DITAS
Data-intensive applications Improvement by moving daTA and computation in mixed cloud/fog environmentS

D5.1 Specification of integration and validation testbed and Cloud platform for DITAS

Project Acronym	**DITAS**
Project Title	Data-intensive applications Improvement by moving daTA and computation in mixed cloud/fog environmentS
Project Number	731945
Instrument	Collaborative Project
Start Date	01/01/2017
Duration	36 months
Thematic Priority	ICT-06-2016 Cloud Computing
Website:	http://www.ditas-project.eu

Dissemination level: Public

Work Package	**WP5 Real world case studies and integration**
Due Date:	M11
Submission Date:	30/11/2017
Version:	1.0
Status	Final
Author(s):	Aitor Fernández (IDEKO), Jon Kepa Gerrikagoitia (IDEKO), Peter Gray(CS), Grigor Pavlov (CS), Andrea Micheletti (OSR), Mariet NouriJanian (FCSR) Paola Aurucci (FCSR), Eleonora Lavalle (FCSR)
Reviewer(s)	David García Pérez (ATOS), Pierluigi Plebani (POLIMI), Vrettos Moulos (ICCS)
Licensing Information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/

This project has received funding by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731945

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
01	2017/11/14	Initial version with introduction to all sections	Aitor Fernández (IDEKO)
02	2017/11/16	Added the cloud Test bed section	Peter Gray(CS), Grigor Pavlov (CS)
03	2017/11/16	Completed sections 1, 2 and 3	Aitor Fernández (IDEKO)
04	2017/11/19	E-Health use case added	Andrea Micheletti (OSR), Mariet NouriJanian (FCSR) Paola Aurucci (FCSR), Eleonora Lavalle (FCSR)
05	2017/11/20	Industry 4.0 use case added	Aitor Fernández (IDEKO)
06	2017/11/27	Changes to all sections applied after review	Aitor Fernández (IDEKO)
07	2017/11/28	Changes to section 5.2 applied after review	Andrea Micheletti (OSR)
1.0	2017/11/30	Final for submission	David García Pérez (Atos)

Contents

List of Figures

List of tables

Executive Summary

This deliverable has two main pillars, 1.- define common practices for integration and validation of the outcomes of the project, and 2.- detail the two case studies the project will make use of to demonstrate the results.

Regarding the former, GitHub will be the base code repository for the project, where the project already owns an organizational account. Over GitHub [1], the trunk-based development branching policy will be applied, as it is considered the most suitable policy according to the project characteristics. Also, GitHub's issue reporting tool will be adopted, as it is fully integrated with GitHub's features. In the Continuous Integration (CI) field, Jenkins [7] will be the platform over which the CI cycle will be defined. Jenkins will be extended with specific plugins to cover several validation stages, like running tests, measuring the code quality or deploying software. Jenkins Pipelines will be fully adopted to define the CI cycle where tools like *Maven*, *npm* and *Ansible* will be extensively used. The test bed to support the demonstrators will be deployed over CloudSigmas's (CS) infrastructure where flexibility is one of the main features. There will be three environments to cover the deployment cycle: Development, Staging and Production. Every component must go over the three environments. The step from Development to Staging will be done automatically using Ansible, while the step from Staging to Production will be fired manually, using Ansible as well. Automated validation can be carried out in the Staging environment while validation from end-users must be carried out in the Production environment.

Then the two uses cases are introduced. With respect to the Industry 4.0 use case, IDEKO's Smart Box will be the platform for data gathering and edge computing. The box is a small industrial PC with data gathering capabilities along with a dedicated space for container deployment. Respecting the final application, an advanced technical service that improves the way the machine incidences are faced when they occur is going to be developed for demonstrator purposes. The application will join together the concepts of predictive maintenance, failure detection or historical machine behavior analytics, applying DITAS ideas like running data reduction techniques at the edge, or combining near-real time machine data with historical patterns.

Finally, with respect to the e-Health use case, the need to properly manage patient's personal data is described along three scenarios. In the first one, the main actor is the medical doctor, who needs patients personal to perform health care activities. That information can be stored in a trusted cloud inside the hospital. The second scenario introduces a second hospital, where more actors need to access the same information in an aggregated way for analytic purposes. In the third scenario, the medical information coming from different hospitals is sent to an Untrusted Cloud that stores it encrypted and anonymized, making it available for researchers.

The idea behind the e-health case study is that the DITAS platform manages the data movement and computation between cloud and edge in order to make available data and information to the involved actors.

1 Introduction

1.1 Purpose

This deliverable puts the focus on the integration and validation stages of the framework developed in the project. The amount of components to be developed in the technical work packages and the number of partners that will take part in the development process requires some previous agreements on concepts like the code repository to be used, the branching model, as well as some basic guidelines on how to structure the code. To ensure the code quality, metrics regarding unit and integration test must be set, and this, below the umbrella of Agile Development, needs to be defined within a Continuous Integration cycle. The document defines the whole Continuous Integration cycle to be applied during the project. This cycle will have 2 major objectives: 1. ensuring the product quality and 2. speeding up building and deployment stages.

This is just to ensure the homogeneity and quality of the developed components. The validation of the whole system must be done over real world use cases. For that, having a Cloud Testbed where deploy the framework is mandatory. DITAS framework relies on a complete infrastructure that needs to be deployed in order to run it. That infrastructure is as well detailed in this document.

Regarding the use cases, in D1.1 the use cases were introduced in a high level approach. This document gives technical details about both of them, adopting the terms of DITAS framework and detailing how the DITAS outcomes will be demonstrated using real world problems in very distinct contexts, e-Health and Industry 4.0.

This deliverable is structured as follows. Sections 2 and 3 details development and Continuous Integration related practices and strategies with regard to components development, testing and deployment. Section 3 describes the cloud test bed that will support the deployment of all framework components. The test bed will be the base for the developments, staging and production environments. Finally, section 5 describes the two use cases where DITAS results will be demonstrated.

1.2 Glossary of Acronyms

Acronym	Definition
API	Application Program Interface
BPMN	Business Process Model and Notation
CI	Continuous Integration
CMMS	Computerized Maintenance Management System
CMT	Code management tool
CNC	Computerized Numerical Control
CT	Computerized Tomography
DSL	Domain Specific Language

EHR	Electronic Health Record
ERP	Enterprise Resource Planning
EU	European Union
GDPR	General Data Protection Regulation
IaaS	Infrastructure as a service
ICD	International Classification of Diagnosis
JDK	Java Development Kit
KVM	Kernel-based Virtual Machine
MB	Megabyte
MES	Manufacturing Execution System
MHz	Megahertz
ML	Machine learning
MRI	Magnetic Resonance Imaging
MTTF	Mean Time To Failure
MTTR	Mean Time to Repair
NIC	Network Interface Controller
NUMA	Non-Uniform Memory Access
PLC	Programmable Logic Controller
UI	User Interface
VCS	Version Control System
VDC	Virtual Data Container
VLAN	Virtual Local Area Network
WP	Work Package

Table 1: Acronyms in alphabetical order

2 Development

2.1 Code Repository

There are many benefits of using a version control system (VCS) for the project. Every member of the team has a copy of the latest version and can work freely, merging the changes over a common version at any moment and making easy to manage any conflict that appears.

For DITAS, given the open-source nature of the project, GitHub [1] will be used as repository. GitHub is a web-based Git [2] version control repository. GitHub offers the distributed version control and source code management functionalities of Git as well as some additional powerful features. Moreover, provides access control layer and several collaboration features such as bug tracking, feature requests, task management or wikis.

DITAS has already an organization entity created in GitHub where all the repositories for software components will be stored. It can be found here: https://github.com/DITAS-project.

It is important to note that due to the nature of GitHub, in DITAS every component must have its own repository. The following are some basic rules for repository naming:

- Every component should be prefixed with the WP it belongs to, e.g.: *wp3-vdc-resolution-engine*
- *The separator must be a hyphen*
- Only lowercase letters must be used

2.2 Branching Policy

Git is simply a tool that can be used in very different ways. Branching policies tries to give some insights on how to use it in an efficient manner. There are several branching policies for Git, being perhaps GitFlow [3] and Trunk-based development the most commonly used policies. Github itself proposes the Git Flow, which according to them is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly. Trunk-based development simplifies the workflow in order to speed-up developments, especially when a project is starting and lengthy discussions and bureaucracy want to be avoided. Some other branching policies has become famous as well, like the "*Successful Git branching mode*l" [5] proposed by Vincent Driessen on 2010 or more recent proposals like this from Lucian Sabo [6] on 2016, more adopted for agile development. A basic internet search shows a high controversy on which is the best model, something that is highly dependent on the underlying project, the development methodology used on it, the size of the team and some other factors.

In DITAS, as agreed on D1.1, Agile Development methodology will be adopted. The development teams will be commonly small and usually composed by developers belonging to the same company. Having this into account, as well as the two iterations to be carried out during the project which implies developing - deploying and testing in a continuous way, **trunk-based development** is considered the most suitable workflow for the project and will be adopted.

In the trunk-based development model, all developers work on a single branch with open access to it. In GitHub, it will be named *master*. Developers commit code to it and run the committed code.

