexus5001.org/

should not be the sole way of working as artifacts could reside in the premises of an organisation in a single place.
Bamboo – Continuous Integration/Continuous Delivery server which allows the division of code in a hierarchy of dependent components while also enables running the build process in parallel to guarantee its faster completion. When it is used as an artifact repository, it, among others, enables:

- tightening the feedback loop by eliminating duplicate checkouts and compilations
- ensuring that all tests are executed against the same revision number or change list
- verifying that the whole application, and not only its packages, behaves as expected when unpacked and deployed in a distributed manner[38]

Table 2 Artifact repositories - scoring

Req. Id		Nexus	Bamboo
1	One and common code repository	X	X
2	Automatic building support	X	X
3	Automatic tests while building	N/A	N/A
4	Support for everyday submission and merging of code	X	X
5	Code inspections	X	X
6	Build process triggered by every code change	N/A	N/A
7	Complex history log	X	X
8	Built-in solution for continuous software delivery	X	X
9	Search with code recognition	X	X
10	Pre-integration with other elements of Continuous Integration/Continuous Delivery platform	-	X
	Summary	7	8

As both repositories chosen for comparison are applicable for the platform, they are picked to be used together but will be responsible for different kind of tasks accordingly to their design.

3.4 Build server

The following tools have been considered for a Continuous Integration/Continuous Delivery platform for Melodic to choose from:

[38] https://www.atlassian.com/blog/archives/artifact-sharing-build-automation-bamboo

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3.4.1 Jenkins

Jenkins – a self-contained, open source automation server that can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.
Its main features include:

- It is hosted internally
- Building, deploying or launching can be conducted in an asynchronous manner
- Easily defined simple and complex pipelines through a DSL in a Jenkinsfile
- Pipeline as code provides a common language to help teams (i.e., Dev and Ops) to work together
- Easily share pipelines between teams by storing common "steps" in shared repositories
- There is a supply of a suitable interface that makes it easy to visualise the progress across an entire pipeline
- Pipelines are long-lasting and can survive infrastructure outages
- Built-in support for Git/GitHub branches

3.4.2 Travis

Travis[39] – a FOSS service (Free and Open-Source Software) that allows building, testing and deploying projects hosted on GitHub. It is completely free for public repositories. For private ones, there is an option to purchase a subscription.
Its main features include:

- Simple to configure a range of environment versions and settings in a simple YAML file
- Can detect the language being used based on build configuration files
- Supports specifying private environment variables and encrypted credentials, so that deployments can be safely automated
- Free cloud-based hosting that requires no maintenance or administration
- Multiple test environments for different runtime versions which supports testing for different versions of the same runtime
- No dedicated server required

3.4.3 Bamboo

Bamboo – is a continuous integration and continuous deployment server. It creates multistage compilation plans, sets triggers that start compilation after approval, and assigns agents to critical build and deployment processes.

[39] https://travis-ci.org/

Its main features include:

- Central management server which schedules and coordinates all work associated with development.
- Bamboo allows using Docker containers to create build agents. Using Docker agents enables to run multiple remote agents on the same host without conflicting requirements. It makes it easier to duplicate and distribute changes to build agents and to use scripts for creating and maintaining agents.
- Bamboo allows to automatically detect and build new branches, merge branches together when tests pass and continuously deploys code to staging and production servers based on branch name.
- Bamboo can be integrated with Amazon S3 for unlimited storage.
- Bamboo provides a web front-end for configuration and for reporting the status of builds.

Table 3 Build servers - scoring

Req. Id		Jenkins	Travis	Bamboo
1	One and common code repository	N/A	N/A	N/A
2	Automatic building support	X	X	X
3	Automatic tests while building	X	X	X
4	Support for everyday submission and merging of code	X	X	X
5	Code inspections	N/A	N/A	N/A
6	Build process triggered by every code change	X	X	X
7	Complex history log	X	X	X
8	Built-in solution for continuous software delivery	-	-	X
9	Search with code recognition	N/A	N/A	N/A
10	Pre-integration with other elements of Continuous Integration/Continuous Delivery platform	-	-	X
	Summary	5	5	7

Based on the presented criteria Bamboo was chosen as a build server for the project.

