Horizon 2020 Program (2014-2020)

Big data PPP

Research addressing main technology challenges of the data economy

Industrial-Driven Big Data as a Self-Service Solution

D5.2: Big-Data-as-a-Self-Service Test and Integration Report (first version)[†]

Abstract: The current deliverable presents the first version of Big-Data-as-a-Self-Service Test and Integration Report. The work gives important perception towards the release of the envisioned I-BiDaaS solution, ensuring a smooth and effective integration of the separate I-BiDaaS components. It also presents an integration plan which details how and when the individual technical elements of the I-BiDaaS solution will be adapted and integrated in a common solution.

Contractual Date of Delivery	31/12/2018
Actual Date of Delivery	31/12/2018
Deliverable Security Class	Public
Editor	*Vassilis Chatzigiannakis (ITML)*
Contributors	*ITML, CAIXA, SAG, BSC, UNSPMF,UNIMAN, AEGIS*
Quality Assurance	*Raül Sirvent (BSC)* *Burak Karaboga (ATOS)* *Kostas Lampropoulos (FORTH)*

[†] *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780787.*

The *I-BiDaaS* Consortium

Foundation for Research and Technology – Hellas (FORTH)	Coordinator	Greece
Barcelona Supercomputing Center (BSC)	Principal Contractor	Spain
IBM Israel – Science and Technology LTD (IBM)	Principal Contractor	Israel
Centro Ricerche FIAT (FCA/CRF)	Principal Contractor	Italy
Software AG (SAG)	Principal Contractor	Germany
Caixabank S.A. (CAIXA)	Principal Contractor	Spain
University of Manchester (UNIMAN)	Principal Contractor	United Kingdom
Ecole Nationale des Ponts et Chaussees (ENPC)	Principal Contractor	France
ATOS Spain S.A. (ATOS)	Principal Contractor	Spain
Aegis IT Research LTD (AEGIS)	Principal Contractor	United Kingdom
Information Technology for Market Leadership (ITML)	Principal Contractor	Greece
University of Novi Sad Faculty of Sciences (UNSPMF)	Principal Contractor	Serbia
Telefonica Investigation y Desarrollo S.A. (TID)	Principal Contractor	Spain

Document Revisions & Quality Assurance

Internal Reviewers

1. *Raül Sirvent (BSC)*
2. *Burak Karaboga (ATOS)*
3. *Kostas Lampropoulos (FORTH)*

Revisions

Version	Date	By	Overview
0.0.6	27/12/2018	Internal Reviewers	Final review and approval
0.0.5	24/12/2018	Editor	Second draft
0.0.4	17/12/2018	Internal Reviewers	Comments on the first draft
0.0.3	10/12/2018	Editor	First draft
0.0.2	29/10/2018	Dusan Jakovetic	Comments on ToC
0.0.1	18/10/2018	Editor	ToC

Table of Contents

List of Abbreviations

AVT: Advanced Visualisation Toolkit
BDVA: Big Data Value Association
CI: Continuous Integration
CPU: Central Processing Unit
CVS: Concurrent Versions System
DFP: Data Fabrication Platform
E2E: End-to-End
GPU: Graphics Processing Unit
GUI: Graphical User Interface
ISTQB: International Software Testing Qualifications Board
IOPs: Input/output operations per second
IoT: Internet of Things
JSON: JavaScript Object Notation
MIT: Massachusetts Institute of Technology
MQTT: Message Queuing Telemetry Transport
MVP: Minimum Viable Product
PoCs: Proof-of-concepts
QA: Quality Assurance
RTC: Real-time Computing
SCM: Source Control Management
SQL: Structured Query Language
SVN: Subversion
SW: Software
TDF: Test Data Fabrication
UAT: User Acceptance Testing
UM: Universal Messaging

List of Figures

List of Tables

Executive Summary

The current deliverable presents the first version of Big-Data-as-a-Self-Service Test and Integration Report. The work gives important perception towards the release of the envisioned I-BiDaaS solution, ensuring a smooth and effective integration of the separate I-BiDaaS components, taking into consideration their availability, their interoperability potential, their scalability and performance requirements.

The first part of the document (Section 2) presents an integration plan detailing how the individual technical elements of the I-BiDaaS solution will be adapted and integrated in a common solution.

Within the framework of solid and efficient development management throughout the whole software development lifecycle, a number of software tools will be employed to empower the integration activities. The activities -and the software tools that support them- that will be used in I-BiDaaS Integration methodology are summarized below:

- Redmine: Project Organization and Bug/Issue tracking
- Git: Code Source Control
- Jenkins: Continuous integration
- Sonarqube: Code Quality Assurance

In the first part, we also define the set of hardware and software requirements for each component. These requirements are used for creating the infrastructure that supports the MVP.

The second part (Section 3) is dedicated to the MVP (Minimum Viable Product). The MVP is based on one of the use cases specified in D1.3 (Positioning of I-BiDaaS), the "Analysis of relationships through IP address" from CAIXA. It uses a synthetic dataset generated by CAIXA and IBM that aims to be as realistic as possible to the data used in CAIXA for establishing new customers relationships based on the connections of its customers to the online banking services.

The MVP is divided in a batch processing and a stream processing subcase. In the batch processing subcase, we analyze the synthetic dataset in order to find users connecting repeatedly from the same IPs, in a short period of time (e.g. family members leaving in the same house, using the same network). The output of the batch processing is a list of user groups. In the stream processing subcase, we create random transactions of users from the pool of user ids in the first synthetic dataset, and check in real-time whether they are related (whether they belong to the same group or not). The goal of the MVP is to kick-start the integration between the I-BiDaaS components and their distinct technologies.

1 Introduction

1.1 Overview

The current deliverable presents the first version of Big-Data-as-a-Self-Service Test and Integration Report. The work gives important perception towards the release of the envisioned I-BiDaaS solution, ensuring a smooth and effective integration of the separate I-BiDaaS components, taking into consideration their availability, their interoperability potential, their scalability and performance requirements. It also sets the basis for the integration plan which details how and when the individual technical elements of the I-BiDaaS solution will be adapted and integrated in a common solution.