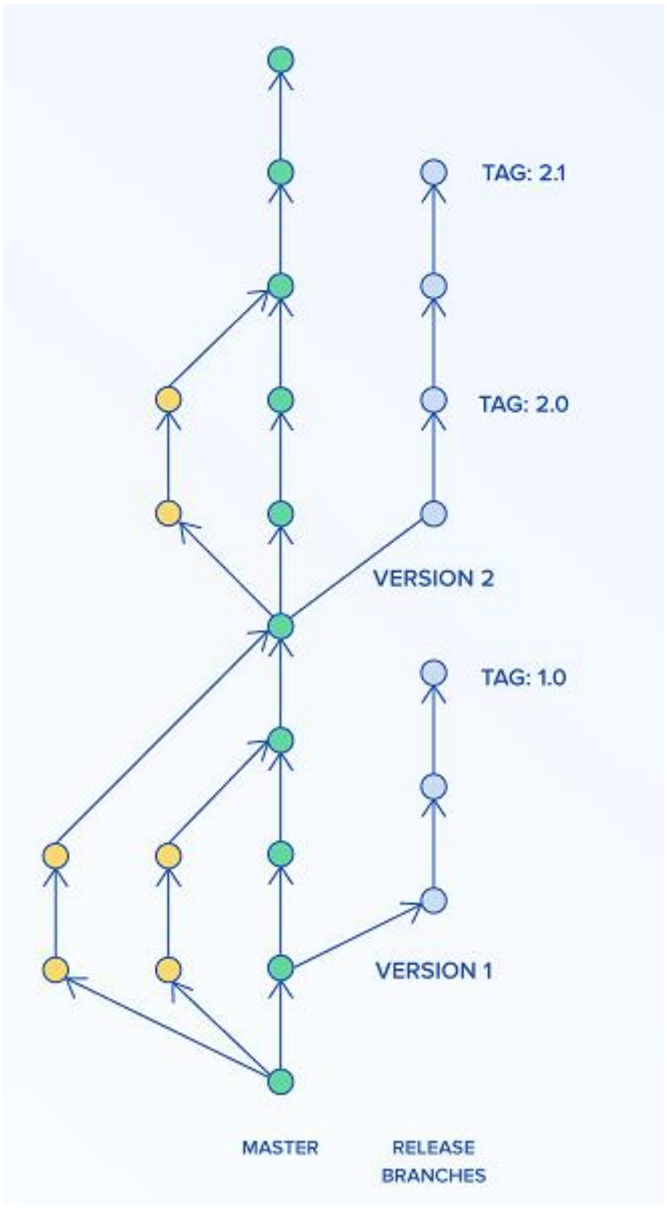


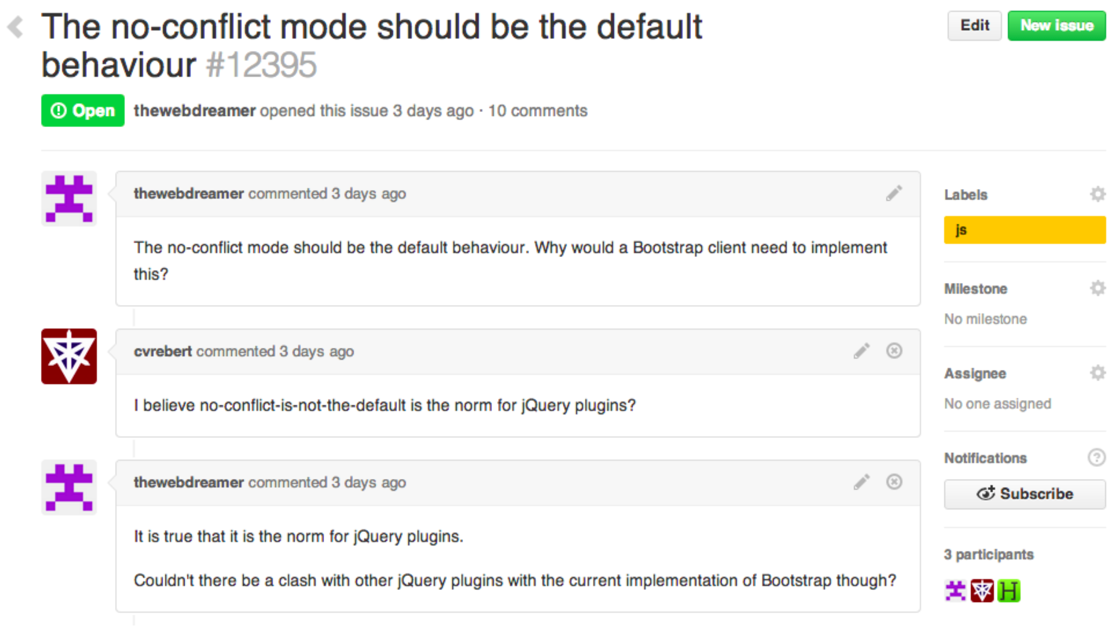
Figure 1: Trunk-based development workflow [11]

Based on the above workflow, all developers work on the master branch. Every developer can commit code that compiles and passes the tests in their local machine.

When necessary, short-lived feature branches can be created. For some features where the development can take some time, creating a short-lived branch is a great way to develop in your own branch without perhaps breaking others work in *master*. Once code on the feature branch compiles and passes all tests, it must be merged into *master*. This strategy prevents developers from creating merge conflicts that are difficult to resolve.

2.3 Issue reporting tool

There are a lot of tools for issue tracking. As DITAS will be using GitHub as code repository, and GitHub already provides an integrated Issue Reporting tools, it makes sense to adopt it for the project. GitHub's issue tracking is focused on collaboration which is very important for a large and geographically separated team.


The no-conflict mode should be the default behaviour #12395

Open thewebdreamer opened this issue 3 days ago · 10 comments

thewebdreamer commented 3 days ago

The no-conflict mode should be the default behaviour. Why would a Bootstrap client need to implement this?

cvrebert commented 3 days ago

I believe no-conflict-is-not-the-default is the norm for jQuery plugins?

thewebdreamer commented 3 days ago

It is true that it is the norm for jQuery plugins.

Couldn't there be a clash with other jQuery plugins with the current implementation of Bootstrap though?

Labels: js

Milestone: No milestone

Assignee: No one assigned

Notifications

3 participants

Figure 2: A typical issue on GitHub looks like this.

It is highly integrated with the GitHub features, offering a discussion board, referencing other issues and mentioning other users. Other valuable feature is the possibility to close issues writing a simply comment on the *commit* that fixes it.

2.4 Directory Hierarchy

Directory hierarchy is strongly related with the programming language and the best practices on it. This deliverable will no set closed rules for directory hierarchy of every component. The *Continuous Integration* section address the plugins to be used for Project Management (Maven or similar), and usually this tools set a common directory hierarchy for every project. Using this will lead to a default folder structure for every developer, despite, as said before, is usually tied to the programming language used and can vary from one to another.

2.5 Summary

This is a summary of the practices detailed in the above sections:

Concept	Decision	Details
Code repository	GitHub	https://github.com/DITAS-project
Repository structure	1 per component	
Repository naming	Basic rules set	*wp* prefixes name, *hyphen as separator and lowercase* letters.
Visibility of repos	Public	Due to the open-source nature of DITAS, the whole code library will be open-sourced since

		the beginning.
Workflow	Trunk based development	*master* branch on every repository.
Branching policy	Short-lived branches	Short-lived branches are allowed for specific features.
Issue reporting tool	GitHub's issue tracking tool	Highly integrated with GitHub's features.
Directory Hierarchy	N/D	

Table 2: Summary of decision for the development stage

3 Continuous Integration

3.1 Contextualization

Continuous Integration is a development practice where developers push code to a common repository several times a day and every change is verified by an automated tool, allowing teams to detect problems early, spending less time debugging and fixing bugs.

DITAS includes three technical work packages (WP), WP2-3-4. The deliverables published on M9 for each WP shows the high number of components to be developed in the next months by different partners, by different development teams. Every component interacts with many other. Having a structured method for discovering integration failures after every code change will make the development team more efficient, avoiding long development periods without being conscious of the impact of the same.

To achieve this, DITAS will build the CI process using Jenkins, an automation tool that will enable the execution of the whole test-suite, and will integrate third party tools to automate the building and deploying processes for every component. These tools will also support the assessment of different metrics described on Section 6 – Technical verification and validation approach of D1.1 [12] (M6).

3.2 Integration and testing

For automating the Build > Test > Deploy process, the Jenkins [7] open-source automation server will be used.. The Jenkins server is extremely extensible using plugins. We will make an extensive use of plugins in order to ease the creation of pipelines, to meet our verification metrics and to ensure the quality of the code. On a first approach, this is the list of plugins to be installed:

- **Blue Ocean [13]:** This plugin overrides the default Jenkins interface, simplifying things, and making it visually simple to check the status of every pipeline.
- **Custom Tools Plugin [14]:** In order to make it easy to use third party tools like test-runners, compilers, etc. this plugin will abstract the user of installing every tool on every Jenkins node.
- **Failsafe [15] and Surefire [16] plugins:** The combination of this two plugins permit splitting the execution of unit and integration tests.
- **Ansible [17] plugin:** A plugin for running Ansible Playbooks from pipelines.
- **Cucumber Reports [18] Plugin:** For running system and acceptance testing using Cucumber.
- **Git Changelog [19] Plugin:** This plugin will allow the automatic generation of changelog files.
- **Cobertura [20]:** Cobertura or similar plugins will be installed to measure the test code coverage.
- **Checkstyle [21]:** Checkstyle or similar plugins will be installed to check the code against coding standards.
- **FindBugs [22]:** FindBugs or similar plugins will be installed for static code analysis (depends on the coding language).

The following will describe a base process on how to integrate build tools into Jenkins, test calls, how to automate the execution of the tests, the results and the deployment tasks.

3.3 Environments

There will be three environments for deploying, testing and execution:

- Development
- Staging
- Production

Every software component must go over the three stages on its way to the final production environment. The development environment is the developer local workstation, an individual environment for every developer where he/she develops and runs local unit tests. After tests, code quality checks and other involved procedures are completed successfully, the component is deployed to the Staging environment where system level tests are performed. In this second environment, any other developer can check the integration between components apart from the individual functionality of each single component. The movement from Development environment to the Staging environment is done automatically using Ansible. Finally, when the component is considered stable enough to move to Production, a manual deployment is fired using the deployment tools as well. Automated validation can be carried out in the Staging environment while validation from end-users must be carried out in the Production environment.

3.4 Pipeline Overview

Jenkins allows the creation of pipelines [9]. Technically, a pipeline is a suite of plugins which supports implementing stages and steps. Every change to the software code, committed in the source control repository, goes through a process on its way to being deployed or released. The pipeline is defined in a file called `Jenkinsfile` [8]. The `Jenkinsfile` is composed by a set of steps, usually calls to third party tool's commands, like Maven, Ansible, Yarn, and other dependencies.

In DITAS, we will use the foundation of "pipeline-as-Code", where the pipeline is defined and then stored in the source repository, where it can be reviewed, maintained and versioned like any other code. The pipeline definition uses a domain specific language (DSL), in this case, based on Groovy. It is mandatory to manually codify the pipeline using the DSL due to reported problems when combining the Jenkins user interface (UI) for pipeline definition and the `Jenkinsfile` itself. While using the UI could seem to be easier, it is not possible to access all the features that the DSL offers.

The next image describes the high level approach for the Continuous Integration Pipeline to be deployed for every repository / component.

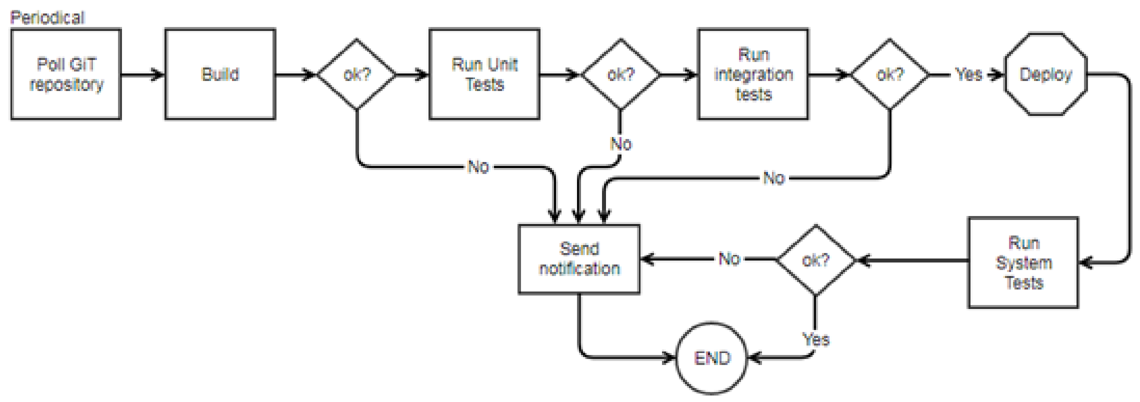


Figure 3: A high level approach of the pipeline

As every component will have its own repository, every component must have a `Jenkinsfile` that defines the above pipeline for it, configuring it according to the component programming language, test suite and deployment strategy. Two pipelines for the two components developed over the same programming language and using the same test runner, could share a high percent of code of the pipeline definition, easing the pipeline definition.