The below

Table 4 shows how the tools described above can be used as a part of a Continuous Integration/Continuous Delivery platform.

Table 4 Tools and their application

	GIT	SVN	Nexus	Bitbucket (Atlassian stack)	Jenkins	Travis	Bamboo (Atlassian stack)	Docker registry
Build server					X	X	X	
Automation server			X				X	
Code repository	X	X		X				
Docker image repository								X
Artifact repository			X				X	
Summary	1	1	2	1	1	1	3	1

4 Continuous Integration platform for Melodic

This chapter analyses the Continuous Integration platform chosen for Melodic.

4.1 Chosen platform

For the Melodic project, the Atlassian stack has been chosen (BitBucket and Bamboo). We based this decision on a research made on the market for available tools dedicated to Continuous Integration/Continuous Delivery process, as well as on the evaluation results of these tools. The respective evaluation results (according to the criteria described in section 3.11) are depicted in

Table 1, Table 2, and
Table 43.
Based on the evaluation results we have selected BitBucket, Bamboo and Nexus as the elements of Continuous Integration/Continuous Delivery platform.
BitBucket has been chosen as a source code repository as it wins 9 out of 10 categories.
Nexus as an external artifact repository because it also supports server automation and is designed to be used among different build plans, Bamboo for the internal artifact repository to be used within single build plan.
Bamboo as a build server for the reason it has a built-in solution for continuous software delivery.
Last, but not least, it shall be noted, that the whole stack was given to the Melodic project by Atlassian without any charges.

4.2 Docker images repository

The Docker image repository was built with the use of a Docker Registry. Any committed and pushed change to the source code of project's components triggers a new Bamboo build which in consequence (when successful) creates new docker images. Those images are then pushed to the Docker Registry where the latest versions for snapshots and all versions for release builds are kept.
The management of created docker images is being handled by the Docker Swarm - the configuration of so-called services is kept within a single YAML file for all the project's components.
The deployment is being conducted with a .ssh script which incorporates the following actions:

- Downloading of software dependencies (Docker CE, Docker Swarm, Cloudiator Client);
- Setting up volumes;
- Setting up the configuration files of Melodic's components (i.e., IP addresses);
- Installing maintenance tools (i.e., scripts that help to maintain the Melodic from an operational point of view);

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- Deploying the Melodic stack using Docker Swarm.

4.3 Implementation of Continuous Integration Platform

The implementation of the platform is supposed to be as simple as possible. Bamboo includes a sensor on the main (master) branch which is croned to check for any pushes every minute. Each branch[40] in Bitbucket is divided into eight repositories [3]:

- Upperware with:
  - Solvers
  - Adapter
  - Generator
  - CDO client
  - CDO server
- Camel with:
  - Camel
  - Paasage-pom
- Integration with:
  - Emule[41]
  - Camunda[42]
  - Cloudiator client
  - Definitions of interfaces
- Meta-data schema
- MUI
- Security
- Social network
- Utils

Bamboo picks only those modules which will be used for the relevant building process. Then, it can store the built one as an artifact in Nexus or make a docker image in case of deploying the Melodic platform. It is also ready to deploy the use-case applications automatically, but this feature has not been yet exploited.

The base configuration of the Melodic platform is stored in a file named as *melodic.yml*[43] (docker compose form to be exploited by developers).

[40] https://bitbucket.7bulls.eu/repos?visibility=public
[41] http://www.emule-project.net/home/perl/general.cgi?l=20
[42] https://camunda.com/
[43] https://bitbucket.7bulls.eu/projects/MEL/repos/utils/browse/melodic_properties/dev/docker/melodic.yml

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

5 Continuous Integration Platform guidelines

There are many articles and best practices guidelines that are mainly targeting developers after the respective Continuous Integration/Continuous Delivery solution has been set up and executed. The most common and important practices are described below and focusing on the smooth process of building and deploying the code[44].