1.2 Relation to other Tasks and Work Packages

In general, this work will provide important perceptions for the implementation and deployment of the I-BiDaaS's integrated framework, a large-scale computational infrastructure that allows real time decision-making supporting analytics. The outcome of the work will be directly exploited in the next releases of "Big-Data-as-a-Self-Service Test and Integration Report" (D5.4 and D5.6 respectively).

Besides, there is a close interrelation between this deliverable and the Task 1.3 (Conceptual architecture specification) and Task 1.4 (Experimental protocol specification) in WP1 (Setting the scene: Baseline framework). It has been initially stated that the outcomes of Task 1.3 will be the starting point for Task 5.3 (Integration towards Big-Data-as-a-Self-Service). In particular, in Task 1.3 a thorough understanding of the challenges, technologies and the state of the art in terms of data economy, big data processing and the requirements of the end users have been successfully addressed. Thus, D5.2 has been structured based on some important insights taken from T1.3 (Conceptual architecture specification).

Furthermore, D5.2 is closely connected to Task 1.4 as the final release of the envisioned I-BiDaaS solution should fulfill all the key quality tests with respect to the industrial-validated benchmarks defined in Task 1.4. Finally, the I-BiDaaS evaluation and impact analyses (Task 6.3) should be closely aligned with Task 5.3 and take some important insights following the I-BiDaaS prototype releases (D5.2, D5.4, D5.6).

1.3 Contribution to the Scientific and Business Objectives

Having a central and core position within the whole concept of I-BiDaaS solution, D5.2 and the related versions contribute towards the following I-BiDaaS objectives:
Objective 1: Develop, **validate, demonstrate**, and support, a complete and solid big data solution that can be easily configured and adopted by practitioners.
Objective 2: Construct a safe environment for methodological big data experimentation, for the development of new products, services, and tools.
Objective 3: Develop data processing tools and techniques applicable in real-world settings and **demonstrate** significant increase of speed of data throughput and access.

1.4 Structure of the Document

This deliverable is divided into the following sections: **Section 2** provides the testing and integration plan ensuring the development of various tools and components and their further

integration within the final I-BiDaaS platform. **Section 3** is dedicated to the MVP use case description and summarizes the MVP components and technologies. Finally, **Section 4**, gives the overall conclusions and the next steps.

2 Testing and Integration Plan

The goal of the I-BiDaaS Test and Integration Plan is to ensure that various tools and components are developed consistently and the final I-BiDaaS platform is well integrated. This section defines the basis for the integration of components developed, and the hardware infrastructure used in the I-BiDaaS project. It is based on the I-BiDaaS architecture and use cases described in D1.3 (Positioning of I-BiDaaS).

In this document, we are presenting an integrated test plan for WP2 – WP5. Integration and Testing is explicitly decoupled from the "development of the other components". That is, development of each SW component (Batch processing, Streaming analytics, Resource management, Visualisation, etc.) follows its own methodology. Some of them (especially the ones that are not open-source) can be considered as “black boxes”, that will be integrated into the I-BiDaaS platform.

2.1 I-BiDaaS Continuous Integration Plan

2.1.1 Continuous Integration

Continuous Integration (CI), is a cornerstone of modern software development. In fact, it is a real game changer—when Continuous Integration is introduced into a project, it radically alters the way teams think about the whole development process. It has the potential to enable and trigger a series of incremental process improvements, going from a simple scheduled automated build right through to continuous delivery into production. A good CI infrastructure can streamline the development process right through to deployment, help detect and fix bugs faster, provide a useful project dashboard for both developers and non-developers, and ultimately, help teams deliver more real business value to the end user[1].

CI is a software development practice where team members/developers integrate their work and involves testing each modification a developer introduces to the codebase automatically, immediately after they occur[2]. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible[3] and helps software developers to complete tasks in a timely and predictable manner. In this context, it is an essential step that should be adopted in the frame of any new project as it serves as a practice of merging all developer working copies, thus allowing team members to develop cohesive software.

Below, we present a list of important benefits that CI brings along:

- Fast and easy integrations
- Increased visibility that enables greater communication
- Early identification of issues
- Spending less time debugging and more time adding features
- Building a solid foundation
- Reducing integration problems and providing easier software delivery

[1] https://www.bogotobogo.com/DevOps/Jenkins/images/Intro_install/jenkins-the-definitive-guide.pdf
[2] https://blog.getty.io/importance-of-continuous-integration-on-software-development-30ab74c61c1
[3] https://martinfowler.com/articles/continuousIntegration.html

CI is a popular practice in agile methodology as the CI rule states that programmers never leave anything unintegrated. This helps the software development teams be truly agile and adapt to rapid data changes, ensuring that software under development is in constant sync.

In CI, one of the first steps is to automate the building process via introducing automated testing into the building process, identify the major areas where things go wrong and get automated tests to expose those failures. The continued integrated process is depicted in the following diagrams revealing the intermediate steps.