3.5 Dependencies

The `Jenkinsfile` invokes several tools for performing the needed tasks. For example, it could define a call to the maven goal test command to run the unit tests, which mean that the maven tool must be accessible in the node where the command is executed. The same may happen with several other tools. In order to simplify the dependencies for the `Jenkinsfile`, the plugin *Custom Tools Plugin* will be used. This plugin allows to globally defining the tools to be used in the pipeline, and takes care of the first-time installation when necessary. This will simplify also the deployment of new worker nodes for Jenkins when necessary, ensuring the new nodes will have the dependencies ready to go. The dependencies mentioned in this document will be installed by default. In the case a development team need a particular dependency to work, for example, some specific version of JDK, it can be added using this plugin.

3.6 Building, running and reporting tests

This section tries to define a common set of tools to be used by the whole members of the project in order to make the build and run software tests. These tasks, as stated previously, will be defined in the `Jenkinsfile` with several calls to third party tools. This is a list of recommended tools:

Tool	Comments
Maven [23]	Components developed using Java
Npm [24], Yarn [25]	Components developed using Javascript or nodeJS
Gulp [26]	For automation in web stack development
Ansible [27]	Server configuration and automation tool for deployment tasks

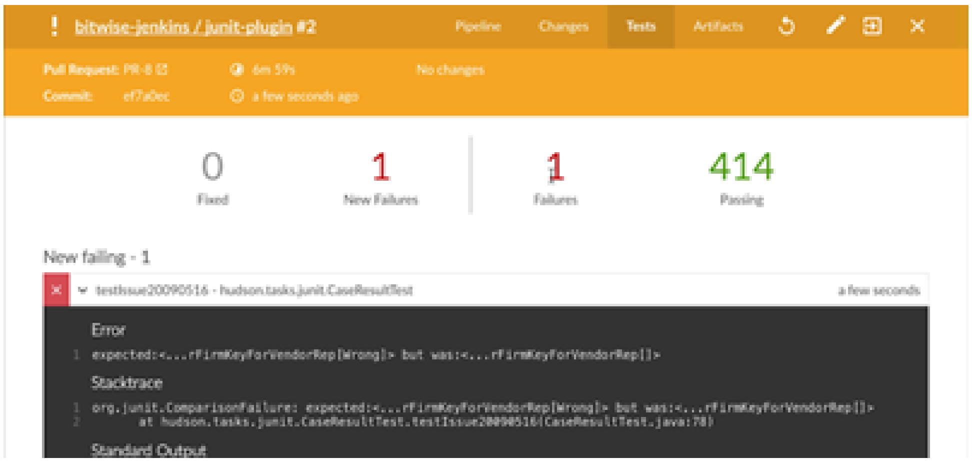
Table 3: List of recommended tools for building, running and reporting

Developers may use any other tool that could speed up developments in any case. The specific pipeline will have to be adapted to fit the chosen tool. With respect to the tests, splitting the test suite is a key action to get a fast feedback to know right away if the code breaks something critical on the project. In the specific case of Java components, the *Surefire* and *Failsafe* plugins will be used. Two different maven goals will be created in order to split unit and integration tests, and it can be executed running two *maven* goals:

- test (for running unit tests)
- integration-test (for running integration tests)

Even unit test can be divided using Junit categories, in order to run even faster or to make a logical division of tests. In any case, dividing and parallelizing the execution will bring an important reduction of the execution time.

Regarding test reporting, Jenkins can record and aggregate test results as long as the test runner can output test result files. Jenkins pipelines come bundled with the `junit` step, a specific step to manage test results in JUnit style. For this step to work, whatever the test runner the pipeline is using (maven, yarn, ...), it must be configured to output the results in Junit-style XML format. Then, Jenkins will detail the test execution results into the Test tab in the Blue Ocean interface.



```
expected:<...rFirmKeyForVendorRep[Wrong]> but was:<...rFirmKeyForVendorRep[]>
```

```
org.junit.ComparisonFailure: expected:<...rFirmKeyForVendorRep[Wrong]> but was:<...rFirmKeyForVendorRep[]>
        at hudson.tasks.junit.CaseResultTest.testIssue20090516(CaseResultTest.java:78)
```

Figure 4: Example of the test results dashboard

Junit will grab the test results and let Jenkins track them, calculate trends and report on them. A pipeline that has failing tests will be marked as "UNSTABLE". For more detailed information refer to the official documentation on https://jenkins.io/doc/pipeline/tour/tests-and-artifacts/.

3.7 Deployment

Ansible is selected as the main tool for deployment stuff. For web-stack development, in the case there is any development of this type; Gulp can be used as alternative if it is preferred by the development team in charge. So, this is the list of recommended tools:

Tool	Comments
Ansible [27]	Main server configuration and automation tool for deployment tasks
Gulp [26]	For automation in web stack development, JS or nodeJS

Table 4: list of recommended tools for deployment

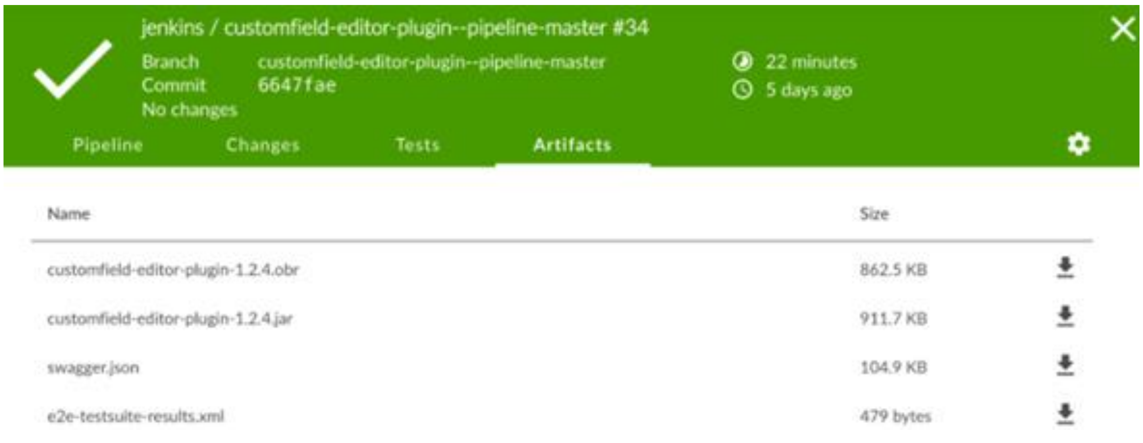
As stated previously, the Ansible Plugin will be installed, allowing direct calls to specific Ansible's steps in the `Jenkinsfile` using `ansiblePlaybook()` or `ansibleAdHoc()`. For a more detailed information refer to the official documentation on https://github.com/jenkinsci/ansible-plugin. In the case of Gulp, direct shell calls can be made from the pipeline using the `sh` step.

3.8 Notifications

Notifications will be performed using the post `stage` in combination with the built in step `mail` for failed and unstable states. It is up to the developer to set notifications for success builds too. The official documentation has some clear examples on how to use this feature. For more detailed information refer to it on https://jenkins.io/doc/pipeline/tour/post.

3.9 Artifact generation

By default, Jenkins will store successful builds into its own repository. For an artifact to be stored, the Pipeline developers must use the archive step inside the post section. The generated artifacts are stored and accessible from the *Artifacts* tab in the Blue Ocean interface. All the legacy builds are also accessible from the same place. See the official documentation for more details: https://jenkins.io/doc/pipeline/tour/tests-and-artifacts/.



jenkins / customfield-editor-plugin--pipeline-master #34

Branch customfield-editor-plugin--pipeline-master
Commit 6647fae
No changes

22 minutes
5 days ago

Name	Size
customfield-editor-plugin-1.2.4.obr	862.5 KB
customfield-editor-plugin-1.2.4.jar	911.7 KB
swagger.json	104.9 KB
e2e-testsuite-results.xml	479 bytes

Figure 5: Example of artifact list for a pipeline

3.10 Parallelizing tasks

Running a complete Jenkins pipeline can take a lot of time for some big projects. On the other side, there are some tasks of the pipeline that could run in parallel. The Jenkins architecture, based on node workers, allows running several tasks at the same time. When defining the pipeline, it is a good practice to define which tasks must run in parallel. Unit and integration tests are a clear example of tasks that could perfectly run in parallel.

In the next section a detailed approach of a complete pipeline is described, identifying several tasks that can run in parallel.

3.11 Detailed pipeline

The following image depicts an example of a complete pipeline, where tasks are processed in parallel and reporting results are generated.

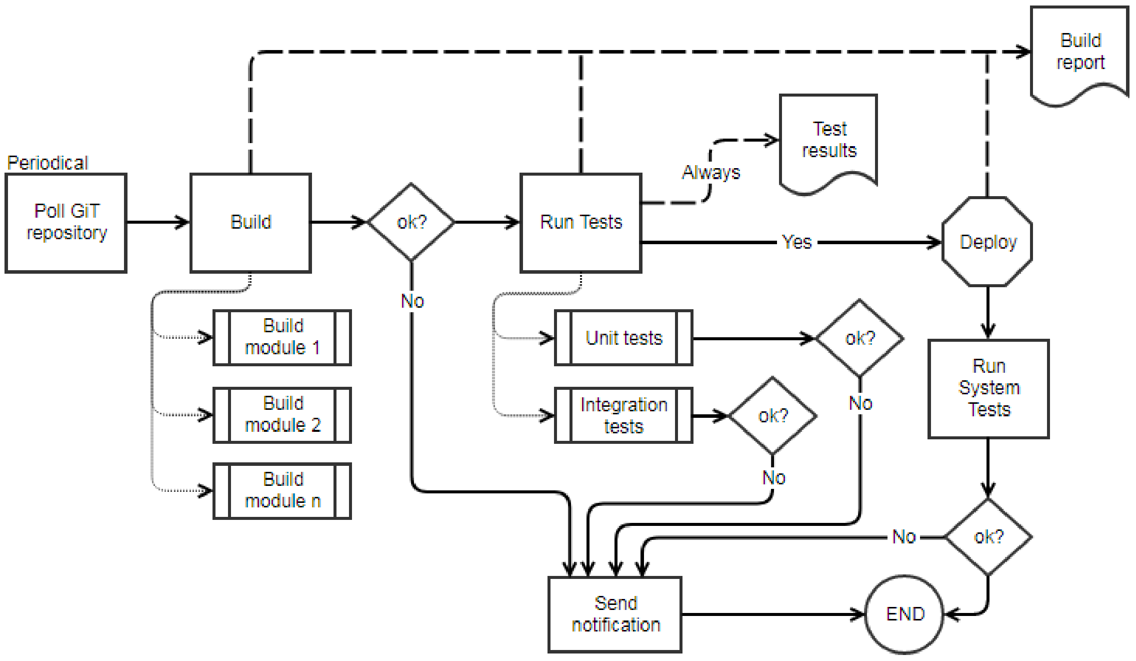


Figure 6: Detailed pipeline flow

The above image represents an ideal pipeline definition flow. Depending the component being developed, the final pipeline can vary from the defined above, but it can serve as a general base for the definition of pipelines.

The main activities, Polling GIT repository, build and running tests must be represented as stage-s within the `Jenkinsfile`. This allows to separate each from the others into the Blue Ocean interface, having a simple view for fast failure detecting on unsuccessful pipeline executions.