One of the most important practices in implementing continuous integration is to encourage small changes. Developers should practice crumbling their work into small parts and commit them as quickly as possible[45]. Special techniques, such as branching by abstraction[46] and flags for features[47] (described later in this section) help to protect the master branch from undesired and untested code changes.

Iterative and small changes minimise the possibility and impact of integration problems. By engaging them in a shared branch at the earliest possible stage, and then continuously throughout the whole development period, integration costs get reduced, and un-associated work is regularly synchronized, which also allows ongoing changes to be observed more often.

Another valuable suggestion focuses on the time spent on building and deploying the code, as it also maps to a process repeated quite often. If it takes too much time for developers to go through it, it generates high, undesired costs by using more resources for keeping an eye on the process, apart from the computational resources devoted to the execution of the building process.

Once the part of the code is delivered and submitted, it needs to be verified; this is when the time for the testing arrives. As the builds and tests are often repeated, it is highly desired to make them automated and pipelined, such that the execution time of the frequently repeated steps is minimised. Another recommendation for testing is to split tests into smaller scenarios and, if possible, to run them in parallel. If testing is divided into smaller steps, it reduces execution time and speeds-up the whole process. This is called agile testing[48] [1] [2] [4] [5].

The typical flow in such a platform is shown in Figure 2 below.

[44] http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/2013-14/lecturas/10_Fowler_Continuous_Integration.pdf
[45] http://ieeexplore.ieee.org/abstract/document/6802994/
[46] https://martinfowler.com/bliki/BranchByAbstraction.html
[47] https://martinfowler.com/articles/feature-toggles.html
[48] http://ieeexplore.ieee.org/abstract/document/5261055/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