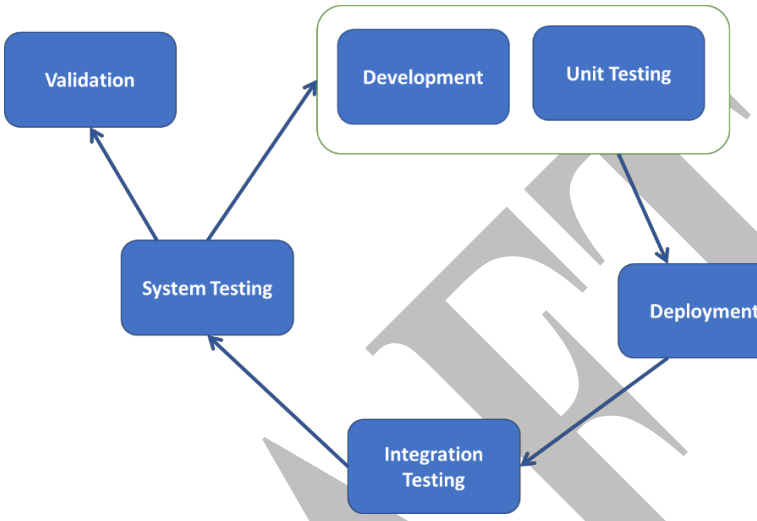

Figure 1. Continuous Integration

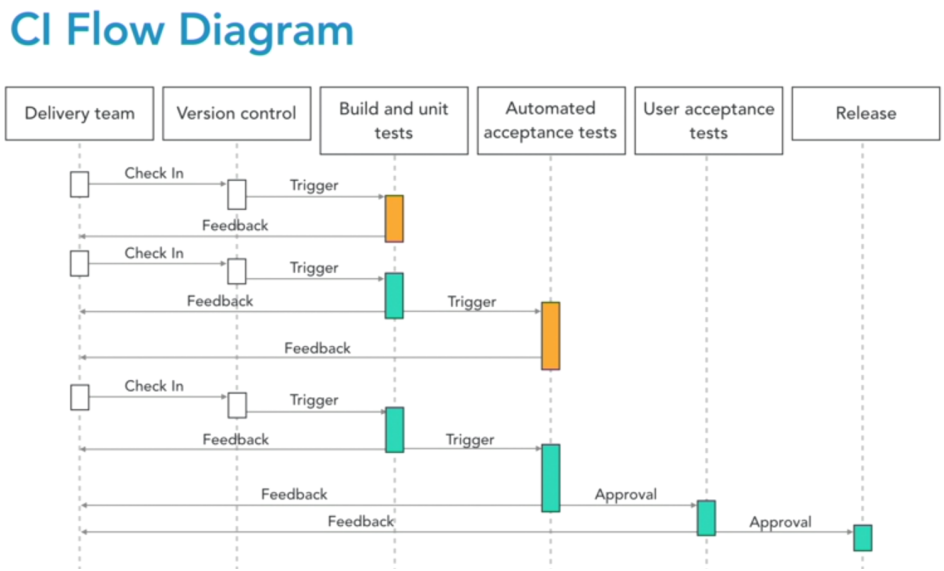

Figure 2. CI Flow Diagram[4]

[4] https://www.linkedin.com/learning/devops-foundations-continuous-delivery-continuous-integration/introducing-our-delivery-pipeline

2.1.2 Planned releases of I-BiDaaS solution

There are three major releases planned:

- A proof of concept demonstration (Minimum Viable Product (MVP) - M12)
- A 1st complete prototype – M18
- A 2nd prototype – M30

CI essentially starts with the delivery of the MVP and is supposed to last until the delivery of the final prototype. Part of this deliverable is to identify and prepare the infrastructure and procedures needed for supporting CI from M12 to M30.

2.1.3 CI Software

Within the framework of solid and efficient development management throughout the whole software development lifecycle, a number of software tools will be employed so as to empower the integration activities. The software tools that will be used in I-BiDaaS Integration methodology (for integration testing also) are given below:

Project Organization: Redmine

Source Control System: Git

Continuous Integration: Jenkins

Quality Assurance: Sonarqube

Bug Tracking: Redmine

Figure 3. Software development

These tools have been selected based on the following high-level requirements:

- Source control for collaboration, storing/restoring versions, staging, backup and enabling more advanced activities like monitoring code quality.
- Issue/bug tracking for better collaboration between different teams.
- Code quality assurance for different programming languages.
- Continuous integration and reporting.

2.1.3.1 Project Organization Software: Redmine

Redmine[5] is an open-source, flexible project management solution specializing in software development projects. It is a cross-platform and cross-database solution and allows users to manage multiple projects within the system. The issue management system allows users to define the status of each issue and they can set the priority on a need basis. Additionally, it permits hyperlinking information between an issue-tracking database, revision control and wiki content.

[5] https://www.redmine.org/

Redmine will be used in I-BiDaaS to track issues and feature requests. The main features of Redmine include:

- Multiple projects support
- Flexible role-based access control
- Flexible issue tracking system
- Gantt chart and calendar
- News, documents & files management
- Feeds & email notifications
- Per project wiki
- Per project forums
- Time tracking
- Custom fields for issues, time-entries, projects and users
- SCM integration (SVN[6], CVS[7], Git[8], Mercurial[9] and Bazaar[10])