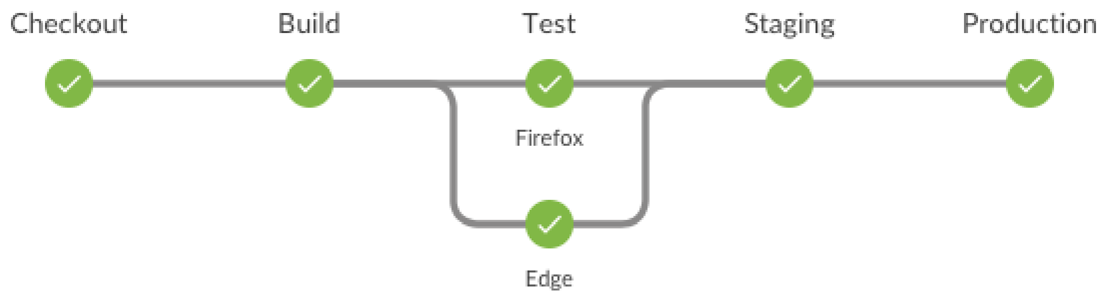


Figure 7: Example of successful pipeline execution in Blue Ocean

If the build process implies building different components, it could run in parallel as well. Same thing happens with the tests. In order to avoid large test executions and to have a fast feedback of the results, splitting tests and running them in parallel is a good practice.

3.12 Summary

The following is a summary of several topics covered before, with the aim of compiling all the best practices in a single and handy list:

Pipeline generation

- Poll the project repository frequently enough
- Separate the pipeline in several stages, to have a better view of the failure in Blue Ocean
- Codify the pipeline, don't use the Jenkins UI controls
- Parallelize when possible
- Better to parallelize stages instead of steps
- Depending on the Jenkins infrastructure, use the `node` command to target specific, powerful nodes for expensive tasks

Jenkinsfile

- Keep the `Jenksinfile` on the source repository
- Version the `Jenkinsfile`

Tests

- Split Unit and Integration tests
- Report test results in Junit-style format for integration into Blue Ocean
- Simulate external systems to reduce deployment complexity
- Send notification to be aware of failing tests
- Use one of the designed test runners when possible

Artifacts and deployment

- Use the `archive` step to save artifacts into the Jenkins own repository
- Use ansible for deployment task when possible
- Run the Ansible playbook from the Jenkins file using the specific commands
- Use `sh` step for running any third party tool with no specific step in the pipeline

4 Cloud Testbed

4.1 Overview

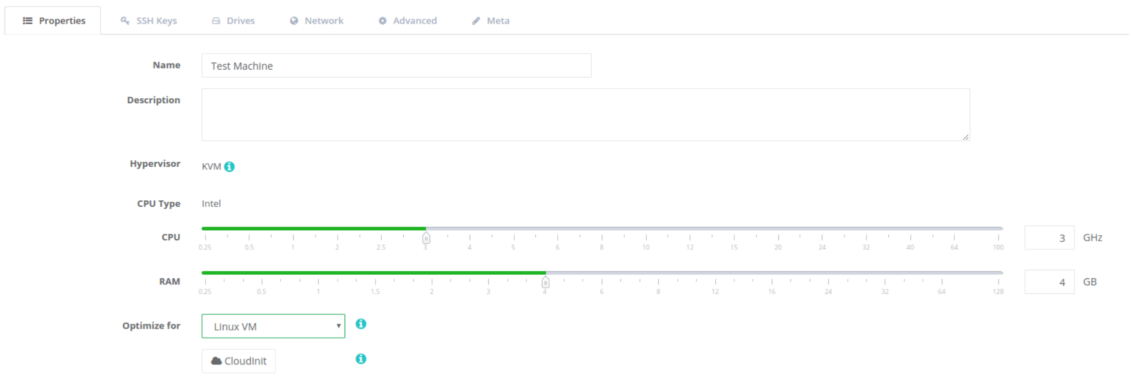
The testbed infrastructure is comprised of one production cloud provided by CloudSigma (public IaaS), and two fog/edge environments provided by IDEKO and OSR. CloudSigma ensures full availability of its cloud infrastructure for all beneficiaries, providing to the project a heavily discounted 3 years subscription for 100 Ghz CPU, 100GB RAM, 1500 GB SSD, 3000 GB HDD as indicated in the Grant Agreement. Any one of CloudSigma's European locations (e.g. Zurich, Frankfurt, Warsaw) can be used, but other non-EU locations could be used for testing latency.

We provide a more detailed description and the current specifications of the CloudSigma testbed, showing the total and available capacity of each resource including CPU cores, RAM and storage.

4.2 Definition of the Cloud Testbed

4.2.1 Hardware

CloudSigma is a pure Infrastructure-as-a-Service cloud provider that provides an environment with the same degree of flexibility and customisation as equivalent private in-house environments. All functionality is available via an API or the end-user WebApp. Virtual machines can be provisioned in a matter of seconds (15sec avg.) with a high degree of control. Users have the option of full API access with all account actions available, i.e. 100% API coverage, allowing complete automation and remote infrastructure monitoring, or the option of a feature-rich, yet intuitive web browser based GUI. The CloudSigma WebApp allows for resource management via any web browser. Project partners are able to select CPU and RAM to the nearest MHz and MB. VM provisioning can be achieved via a simplified wizard or custom server creation tool as shown below in figure 8 (CloudSigma WebApp - custom server creation).



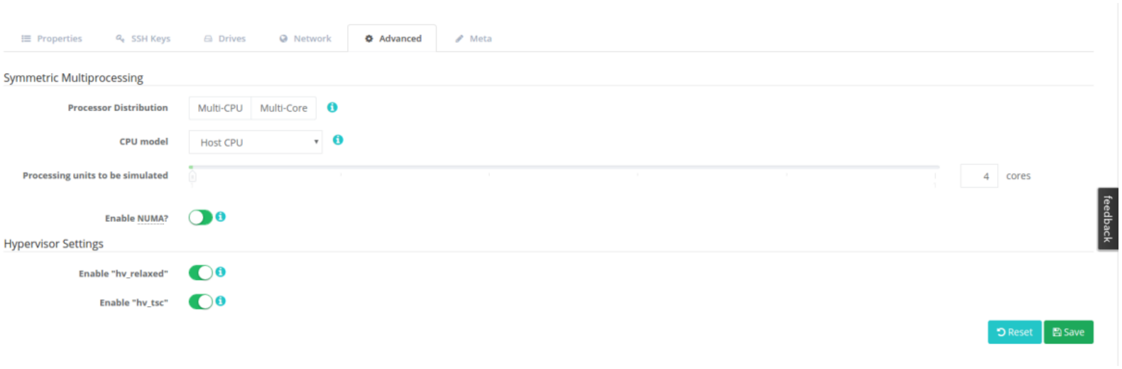
Name	Test Machine
Description	
Hypervisor	KVM
CPU Type	Intel
CPU	3 GHz
RAM	4 GB
Optimize for	Linux VM

Figure 8: CloudSigma WebApp - custom server configuration

Processing, storage, networking and other fundamental computing resources are not bundled, meaning CPU, RAM, storage and bandwidth can be combined independently. This level of flexibility is available to DITAS project partners, as long as the total resource limits outlined in the Grant Agreement are not exceeded. CloudSigma reserved the appropriate accounts and will monitor usage during the project and offer solutions before these resource limits are reached.

All cloud servers and drives are persistent and modelled on the same methodology as physical dedicated server equivalents (i.e. drives, NICs etc.) VLANs and IP addresses are also controlled using standard behaviour and support all types of traffic including multicast and broadcast traffic, which is critical for high availability infrastructure in failover.

CloudSigma's IaaS cloud platform exposes a number of optimisation tools allowing for greater performance levels. For performance sensitive workloads this is especially important.



Symmetric Multiprocessing

Processor Distribution: Multi-CPU | Multi-Core

CPU model: Host CPU

Processing units to be simulated: 4 cores

Enable NUMA?

Hypervisor Settings

Enable "hv_relaxed"

Enable "hv_tsc"

Figure 9: CloudSigma optimisation tools

Figure 8 above shows a screenshot from CloudSigma's provisioning portal advanced tab showing the options for optimisation. The following can be configured:

Processor distribution - This determines whether the symmetric multiprocessing units set for the server are distributed and exposed as CPU cores or CPU sockets. It is recommended to set this to Multi-Core with Windows (due to licensing requirements for Multi-CPUs) and to Multi-CPU for Linux distributions using NUMA (see below).

CPU model - The virtual CPU model (Host CPU or KVM64) can be selected for mitigating compatibility issues between the guest operating system and the underlying host's CPU. By default, all of the hypervisor's CPU capabilities are passed directly to the virtual machine.

Processing units to be simulated - The number for cores can be manually set.

NUMA - With Non-Uniform Memory Access, the memory access time depends on the memory location relative to a processor. NUMA can be enabled for servers with more than six processing units. If it is enabled for Linux distributions, it is recommended to use a Multi-CPU distribution.

hv_relaxed - This is a hypervisor setting, which enables relaxed timing for the CPU. hv_relaxed should be enabled for Windows, as it considerably increases performance. However, for Linux distributions, this should be disabled.

hv_tsc - This hypervisor setting enables the Time Stamp Counter to be passed through from the host to the server. hv_tsc should be enabled for Windows, as it considerably increases performance. It should be disabled for Linux distributions.

4.2.2 Software

Any x86 based operating system and software can be used with complete administration/root control including all variants of BSD, Linux and Windows. Raw ISO images can also be uploaded, and attached to CPU and RAM allowing full backwards compatibility from the platform. The end-user can choose from a wide selection of ready system images including a number of BSD, Linux and Windows based operating systems as well as being able to quickly and easily upload their own ready ISO image. End-users can customise these marketplace images using cloud initialisation frameworks such as cloudinit (see https://help.ubuntu.com/community/CloudInit), allowing them to take a standard installation and contextualise/customise it to their specific requirements on-the-fly first boot-up.

5 Case Studies

In the next two sections, the two case studies are detailed.

5.1 Industry 4.0 case study

The Industry 4.0 revolution has transformed the machines into data generation artefacts. Lots of gadgets, acquisition cards, data gathering systems, the Computerized Numerical Control (CNC) itself and ad-hoc applications can nowadays keep an eye on the machine and ongoing process extracting hundreds of machine indicators per second. The same applies also to shop floor level, where several machines are managed and monitored as a unique entity.

This lead to a huge amount of data generated at every moment for every plant and every machine. The storage of this data varies depending of the gathering platform and client requirements, ending up with a mix of heterogeneous data sources to manage when developing a Data Intensive Applications.

Sensors attached to acquisition cards can usually provide data in streaming basis or in a data lake mode. Historical data for flag-like indicators is stored in relational databases for short windows of time, or in non-relational for long term storage. For specific needs, data may be transformed and aggregated into a data warehouse to run specific queries. Furthermore, companies want to integrate the production workflow with the reality of the machines, so new data sources comes into play: managing production orders from the ERPs or Manufacturing Execution System (MES) or connecting with Computerized Maintenance Management Systems (CMMS) is becoming usual.

The data is very valuable. Apart from simple dashboards, the exploitation of the data leads us to the concept of Social Machines. This concept brings up a near future scenario where machines learn and adapt themselves from other similar machines. They learn not only from their own behavior but communicate and interchange data with other machines working on any geographical location around the world. The interchanged data and computation result serves for several objectives like:

- **Condition monitoring:** gain insights from the lifetime experienced in other machines to assess the remaining lifetime of critical components.
- **Production:** suggest some parameters in order to increase the productivity.
- **Working parameters:** custom dynamic machine configuration to increase the Mean Time To Failure (MTTF) and improve machine availability.
- **Smart maintenance plan:** Acquire new maintenance knowledge and propose a maintenance strategy based on this information.