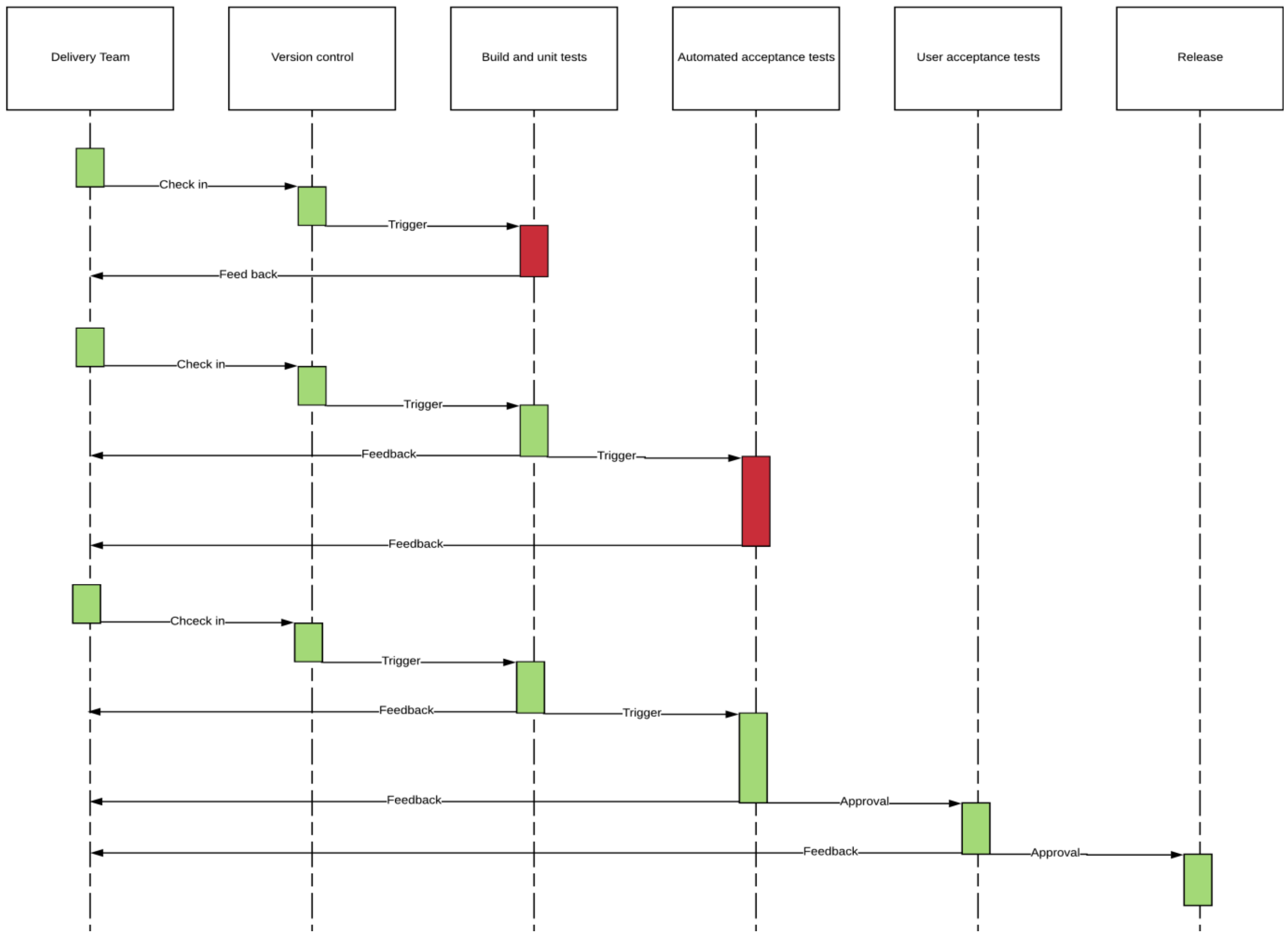

Figure 2 Continuous Integration/Continuous Delivery flow between components

6 Summary

To achieve continuous integration on its best level, a Continuous Integration Platform needs to map to a repetitive integration process which enables automatic execution of tests and release of the software product to production environment anytime. As such, the platform is a kind of general tool which realises a process that can incorporate a testing activity.
Furthermore, as described in this deliverable, such platform assists the development process in many ways by making it easier, quicker, and more user-intuitive, towards delivering the respective software product to the end user[49] [5].

[49] http://ieeexplore.ieee.org/abstract/document/4599493/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

The typical architecture of a Continuous Integration/Continuous Delivery solution was analysed along with the components that it incorporates such as:

- The source code repository
- The artifact repository
- The Docker image repository
- The build and deployment server

The main goal of work described in this document was to review available solutions, choose the best ones and to develop the Continuous Integration/Continuous Delivery platform for the Melodic project.
Based on the current market offering as well as certain evaluation criteria described in chapter 3.11 (chosen by the development team, that has many years of experience in software development), the Continuous Integration/Continuous Delivery solution for Melodic was formulated by selecting a small set of the best tools available on the market. Furthermore, the implementation and configuration details for this solution were supplied.
Although the Continuous Integration/Continuous Delivery platform developed for the Melodic project meets the main Continuous Integration/Continuous Delivery requirements set in the chapter 3.11, it still could be improved by, e.g., integrating additional tools. Such tools could help to further automate the tests and builds, increase speed-up of the application deployment process as well as to help and speed-up the collaboration and communication between individuals and teams involved in the process. However, such work requires more research and study before it can be realised.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

7 References

[1] K. Materka, D5.06 "Test Strategy and environment".

[2] M. Jakubczyk and M. Prusiński, D5.10 "Quality Assurance Guide".

[3] F. Zahid, "D2.2 "Architecture and initial feature definitions"".

[4] P. Skrzypek, D5.04 "Integration & testing requirements".

[5] E. Bańkowska, D5.07 "Integration release and initial test environment".

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664