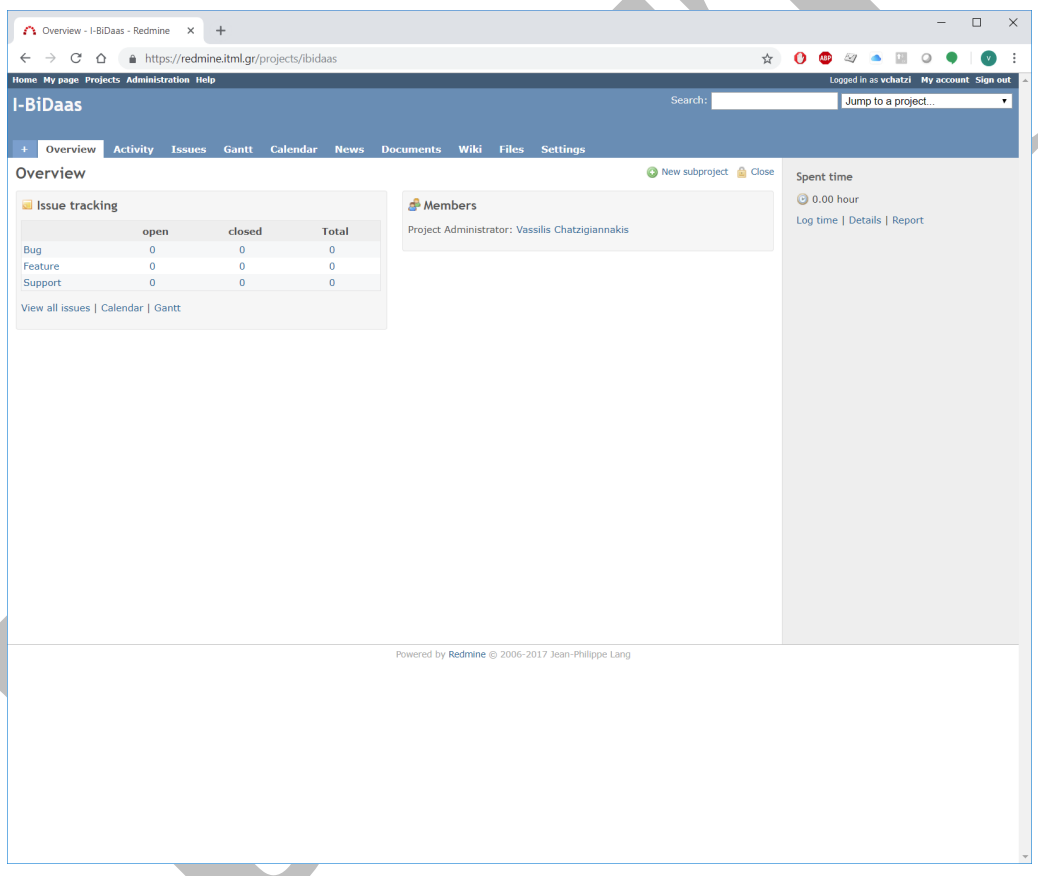

Figure 4. Redmine screenshot

2.1.3.2 Source Control: Git

Source control systems, also known as version control systems (VCS) are used to track changes to software development projects and allow team members to change and collaborate

[6] http://subversion.apache.org/
[7] http://www.nongnu.org/cvs/
[8] https://www.git-scm.com/
[9] https://www.mercurial-scm.org/
[10] http://bazaar.canonical.com/en/

on the same files, to streamline the development process, manage code for multiple projects, and maintain a history of code changes. These systems allow developers to work simultaneously on code and isolate their own work through what are known as branches. There are a lot of Source Control Management technologies in place, many of them distributed under GNU Public Licence. Git[11], Mercurial[12], AWS CodeCOmmit[13], HelixCore[14] Subversion[15]are some of the most widely used technologies.

In I-BiDaaS project, Git will be used, since it fits all our needs according to the requirements. Git is a relatively new, free, open source and fast-rising VCS. It is primarily used for software development, but it is designed to handle everything from small to very large projects with speed and efficiency. As a distributed revision control system, it is aimed at speed, data integrity, and support for distributed, non-linear workflows. Git allows and encourages to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.

As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version tracking abilities, independent of network access or a central server.

One thing that sets Git apart from other tools is that it's possible to quickly stage some of the files and commit them without committing all of the other modified files in the working directory or having to list them on the command line during the commit. This allows to stage only portions of a modified file (Figure 5). Of course, Git also makes it easy to ignore this feature.

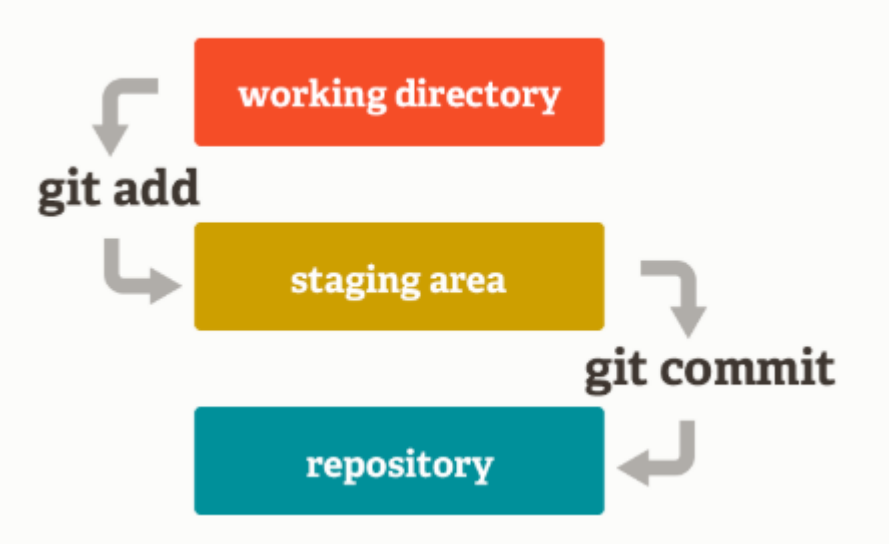

Figure 5: Git staging area

[11] https://git-scm.com/
[12] https://www.mercurial-scm.org
[13] https://aws.amazon.com/codecommit/
[14] https://www.perforce.com/products/helix-core
[15] http://subversion.apache.org/

2.1.3.3 Quality Management Software: SonarQube

Regardless of development methodologies, languages, frameworks or tools, enforcing a high code quality is a way to achieve faster and easier development, testing and maintenance. Code quality can be measured in a number of different ways, but some of the most important aspects are[16]:

- **Readability, consistency**—how easy it is to read and understand sections of the code; this includes code clarity, simplicity, and documentation.
- **Predictability, reliability, and robustness**—software behavior should be predictable, and not prone to hidden bugs.
- **Maintainability and extensibility**—fixing, updating and improving software should be as simple as possible, not inherently complex.

Within the I-BiDaaS project, Sonarqube[17] will be utilized aiming to enhance the code quality and thus ensuring the software reliability, testability, sustainability and long-term maintenance. It's by far the most powerful and open source platform developed by SonarSource for continuous inspection of code quality. It can be used to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on 20+ programming languages.