While this is a very powerful scenario, in the harsh shop floor environments some problem arises.

5.1.1 Common problems

The following exposes the common problems when dealing with Industry 4.0 data and applications.

5.1.1.1 Data Volume

While persisting the data is critical, data gathering systems are getting more powerful every day, being able to generate data at very high sample rates for some sensor types. This leads to vast amounts of raw data that usually need to be pre-processed before for different reasons:

- Storing all raw data is unpractical.
- Network is a bottleneck.
- Simple useful computations at the edge are useful to close the loop.

The data volume problem is usually a combination of the volume with some other network restriction or bottleneck.

5.1.1.2 Network Problems

In the shop floors, dealing with connectivity, latency or making the necessary computations timely, arise as critical problems for highly distributed systems. Connectivity on remote shop floors is usually poor, dealing with limited bandwidth and jitter. While persisting data is critical, it could even be impossible to persist in the cloud all the data generated by a machine, due to network factors and the volume of the generated data.

It is a common practice to apply data reduction tasks at the edge. For example, it is usual to apply Fast Fourier Transform to data coming from accelerometers to reduce the amount of data to be transferred.

5.1.1.3 Data quality

The quality of the data is another issue to deal with. Connectivity between the machine and data gathering elements are not always reliable. Also data gathering systems can hang up sometimes, stopping to read data, ending up with partial datasets for certain periods of time. Identifying blond time segments or curating datasets is a hard work.

5.1.1.4 Data heterogeneity

Another big issue comes from the heterogeneity and non-interoperability of the data origins: different machines, even in the same shop floor, may have different CNC models that lack interoperability protocols, which forces to implement a complex interoperability layer to enable communications among them.

5.1.1.5 Data synchronicity

Data synchronicity problem arises when there is the need to mix data from two or more data sources and for the underlying application, it is critical to synchronize data samples in time. The latency introduced by the data gathering systems for some data types, and the accuracy of other gathering systems, lead to a complex synchronicity mechanism. A simple example of this problem occurs when dealing with accelerometers, for measuring vibrations, and having to mix with other dataset coming from the CNC. The data gathering mechanism for accelerometers consist in an extremely precise apparatus, while the system to read data from CNC introduces a small latency. Accelerometers generate data at high frequency, at a rate of 20.000 samples per second usually, while CNC variables are read at a rate of 1-2 samples per second. The different sample rates along with the timestamp makes difficult to sync data streams.

5.1.1.6 *Unbalanced edge computing power*

IDEKO uses its own data gathering smart-box (introduced in details later on), an industry PC for gathering machine data that can connect to the most common CNC models and other data origins and sensors.

There are different models of boxes, with different price-ranges and computational power. Boxes are attached to machines with 1 to 1 relation. This leads to a scenario where some machines have more computational power than others, just because the box attached is more or less powerful. So, all computations at edge level are strongly dependent on the characteristics of the box, and even not possible, having to come up with a different technical solution, where the usual path is to move data to the cloud in order to have a powerful server where run computations and get the results back to the edge.

5.1.2 Data gathering systems

IDEKO works closely with Savvy, a technological startup focused on machine-monitoring and data analytics. In conjunction with them, IDEKO has developed the **Smart Box**, an industry-ready box for gathering machine data. The Smart Box is actually a small as possible industrial PC with a case designed to support the rude environment of the shop floors.

The box is already built in with all the connectivity needs to get data out of the machine and send it to a private cloud. The box can read data from several CNC models and standard manufacturing protocols:

- Siemens, Heidenhain, Fanuc and Fagor CNCs / PLCs
- OPC-UA
- OPC-DA
- MTConnect
- Modbus
- Profinet

This allows the box to read low frequency variables or to enable publisher / subscriber mechanisms for flag like variables at the same time it connects to a data acquisition card from to read high frequency data types.

5.1.2.1 *Technical features*

IDEKO usually deploys boxes with these characteristics:

- Debian Jessie host
- 4GB RAM, 60GB SSD, Celeron Quadcore 1.6GHz boostable to 2.09GHz.

There are other configurations available:

- i5 – i7, 16GB RAM, up to 1TB SSD

5.1.2.2 *Containers in the box*

There is a dedicated space for deploying Docker containers in the box. Deployed containers are limited to use up to128M RAM and 25% of CPU usage. The deployed containers have direct access to the machine data using a polling REST API.

5.1.3 Data storage systems

The following is a list of common data storage types used in applications when dealing with machine data. The aim of this list is to get the idea of the large variety of sources that can take part in Industry 4.0 applications. The case study will not make use of all of them, and probably, depending on the final framework, some other sources like Minio Multi-Cloud Object Store [28], introduced on D2.1- Section 6 [29] could be added to the list.

5.1.3.1 MySQL / PostgreSQL / MariaDB

Commonly used for in-machine edge storage system. When storing machine status every second or half of a second for a hundred of variables, storing all this data in a relational database can be challenging. On the other hand they provide an easy way to query data and plenty integration with third party systems. That's why this kind of databases can be a nice solution to store data for a predefined short window of time, to not exceed the limited resources of the edge.

5.1.3.2 Influx DB

Lightweight enough to serve also for in-machine storage, Influx DB and other time-series databases are suitable for storing series of data that do not change too much frequently. The storage footprint is smaller than in the relational databases while these database types provide also structured queries for data retrieval.

5.1.3.3 MongoDB

We count with a non-relational database to store large amounts of semi-structured data. Deployed in the cloud along with an API for data access accessible in two ways:

- REST API: a common REST API for querying data in the database.
- Streaming API: a continuous flow of data that is sent to the client as it reaches the database.

5.1.3.4 Data Lake

The data lake storage system is usually used for storing large amounts of raw data. Data lakes are usually developed over cloud providers to benefit of storage capacity low rates. Data stored in data lakes is usually useful for historical analysis, research projects or even for semi real time applications that need historical data for some tasks.

5.1.3.5 Data Warehouse

This last storage system is less usual but sometimes useful. The data can be sent over to a data warehouse system, usually in the cloud, in order to enable fast reporting and data analysis.

5.1.4 Data and computation movement benefits

Having the flexibility to execute the computation load at the edge or in the cloud without having to manage the data and/or computation movement between the both, is a great benefit for application developers, moreover, when the computation power of the edge nodes varies depending on the selected box.

This is a scenario where computation adapts it execution to the context features like Box performance or network quality, and plays between a far but reliable and fast environment like the cloud, to a more unpredictable but closer, and thus low-latency environment like the edge.

To give an example, let's say that a machine is computing the remaining life-time of some long-life critical component (condition monitoring) and the aim of the computation is just inform the operator about the maintenance policies, so the computation is performed once a week. Then, the machine manufacturer identifies a critical design failure on the same component and wants to monitor that component closely. The manufacturer asks the application to perform the remaining lifetime computation in a continuous way in order to have early failure detection. In this scenario, initial application requirements can be satisfied computing at the edge, but it is possible that to meet the second scenario's requirements, computation must be performed at the cloud and that computation movement will be transparent for the developer thanks to the DITAS framework. It will deal with computation and data-movements if necessary, performing data-reduction tasks as well.

5.1.5 Scenario description

The use case will develop an **advanced technical service application** that improves the way the machine incidences are faced when they occur. Currently, when an incident occurs, the customer gets in touch with the technical service of the machine manufacturer. The machine manufacturer tries to identify the failure based on the following aspects:

- The description the customer makes of the incidence
- The values of key indicators he gets from the machine:
    - Remotely connecting to the machine using Team Viewer like tools
    - Asking the customer to look at them
- Previous experience on similar failures

Once the origin is detected, if the fix cannot be applied remotely, an engineer must travel to the customer's shop floor, he/she then inspects the machine, confirms the fault and tries to repair it. If the knowledge of the person is not suitable to make the repair, usually because the initial data for failure evaluation was not enough to select the proper person, another person must travel. This strategy derives in delays both in the incidence origin detection and the repairing of the same.

On the other hand, at the moment, every machine sold by IDEKO's industrial partner DANOBATGROUP, is sold with the Smart Box attached. The box automatically enables data cloud storage and edge computing. Being so, this is a proper scenario to develop high added-value data-based applications.

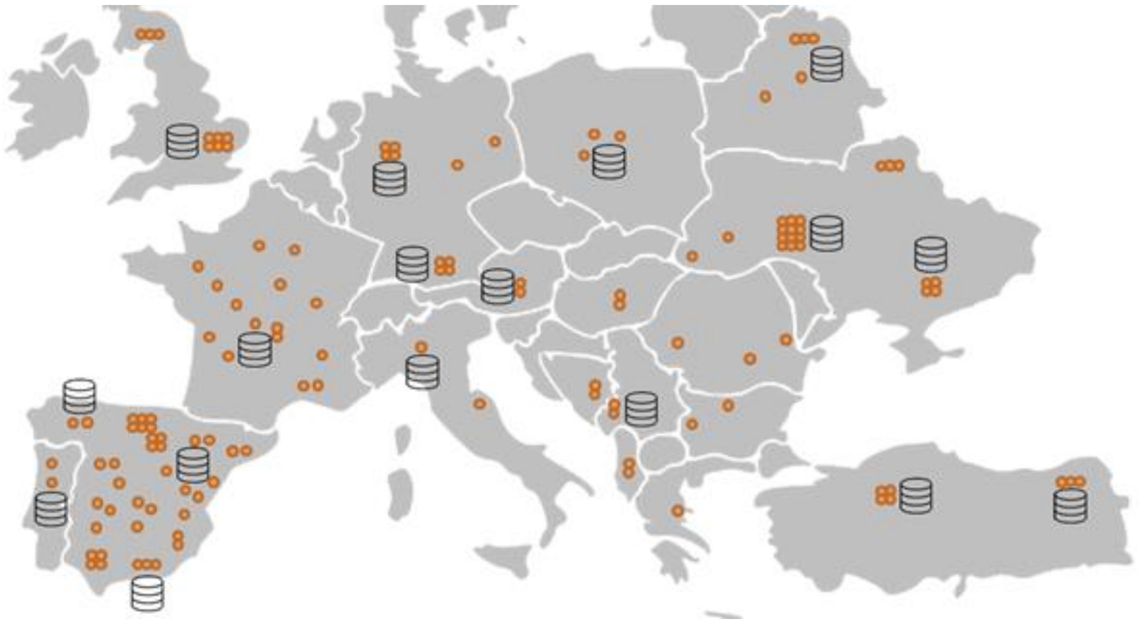


Figure 10: Scenario with machines with data gathering systems

Moreover, some critical components like spindles and axes are present in every machine in the market, and there are just a bunch of different models of both. So, it seems to be a promising scenario to leverage the machines to learn from experience of others.

In summary, the present-future scenario has the following characteristics:

- Lots of machines with Smart Boxes attached
- Indicators (data) currently being compiled
- Local storage and computation enabled in every box
- Quite similar critical components in every machine

This is an appropriate scenario for developing high added-value applications. The proposed technical service application will help facing technical service common drawbacks. The app will make possible to the technical service, to have a detailed view of the state of the machine when it is needed. The main objectives to achieve are the following:

- Reduce diagnostics time
- Decrease response time
- Decrease travel costs
- Prevent failures of other machines

The next section defines more precisely the expected work to be done regarding the application.