SonarQube provides the capability to monitor the health of the application and highlights the newly introduced issues. The code analysers, reporting tools, defect hunting modules and TimeMachine are the core functionality that the Sonar provides. It also embarks a plugin mechanism enabling the community to extend the functionality (more than 35 plugins available), making Sonar the one-stop-shop for source code quality by addressing the needs of developers.

The main features of Sonarqube, which cover our requirements, are:

- Continuous Inspection practice: SonarQube provides the capability to not only show the health of an application but also to highlight issues newly introduced. SonarQube can record metrics history and provides evolution graphs. SonarQube's provides fully automated analysis and integration with Maven[18], Ant[19], Gradle[20], MSBuild[21] and continuous integration tools (Atlassian Bamboo[22], Jenkins[23], Hudson[24], etc.)
- Supports more than 20 Programming Languages including Java, C#, C/C++, T-SQL, TypeScript, JavaScript, and COBOL, SonarQube offers a unique solution to cover large portfolios of applications.
- Supports security vulnerabilities detection: Examples include SQL injection, hard-coded passwords and badly managed errors.
- Explores all execution paths: SonarQube relies on several path-sensitive dataflow engines and thus code analysers explore all possible execution paths to spot the

[16] https://medium.com/@mkt_43322/why-is-code-quality-such-a-big-deal-for-developers-91bdace85d44
[17] https://www.sonarqube.org
[18] https://maven.apache.org/
[19] https://ant.apache.org/
[20] https://gradle.org/
[21] https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2017
[22] https://www.atlassian.com/software/bamboo
[23] https://jenkins.io/
[24] http://hudson-ci.org/

trickiest bugs. Issues raised by SonarQube are on either demonstrably wrong code, or code that is more likely not giving the intended behaviour.

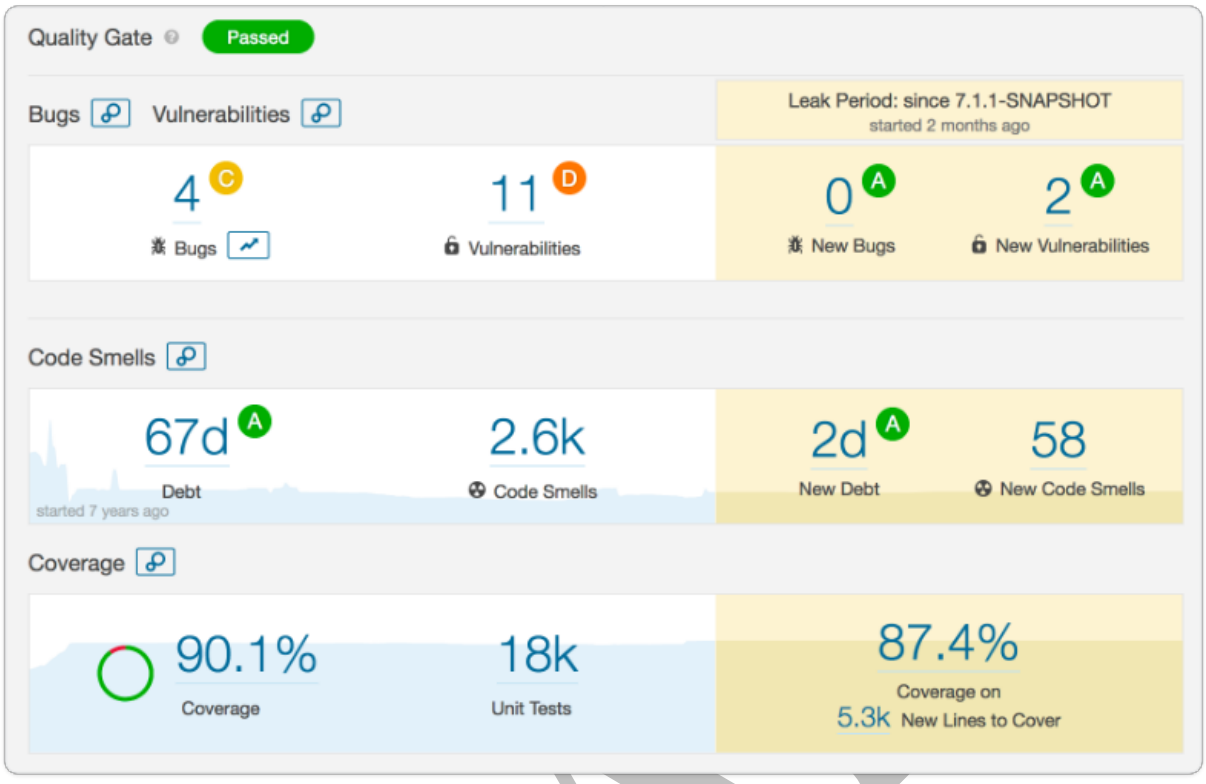

Figure 6. SonarQube interface[25]

2.1.3.4 Continuous Integration Software: Jenkins

Jenkins[26] is a popular tool designed for performing continuous integration and build automation. Jenkins is a Java-based, open source, free software released under the MIT License.

[25] Screenshot from https://www.sonarqube.org/
[26] https://jenkins.io/

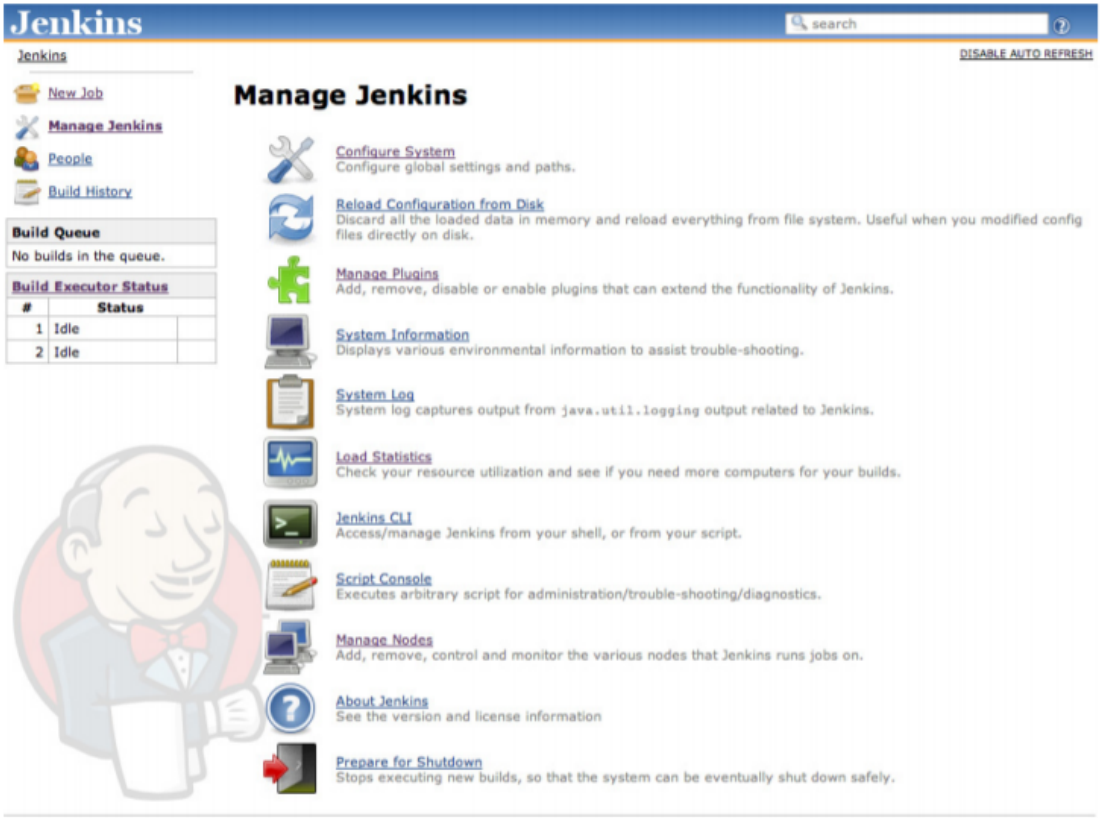

Figure 7. Jenkins management screen

Jenkins is easy to use, the user interface is simple, intuitive, and visually appealing, and Jenkins as a whole has a very low learning curve[27]. The basic functionality of Jenkins is to execute a predefined list of steps, e.g. to compile Java source code and build a JAR from the resulting classes. The trigger for this execution can be time or event based. For example, every 20 minutes or after a new commit in a Git repository. Jenkins monitors the execution of the steps and can stop the process if one of the steps fails and/or send notification about the build success or failure. The following flowchart demonstrates a very simple workflow of Jenkins operation.

[27] https://www.bogotobogo.com/DevOps/Jenkins/images/Intro_install/jenkins-the-definitive-guide.pdf

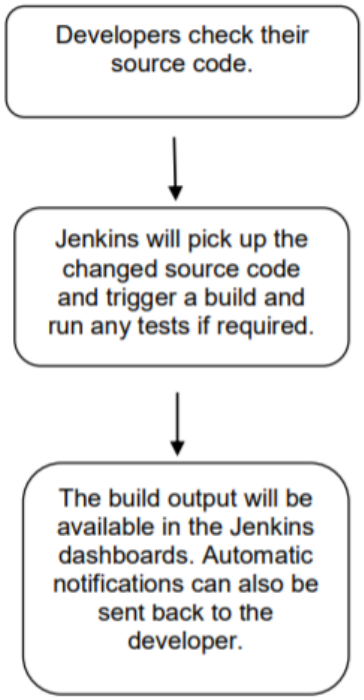

Figure 8. Simple workflow of Jenkins operation[28]

Jenkins is a server-based system running in a servlet container such as Apache Tomcat[29]. It supports SCM tools including AccuRev[30], CVS[31], Subversion[32], Git[33], Mercurial[34], and many others. It can execute Apache Ant[35], Apache Maven[36] and sbt[37] based projects as well as arbitrary shell scripts and Windows batch commands.

Jenkins functionality can be extended to projects written in languages other than Java with the use of plug-ins. Plug-ins can help to integrate Jenkins with most version control systems and big databases and add new functionalities. Additionally, with the support of plug-ins, builds can generate test reports in various formats. These reports will be incorporated in future Big-Data-as-a-Self-Service Test and Integration Reports.

2.2 System Requirements

In order to determine the overall software and hardware of the I-BiDaaS platform, a set of requirements has been gathered (via the table presented in Appendix I) from each component leader.

[28] https://www.tutorialspoint.com/jenkins/jenkins_tutorial.pdf
[29] http://tomcat.apache.org/
[30] https://www.microfocus.com/products/change-management/accurev/
[31] https://www.nongnu.org/cvs/
[32] http://subversion.apache.org/
[33] https://www.git-scm.com
[34] https://www.mercurial-scm.org/
[35] https://ant.apache.org/
[36] https://maven.apache.org/
[37] https://www.scala-sbt.org/

2.2.1 Environments

For each component, there will be both a testing and a production environment. Testing environment will be used to test any new changes before those are integrated with the production environment. Following the principles of CI, changes will be constantly deployed by each team, allowing the early detection of bugs and issues, thus reducing the total time needed to resolve them. Any new change will first be deployed on the test environment and if it passes the tests, it will then be deployed in the production environment.

2.2.2 Specifications

The testing and production environment requires both Windows and Linux servers, which will host the different components. In terms of hardware, all components require multi-core processors, in order to provide results efficiently.

2.2.2.1 Hardware Specification

Hardware requirements for the production environments of the platform were collected for each component in terms of processing power, memory and storage space. These requirements are briefly presented in the table below.