5.1.6 Application

The use case have the aim to combine in a single demonstrator, some of the possibilities that the concept of a Social Machine along with the increasing edge computing power makes possible. It will be developed around the idea of improving the machine diagnostics with the technical service department in mind. The technical service department is a key department where lot of key concepts of Smart Machines comes into play:

- Predictive maintenance

- Failure detection
- MTTF and MTTR improvements
- Historical machine behavior analytics
- Knowledge over the process
- etc.

Thus, this is an open, wide scenario where develop value-added applications. The demonstrator will serve for:

- Enabling an easy way to determine the status of the critical components of the machine.
- Enabling the real time view of key machine indicators.
- Allow the technical operator to know the machine condition at every moment.
- Predict the remaining lifetime of critical components.
- Give the customer customized advice based on evidences.

And it will be done by:

- Mixing historical data in the cloud with streaming data generated at the edge.
- Running digital signal processing algorithms and anomaly detection mechanism at the edge.
- Combining real time analytics at the edge with historical patterns.
- Learning from data and computations made in similar machines in remote locations.

Running computations at the edge to close the loop (to send commands to the machine) are critical if the commands affect the manufacturing in real-time. Despite feasible, this is not the common scenario. Usually, computations are maintenance-oriented or components lifetime-oriented and can be delayed on time. Regardless of that, sending vast amounts of data to the cloud can be impractical or even not possible, so, running at the edge is always a good practice.

Apart from the problems DITAS project is aimed to deal with, there are a lot of complex problems behind the application itself. The work behind the Machine Learning algorithms to predict the remaining lifetime of machine components is hard and requires time. Because the aim of the project is not to deal with Machine Learning algorithms but with computation and data movement to make it possible to run high performant jobs, for demonstrator purposes the application will focus on demonstrating the power of the developed framework instead of the application itself.

5.2 E-Health case study

Hospitals and clinics have collected, are collecting, and will collect significant amount of data with different types and formats. For instance, every year in Ospedale S. Raffaele more than 15 TB of clinical images (e.g. MRI, CT, etc.) are produced, managed and archived. In addition to them, personal data, clinical data, notes from clinicians are also collected.

Hospitals have the **need** and the **obligation** to **securely** store these data for a long period even if they are accessed mainly when the patient is hospitalized and less frequently when the patient comes back to the hospital for **emergency events** or **follow-up visits**. For instance it is of fundamental importance the possibility to retrieve the entire **patient's clinical history** in case of an emergency event.

Moreover, the resulting information heritage owned by hospitals, represent an **invaluable source of information** for other stakeholders working outside the hospital (e.g. researchers). Particularly, externally to the hospital, other organizations may be interested in these data, in order to create benchmarks, validate research studies, or to provide scientific statistical evidences. This opens to Ospedale S. Raffaele (and to other hospitals in the same situation) the possibility to provide these external actors with this invaluable source of information.

According to this scenario, **cloud based infrastructures** represent the most suitable solution to achieve the above mentioned goals, due to their potentially unlimited storage capacity and the ease of reachability. Conversely, **privacy and security issues** represent the main hurdle to the feasibility of the adoption of cloud based solutions as data belonging to the hospital are **sensitive** and specific regulations, such as **European General Data Protection Regulation 2016/679**, have to be strictly respected.

5.2.1 Scenario description

The e-health use case scenario is described in terms of edge / cloud infrastructure and in terms of the specific actors involved; the scenario is subdivided in three incremental phases.

5.2.1.1 Scenario description phase 1

The first actor considered is the medical doctor, that has to use patient's personal data in order to perform health care activities.

Workstations and mobile devices enable the access to patients' data for medical doctors. Patients' data are stored in a "**Hospital trusted Edge Infrastructure**" (current OSR core databases) until the patient is hospitalized and for a specific period after hospitalization (for instance 30 days).

The idea of the DITAS e-health use case is to exploit an **internal Trusted Cloud** where data could be sent after being **pseudo-anonymized** and **encrypted** in order to act as a long term repository. In general, data are supposed to be moved to the internal Trusted Cloud after one month of patient dismissal so that OSR core databases can be lightened. Obviously, medical information can be retrieved whenever is necessary by decrypting them with a proper key. The **internal Trusted Cloud** can be on premises of San Raffaele hospital.

The following figure represents the e-health use case description above mentioned.

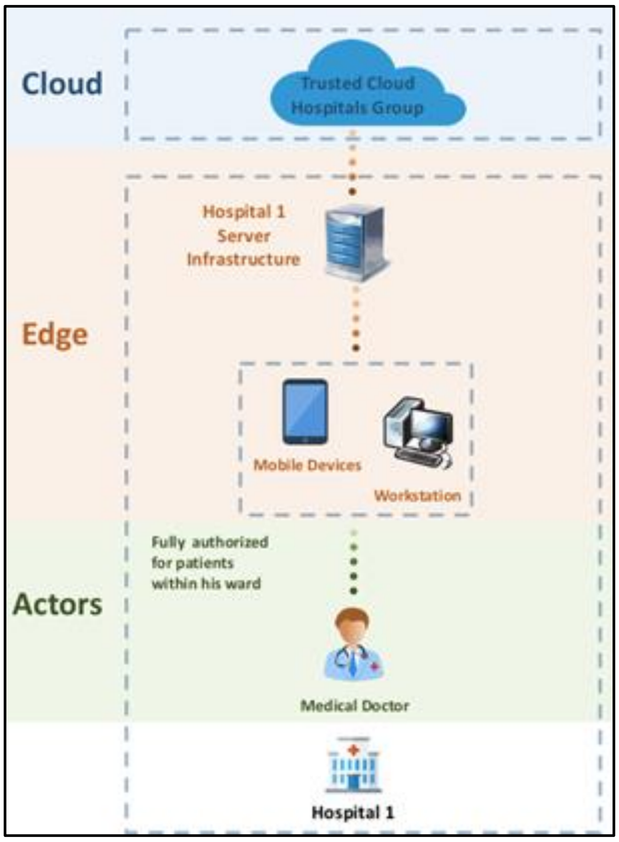

Cloud
Trusted Cloud Hospitals Group
Edge
Hospital 1 Server Infrastructure
Mobile Devices
Workstation
Actors
Fully authorized for patients within his ward
Medical Doctor
Hospital 1

Figure 11: e-Health use case scenario "phase 1"

5.2.1.2 Scenario description phase 2

The scenario "phase 1" can be expanded considering another hospital (Hospital 2) belonging to a specific group of hospitals including also San Raffaele hospital (i.e. same company; San Raffaele hospital is Hospital 1). In this situation the internal Trusted Cloud will host not only data coming from San Raffaele hospital but also other medical information coming from the second hospital; this cloud can be on premises of San Raffaele hospital acting as **"data center"** for the other hospital.

Scenario phase 2 also introduces a new actor, Medical Direction Administrative Staff, mainly interested in data analytics for statistical analysis and for key clinical indicators (meaning aggregated data). Just in case needed, with a specific request (traced and stored) Medical Direction Administrative Staff can see specific single patient medical records.

The following figure represents the e-health use case with two hospitals.

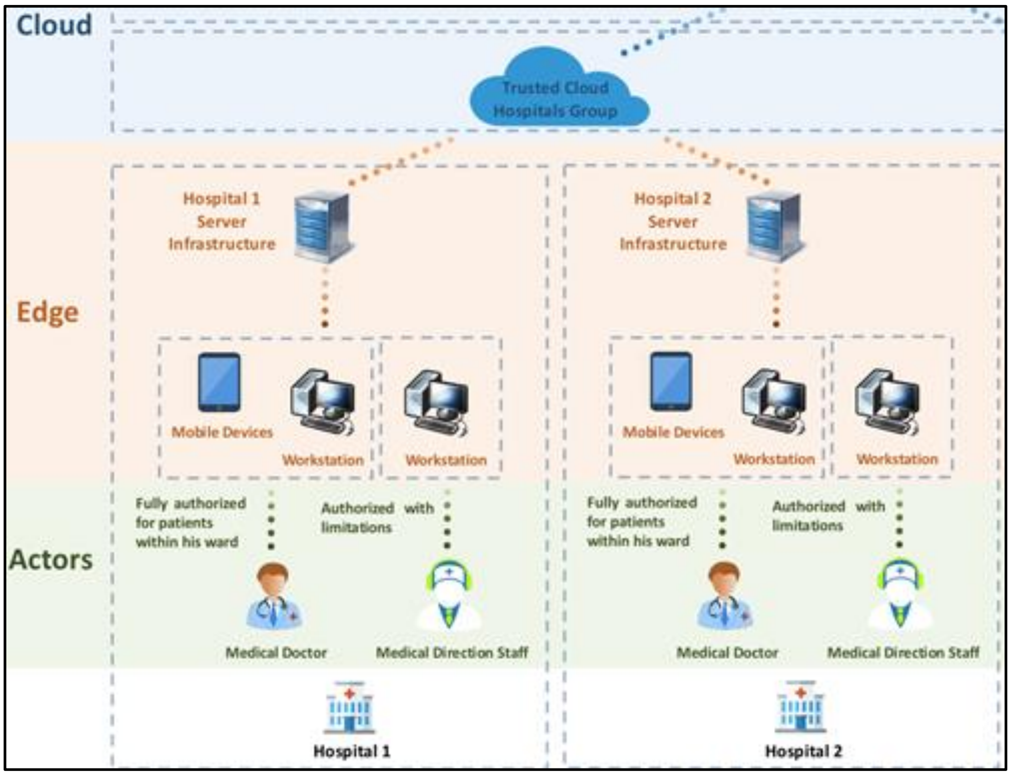

Cloud
Trusted Cloud Hospitals Group
Edge
Hospital 1 Server Infrastructure
Hospital 2 Server Infrastructure
Mobile Devices
Workstation
Workstation
Mobile Devices
Workstation
Workstation
Actors
Fully authorized for patients within his ward
Authorized with limitations
Fully authorized for patients within his ward
Authorized with limitations
Medical Doctor
Medical Direction Staff
Medical Doctor
Medical Direction Staff
Hospital 1
Hospital 2

Figure 12: e-health use case scenario "phase 2"

Coherently with Scenario phase 1, patients' data are stored in "Hospital trusted Edge Infrastructure" of Hospital 1 or Hospital 2 depending if the patient is hospitalized respectively in Hospital 1 or Hospital 2. Then data are supposed to be moved to the internal Trusted Cloud after one month of patient dismissal so that the Hospital Edge Infrastructures can be lightened.

The idea of the e-health case study is that the DITAS platform manages the **data movement and computation** between cloud and edge in order to make available data and information to the involved actors.

For instance we can mention the possibility for a Medical Doctor to have at disposal the clinical history of a patient currently hospitalized in Hospital 1 (data stored in Hospital 1 Edge Infrastructure), that had another hospitalization in Hospital 2 within previous 30 days (data stored in Hospital 2 Edge Infrastructure) and had another hospitalization in Hospital 1 some years ago (data stored in the internal Trusted Cloud).

Another example is about the possibility for the Medical Direction Administrative Staff of Hospital 1 to have dedicated data analytics applications in order to perform statistical analysis regarding both current hospitalizations and past hospitalizations in the previous years (so using data in the Edge Infrastructures and in the internal Trusted Cloud).