Table 1. Hardware Requirements

Component(s)	Platform	Cores	RAM	Storage
IBM InfoSphere® Optim Test Data Fabrication	Linux server (x86_64)	4	8GB	20GB
Hecuba (2 nodes)	Linux server (x86_64)	4 per node	4GB per node	20GB per node
Software AG APAMA Server	Linux server (x86_64)	4	8GB	50GB
Software AG Universal Messaging	Linux server (x86_64)	4	8GB	50GB
Visualisation Toolkit / I-BiDaaS Dashboard	Linux server (x86_64)	2	4GB	30GB
Database (Cassandra)	Linux server (x86_64)	4	4GB	100GB
Integration Tools	Linux server (x86_64)	4	16GB	100GB
	TOTAL	30	56 GB	400 GB

2.2.2.2 Technology stack

The following table contains a brief list of the technologies that will be used.

Table 2. Technologies

Technology	Minimum Version
SonarQube	7.4
Redmine	3.4.6
Jenkins	2
Cassandra Database	3.11
Mysql server	5.6
Python modules: Numpy and cassandra-driver	-
Apache Tomcat	8
Apache Web server	2.4
Java	8
Ruby on Rails	2.0
Python	3.5
Node.js	6.14
Centos Linux	7.x
Ubuntu Linux	16.04

The final versions of each element will be determined before the implementation begins but will be subject to change in case of any arisen compatibility issues.

2.2.3 Major system requirements

There are some requirements concerning the high-level functionality of the platform.

- All components need access to the internet, both for testing and deployment purposes, but also for component intercommunication and availability to the mobile and web applications.
- The platform needs to be accessible 24 hours per day, every day. Therefore, maintenance routines need to be planned in a way that will never hinder the platform's operation.

2.2.4 Continuous integration practices

Beyond the software and hardware requirements that will allow the continuous integration, there are some practices concerning the development teams that will make this process easier and the collaboration more productive.

- Broken and untested code should not be submitted.

- If the current built is not functional, no submits should be made, other than those towards the effort to repair it.
- After the new code has been submitted, wait until the built is complete in order to verify that it remains operational.

2.3 Testing

Testing is primarily a risk mitigation function. In short, the role of testing is to advise the project manager on the quality risks in the delivery, seeking to mitigate the greatest risks as early as possible and to provide information on the residual risks such that a rational judgement between cost, time and quality of the delivery can be taken.

Testing is defined by the International Software Testing Qualifications Board (ISTQB) as *'The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects'*. Testing terminology is defined by ISTQB Standard Glossary of Terms used in Software Testing[38].

2.3.1 Test Types

Table 3. Test types

Test type	Description
Unit and component test	To verify the software component against the lowest level of documentation defining that component e.g. detailed design specifications. Unit testing focuses on verifying the smallest testable elements of I-BiDaaS, namely by testing individual development objects (e.g. individual units of source code) as they are developed during the component build. More specifically, unit testing is performed at the low-level architectural layers
Component integration test	Supplier Component Integration Testing verifies that all the components of the supplier solution can interface to and interact correctly with all other components provided by the supplier according to the contractual solution specification documents produced by the supplier (e.g. functional designs, system specifications). Depending on the nature of the supplier's solution and how completely it fulfils the business requirement, it may be possible to verify some of the business requirements in this test phase. If this can be done, it is to be encouraged to detect significant business risks earlier and fix them with less effort than waiting for later test phases to detect the fault.
System integration test	System Integration Testing verifies that the delivered systems interface with each other and external organizations/systems as expected. This includes the testing of all interfaces including interface logic, interface transmission, and target system processing.
User acceptance	User Acceptance Testing (UAT) is a formal test that proves to the end user that the delivered system is fit for their purpose. The risk assessment shall determine the appropriate amount of regression testing to verify that

[38] http://istqb.org

test	unchanged parts of the system continue to behave as expected and the existing user expectations can be met.

2.4 I-BiDaaS Actors and their integration activities

The I-BiDaaS actors, integration activities together with integration components are listed in Table 4.

Table 4. I-BiDaaS actors and their integration activities

Actor	Integration Activities	Integrated Components
Foundation for Research and Technology – Hellas (FORTH)	Development of GPU-accelerated Analytics	Streaming Analytics
Barcelona Supercomputing Center (BSC)	Development of Hecuba Tools	Hecuba: Batch Processing Module
IBM Israel – Science and Technology LTD (IBM)	Development of Test Data Fabrication	IBM InfoSphere® Optim Test Data Fabrication
Centro Ricerche FIAT (FCA/CRF)	Pilot leader / Data provider	
Software AG (SAG)	Development of Streaming analytics & Messaging	• APAMA Server • Universal Messaging
Caixabank S.A. (CAIXA)	Pilot leader / Data provider	
University of Manchester (UNIMAN)	Resource Management and Orchestration	
Ecole Nationale des Ponts et Chaussees (ENPC)		
ATOS Spain S.A. (ATOS)	Resource Management and Orchestration	
Aegis IT Research LTD (AEGIS)	Development of Visualization	I-BiDaaS Dashboard
Information Technology for Market Leadership	Integration	

(ITML)		
University of Novi Sad Faculty of Sciences (UNSPMF)	Development of Batch Processing algorithms	Batch Processing Module
Telefonica Investigation y Desarrollo S.A. (TID)	Pilot leader / Use case and Data provider	

2.5 Integration Testing

2.5.1 Unit Testing

Each team is responsible for the unit testing of the component they are developing. Unit testing will verify the functions created to implement the business logic for each component, verifying the results.

2.5.2 UM Testing

The Universal Messaging (UM) handles almost all communication between the major components of the I-BiDaaS platform. This will simplify the integration testing up to a point, since even if a component is not ready, the associated message queue receiving messages for that component will be available, thus allowing at least to test the sending of messages. The communication concerning the UM must be tested for three important aspects:

- Each queue/topic must correctly deliver the messages to the correct components.
- Each component must have a worker retrieving the messages from the message queue/topic that is assigned to that component. The use of the message after it is retrieved does not belong in the scope of this part of testing.
- Each component must correctly implement the logic that sends the messages to the correct queues when attempting to communicate with another component.