5.2.1.3 Scenario description phase 3

Finally, a part of all the medical information coming from different hospitals are sent to an **Untrusted Cloud** that can exclusively store **encrypted** and **anonymized, or in some cases pseudo-anonymized**, patients' data, to be used outside hospitals. The data on the **Untrusted Cloud** can be used, for instance, by Researchers that can access information to the Untrusted Cloud through a Researcher Trusted Cloud, where data can be decrypted without the possibility to go back to patient's identity.

The actor "Researcher" can access **only authorized, in terms of consent given by the patient, data**. The Researcher can be employee of Hospital 1, or Hospital 2 or can be external to Hospital 1 and 2.

The following figure represents the e-health use case with two hospitals and with data used by researchers outside hospitals.

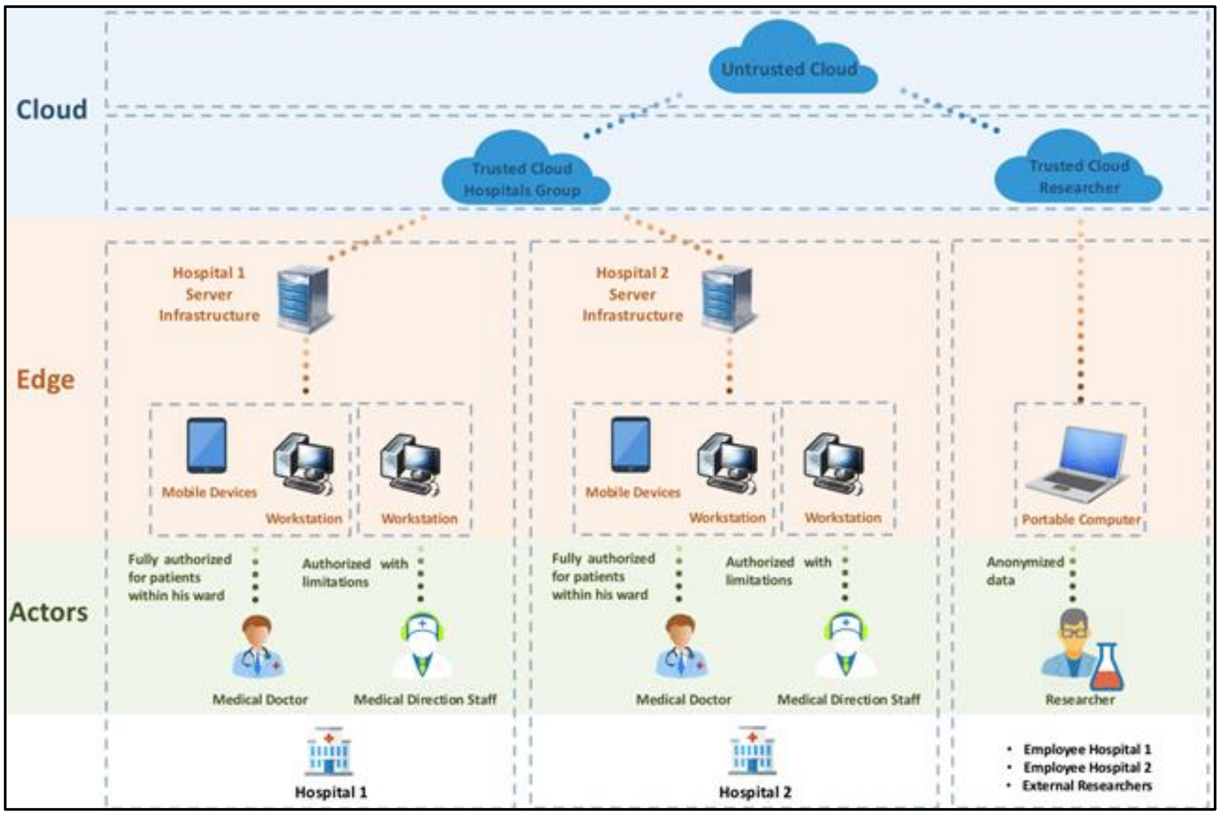


Figure 13: e-health use case scenario "phase 3"

5.2.2 Regulatory requirements

As already mentioned, one of the most important aspects regarding the described e-health scenario is about the privacy and security issues. Particularly the new **General Data Protection Regulation (EU) 2016/679**, also known as **"GDPR"**, has to be carefully considered.[1]

Moreover most of the data collected and processed in the e-health scenario are sensitive. To this respect the GDPR Article 9(1) expressly prohibit - as a general rule - processing of "special categories of data" (biometric, genetic, and **data concerning health**). However, considering the importance of using information about a patient in order to provide appropriate medical treatment, there are exemptions to the general prohibition of processing medical data. Article 9(2)(a-j) sets out the circumstances in which the processing of sensitive personal data, which is otherwise prohibited, may take place. Among them: explicit **consent, vital interest, public interest exception and research exception** (relevant articles: Articles 9, Recitals 51-56).

- ***Use of data for research purposes***

[1] The Regulation repeals the Directive 95/46/EC (General Data Protection Regulation) and shall apply from 25 May 2018 according to Article 99(2) of the Regulation.

Regarding the usage of data for research purposes, a distinction must be made between research as the primary purpose of the collection of data and research as a further processing. Starting from the first option, **Article 9(2)(j)** allows researcher to process personal sensitive data **without consent** where the processing is necessary for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes for the benefit of natural persons and society as a whole, and based on Union or Member which shall be "proportionate to the aim pursued, respect the essence of the right to data protection and provide for suitable and specific measures to safeguard the fundamental rights and the interests of the data subject." Thus, as clarified in Recital 52, research serves as a basis for processing sensitive data only "when provided by Union or Member State law and subject. In addition, the data controller shall respect the new Article 89(1) of the GDPR requiring both sufficient and adequate technical and organizational measures ensuring data protection and, in particular, in this context, the respect of data minimization.

It is important to note here that European Member States are entitled under the article 9(4), to maintain or impose further conditions (including limitations) in respect of genetic, biometric or health data. As such, existing differences in approach on these topics will likely be maintained, and further divergence will be permitted. Entities that process these categories of data should continue to keep the development of relevant national law under review and consider the need for further lobbying work in this area.

However, thanks to the digital revolution and the availability of large quantities of data - routinely collected and stored in gigantic health facilities datasets - it is becoming increasingly more common to use pre-existing personal health data in scientific research. Important sources such as electronic health records (EHRs) and datasets used in previous research offer large quantities of data without the need to engage in the primary collection of data. Whilst the possibilities in terms of innovative research are very expanding, re-use of health data is often perceived of as concerning from a privacy perspective. This is often because further processing health data gathered for treatment purposes may often not be based on the explicit and informed consent of the data subject. It is indeed very difficult for the scientists to list all research purposes in consent form at time data collected.

GDPR, on Recital 33, acknowledges that the purposes of scientific research cannot always be specified at the time of the initial data collection. This is why, at the same recital, it recommends that at the time of collecting data for treatment purposes, patients are permitted to express their consent to some fields of scientific research; this is done in line with recognized ethical standards for scientific research and compliance framework safeguards have been effectively implemented. With this broad consent personal data can be repurposed without getting further consent once individuals have given their consent to certain areas. It is, therefore, foreseeable that clinical centres will take steps - where they have not already done so - to implement templates of data processing consent, including in them also the general purpose of clinical-scientific research, albeit with reference to certain fields of research.

In other cases of reuses, where the processing for another purpose is not based on the data subject's unambiguous consent or on an EU or Member State law, the controller shall perform a purpose compatibility test. The proposition goes even further, stating that further processing for scientific research should in any case be considered "compatible lawful processing" (Article 5(b) - Recital 50).

This presumption of compatibility with the initial purposes of the processing advanced at the time of the first collection is notably related to the specific exemption to the principle of storage minimization, where the further processing (e.g. storage) is for research or archiving purposes in the public interest. This is a good news for health registries, cohorts and research biobanking maintaining personal sensitive data available for future scientific or statistical reuses. However, this presumed compatibility is not fully automatic and must answer to several requirements such as the respect of data minimization principle. Indeed, according to Article 89(1) and Recital 156, this further processing 'is to be carried out when the controller has assessed the feasibility to fulfil those purposes by processing data which do not permit or no longer permit the identification of data subjects (e.g. pseudonymization of the data), and provided that appropriate safeguards exist [e.g. secured and separated storage of the identifiers (codes) and respect of the relevant ethical standards in the field].

The GDPR applies only to information concerning an identified or identifiable natural person, meaning that it does not apply to Anonymous data, but it applies to personal data which have undergone pseudo-anonymization because, as clarified in recital 26, it should be considered to be information on an identifiable natural person. This aspect is very important because, since anonymization may reduce or even destroy the potential value of data for research purposes, in order to facilitate the use of data in the contest of research projects, the GDPR, certain exception, apply in case of treatment for research purposes with pseudo-anonymized data.

5.2.3 Business process analysis

In order to focalize the attention on a concrete **real world case study**, a **specific healthcare business process** has been selected and analyzed.

The work related with the e-health use case starts with the business process analysis in order to have a citizen / patient centered point of view; indeed the business process analysis is developed following the citizen / patient flow of activities during the healthcare process.

The idea is to consider the **data involved in the specific business process**, meaning the data used and generated during the healthcare activities.

The selected business process is about a patient that incurs a **stroke**. The motivations for this choice are the followings:

- stroke is a relevant problem in terms of population public health (14.000 cases a year in Lombardy Region, where OSR is operating)
- stroke is a pathology strictly monitored by the healthcare authorities
- stroke involves almost all the healthcare phases within and outside hospital (emergency care, acute hospitalization, rehabilitative hospitalization, follow up activities)
- OSR is recognized as an excellence in this field (700 cases a year)
- stroke business process involves different types of healthcare data (diagnostic images, blood tests, unstructured doc, etc.)

The business process is studied involving the related Healthcare professionals, Medical Direction representatives and IT department professionals; then the business process can be described in details using BPMN diagrams.

5.2.3.1 Storyline

The storyline of the business process begins considering a **citizen that incurs a stroke and the ambulance take her to the closest hospital**.

During **first aid in Hospital Emergency Department**, the emergency doctor discovers that the reason of the stroke can be related with a past patient's health problem that is a blood clot dislodged from a cardiac mechanical valve. Therefore, doctor decides to check patient's Electronic Health Record (EHR) to know in detail her medical history. As a result, the emergency doctor wants to view patient's previous medical images (before and after the surgery); this is necessary to compare the current situation with the one at the last checks. The emergency doctor has fully access to patient's EHR because she is treated in his yard. This situation is just exemplificative of a generic situation where a doctor needs to retrieve clinical data of a past health problem in previous hospitalization.

When the health care operations at the Hospital Emergency Department are completed (generally some hours), the patient is reassigned to the **Hospital Acute Operative Unit** where is treated with a strong anticoagulant medication able to dissolve the blood clot. During the therapy the woman needs a close monitoring, comparing the exams with the previous one to find evidence of the improvement. Particularly, by comparing biomedical images it is possible to get a percentage of the thrombus reduction and the correct valve work. Moreover, it is necessary to check if the blood parameters are retracting. During this period, that can last some days, the Acute Operative Unit Doctor has the patient in charge and can access her data.

After the acute phase is solved, the patient is discharged from Hospital Acute Operative Unit and reassigned to the **Hospital Rehabilitation Operative Unit** where she is supposed to stay recovered for more or less 30 days. During this period, the Rehabilitation Operative Unit Doctor in charge can access patient's data.

Follow up: after Hospital Rehabilitation Operative Unit discharge, the patient is supposed to have, in the subsequent years, **periodic outpatient visits** that are generally performed in another hospital (H2) different from the one where occurred the emergency and rehabilitative hospitalization (H1). During an outpatient visit in H2, the medical doctor wants to have at disposal a medical image from H1 acquired during the previous hospitalization and another image acquired in H2 in a previous outpatient instrumental diagnostic exam. H2 and H1 are supposed to be part of a trusted group of hospitals (e.g. same company).

The use case storyline above described is therefore based on the following schematic flow:

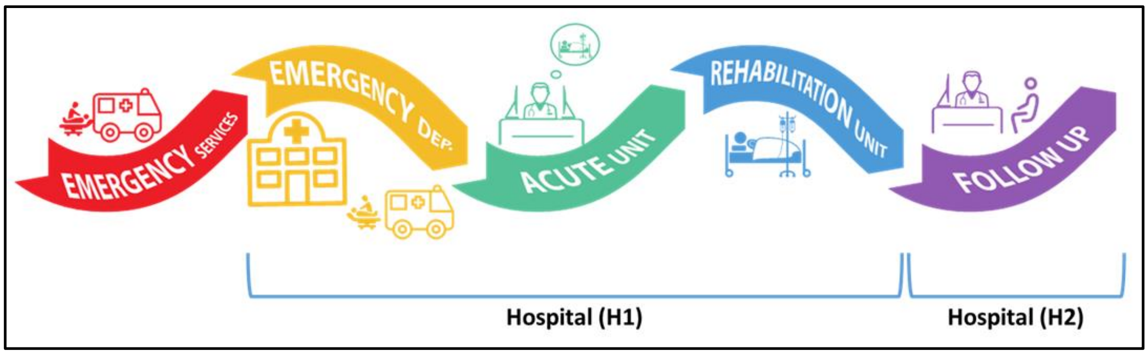


Figure 14: e-Health use case

5.2.4 Applications

The e-health scenario described at the beginning of this chapter enable the possibility of a lot of new applications **to enrich knowledge and to enhance care exploiting health data**.

Applications can be designed for all the actors considered:

- patients / citizens
- medical doctors and healthcare professionals in general
- medical direction administrative staff and administrative personnels in general
- biomedical researchers within and outside hospitals

and also for other different involved stakeholders: for instance we can mention marketing applications or also applications oriented to the pharmaceutical industries.

All the applications that can be designed go in the direction of a **digital and data-driven healthcare**.

5.2.4.1 "Fast Clinical History" app

One of first concrete applications that can be designed using DITAS platform is related with the healthcare business process above mentioned where we consider a patient with stroke, assisted in emergency care settings (hospital emergency dept.).

The idea is to have an application, to be used in the emergency department by the emergency medical doctor that gives the possibility of an **easy and fast way to discover patient's clinical history**.

Particularly, for e-health use case, in which a woman with a stroke is carried to the hospital emergency dept., the hospital emergency medical doctor can consult the past patient's health data related to the specific *Neurological Event* (stroke) using the ICD (*International classification of diagnosis*) code that identify the disease. This solution is fast and easy and allows medical staff to promptly organize the first aid in case of critical patients.

Going into details, the medical doctor can look for "*Serena Rossi*"+"*birth date*" and for the ICD "434.91 = ischemic stroke" acquiring a summary of her medical data related to the pathology under interest; in this way the doctor owns the information related to the patient's medical situation and can guarantee a proper treatment, also composing a team made of the appropriate specialists (e.g. the Emergency Dept. internist, cardiologist, neurologist, aesthetician, etc.).

The medical doctor can even look for patient's health data before patient arrival in hospital because the emergency dept. is warned by the ambulance, or emergency medical services in general, about patient, with specific characteristics, arrival.

To know the patient's clinical history is fundamental to discover possible further illness or chronic conditions and to ensure that the patient does not suffer of allergies; indeed, the first aid protocol for a stroke, may vary in case of other pathologies or allergies. For example, in case of renal failure the cranium Computerized Tomography scan (the traditional exam made in case of stroke) can be replaced with a Magnetic Resonance Imaging in order to avoid contrast agent that can aggravate kidneys condition.

Moreover the examinations (*e.g.* blood tests, bio images etc.) made in the previous 60 days of the stroke circumstance can give a general overview of the clinical condition and quite often it is not necessary to repeat them.

These aspects are very important for the stroke case, because faster the decided therapy is provided, better patient response is expected.

The idea of the "fast clinical history app" is that the health data are classified in main categories (past hospitalizations, past surgeries, allergies) as shown in the following figure:

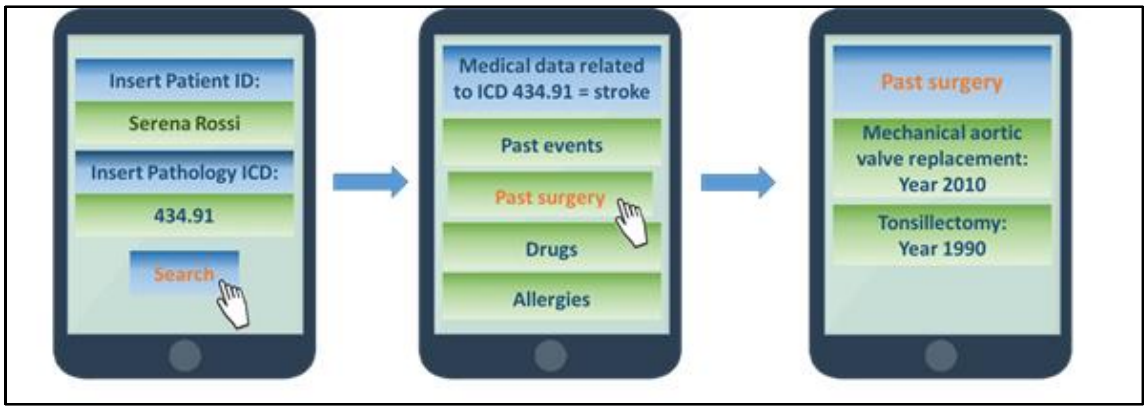


Figure 15: "fast clinical history" exemplificative interface

For DITAS e-health use case, the doctor discovers in advance that the patient owns a cardiac mechanical valve that has been inserted more than ten years before; this is possible because the medical doctor can select "past surgeries" and the app interface will show the patient's past surgical operations. As a result, the doctor can imagine a correlation between the neurological event and the previous surgery. Indeed, the mechanical valve is a risk factor for thrombosis.

Moreover the "fast clinical history" application is also able to provide access to the patient's data by textual research of keywords. This is possible thanks to a textual research that returns a list of examinations or events related to the typed keywords (figure 16).

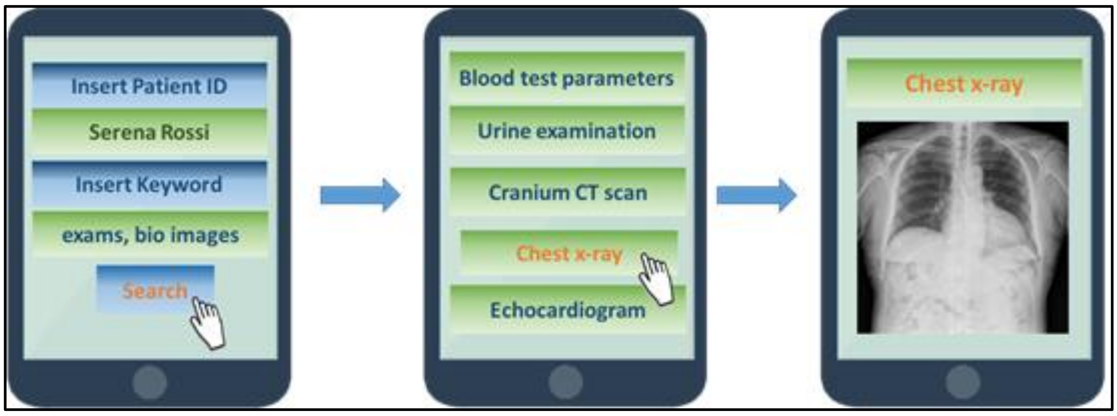


Figure 16: "fast clinical history" exemplificative interface

6 Conclusions

The document has established the guidelines for a proper development strategy, code managing and component integration. With the recommendations and best practices the document defines, the development teams can start working following a common strategy that eases the integration of the components in the near future. Also, with the best practices defined for the building and deployment stages, a fast development feedback is automated, where developer can know if new versions break critical things in dependent components or at system level.

Regarding use cases, a cloud testbed (over which the real world use cases will deploy their infrastructure and data needs for demonstrating the framework) has been introduced. Finally, the two use cases have been detailed, pointing out the common problems that real world data intensive applications have or are starting to experience, and detailing demonstrator applications for both of them.

7 References

[1] Github - https://github.com

[2] Git - https://git-scm.com

[3] GitFlow - https://datasift.github.io/gitflow/IntroducingGitFlow.html

[4] Trunk-based development - https://trunkbaseddevelopment.com

[5] Successful Git branching model - http://nvie.com/posts/a-successful-git-branching-model

[6] http://luci.criosweb.ro/a-real-life-git-workflow-why-git-flow-does-not-work-for-us/

[7] Jenkins - https://jenkins.io

[8] Jenkinsfile - https://jenkins.io/doc/book/pipeline/jenkinsfile/

[9] Jenkins Pipelines - https://jenkins.io/doc/book/pipeline/

[10] Savvy Data Systems - http://www.savvydatasystems.com/

[11] Trunk-based Development vs. Git Flow - https://goo.gl/1UJ5QV

[12] D1.1 – Initial architecture document with market analysis, SotA refresh and validation approach

[13] Blue Ocean – https://jenkins.io/projects/blueocean/

[14] Custom Tools - https://wiki.jenkins.io/display/JENKINS/Custom+Tools+Plugin

[15] Failsafe - https://maven.apache.org/surefire/maven-failsafe-plugin/

[16] Surefire - http://maven.apache.org/surefire/maven-surefire-plugin/

[17] Ansible plugin - https://wiki.jenkins.io/display/JENKINS/Ansible+Plugin

[18] Cucumber Reports - https://wiki.jenkins.io/display/JENKINS/Cucumber+Reports+Plugin

[19] Git Changelog – https://wiki.jenkins.io/display/JENKINS/Git+Changelog+Plugin

[20] Cobertura - https://wiki.jenkins.io/display/JENKINS/Cobertura+Plugin

[21] Checkstyle - https://wiki.jenkins.io/display/JENKINS/Checkstyle+Plugin

[22] FindBugs - https://wiki.jenkins.io/display/JENKINS/FindBugs+in+plugins

[23] Maven – https://maven.apache.org/[24] npm - https://www.npmjs.com/

[25] Yarn - https://yarnpkg.com/en/

[26] Gulp - https://gulpjs.com/

[27] Ansible - https://www.ansible.com/

[28] Minion Multi-Cloud Object Storage - https://www.minio.io/

[29] D2.1 DITAS Data Management

[30] European General Data Protection Regulation 2016/679 - http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679