This will verify that all messages will be delivered, that each component will be able to receive messages and that each component will be able to send messages.

Once the communication with the UM has been established and tested, the integration testing will proceed with testing the communication between pairs of components (where available) through the UM in order to verify that each request receives the appropriate response. The purpose is not to check the validity of the responses, but to ensure that they trigger the correct chain of messages between the various components, since in many cases a request to one component will be processed through another component, which will deliver the final response.

2.5.3 End to End Testing

The purpose of End-to-End (E2E) testing is to verify the desired functionality across the whole platform, offering insights on the performance and in this case of a multi-component system demonstrating the intended communication between these components. The design of E2E test will be based on the Use Cases provided in D2.1.

Successful completion of this test will confirm that the I-BiDaaS platform is working as intended both in software and hardware terms. Each test case will remain valid for as long as the tested functionality remains in the application and should be repeated after each update of the application in the future.

The E2E test will be performed by the QA team, using the test cases described below for each use case. In each case, the team must test the successful cases and the exceptions described for these cases. Any case that leads to non-intended results or disruption of the application flow, should be documented as detailed as possible including information about the process that led to it, the conditions and the platform. An attempt to recreate the error should also be made.

2.6 Key Quality Tests in relation to I-BiDaaS industry validated benchmarks

Based on the initial survey conducted during the planning phase, according to the I-BiDaaS experimental protocol, involving both technology providers and business partners, a number of quality features of the I-BiDaaS integrated solution (MVP) should be tested, including scalability, operational performance, data security, etc., (detailed in D1.3[39]).

Table 5. Planning for testing the quality of the I-BiDaaS integrated solution, according to I-BiDaaS experimental protocol

Quality Variable	Measuring mechanism	To be recorded by		
		End User	IT User	Technology Provider
Scalability	Speedup			✓
Operational Performance	Response time			✓
	Data Throughput (IOPs, Number of generated data records per time unit)			✓
	Resource Utilization (Disk Usage, CPU Usage)			✓
Availability	Uptime, % Timeouts			✓
Reliability	Data failure, Fault Tolerance			✓
Data Security	No of policy violations, Compliance with current security and privacy regulations and standards	✓	✓	✓
Privacy	No of policy violations, Compliance with current security and privacy regulations and standards	✓	✓	✓
Compliance	Measured against relevant standards			✓
Cost	Compared against on premise alternatives	✓	✓	

[39] I-BiDaaS, Positioning of I-BiDaaS, Project Deliverable D1.3, 21/09/2018, http://www.ibidaas.eu/sites/default/files/docs/Ibidaas-d1.3%20(1).pdf

The collection of quantitative data regarding the quality and especially the performance of the I-BiDaaS integrated solution and its components, involves the use of applicable benchmarks. Table 6, summarises a number of Big Data benchmarks that according to recent research[40] cover multiple layers (horizontal concerns) of the BDVA Reference Model[41] and are thus relevant to I-BiDaaS.

These benchmarks are classified into two categories: micro-benchmarks and application benchmarks. The micro-benchmarks stress test very specific functionalities of the Big Data technologies typically utilising synthetic data. The application benchmarks cover a wider and more complex set of functionalities involving multiple Big Data technologies and typically utilising real scenario data. From a practitioner perspective, the two categories of benchmarks are quite different. The micro-benchmarks are easy to set up and use as they typically involve one technology, whereas the application benchmarks require more time to set up and involve more technologies in the execution phase.

Table 6. Classification of applicable Big Data benchmarks

Category	Year	Name	Type	Domain	Data Type
Micro Benchmarks	2010	HiBench[42]	Micro-benchmark Suite	Micro-benchmarks, Machine Learning, SQL, Websearch, Graph, Streaming Benchmarks	Structured, Text, Web Graph
	2015	SparkBench[43]	Micro-benchmark Suite	Machine Learning, Graph Computation, SQL, Streaming Application	Structured, Text, Web Graph
	2013	BigDataBench[44]	Micro-benchmark Suite	Online service, offline analytics, graph analytics, artificial intelligence (AI), data warehouse, NoSQL and streaming	Graph, Network, Text, NLP, Web, Image, Audio, Spatio-Temp. Time Series, IoT, Structured, BI
	2010	YCSB[45]	Micro-benchmark	Cloud OLTP operations	Structured
	2017	TCPx-IoT[46]	Micro-	Workloads on typical IoT	Structured,

[40] DataBench 2018, DataBench Architecture, Project Deliverable D3.1, 07/07/2018, https://www.databench.eu/wp-content/uploads/2018/10/databench-d3.1-ver.1.0.pdf
[41] BDV SRIA, Big Data Value association, European Big Data Value - Strategic Research and Innovation Agenda, vers. 4.0, Oct. 2017, http://www.bdva.eu/sria
[42] HiBench Suite, https://github.com/intel-hadoop/HiBench
[43] SparkBench, https://github.com/CODAIT/spark-bench
[44] BigDataBench. http://prof.ict.ac.cn/BigDataBench.
[45] YCSB, https://github.com/brianfrankcooper/YCSB
[46] TPCx-IoT, http://www.tpc.org/tpc_documents_current_versions/pdf/tpcx-iot_v1.0.3.pdf

			benchmark	Gateway systems	IoT
Application Benchmarks	2015	Yahoo Streaming Benchmark[47]	Application Streaming Benchmark	Advertisement analytics pipeline	Structured, Time Series
	2013	BigBench/TPCx-BB[48]	Application End-to-end Benchmark	A fictional product retailer platform	Structured, Text, JSON logs
	2017	BigBench V2[49]	Application End-to-end Benchmark	A fictional product retailer platform	Structured, Text, JSON logs
	2018	ABench[50] (Work in Progress)	Big Data Architecture Stack Benchmark	Set of different workloads	Structured, Text, JSON logs

The correspondence between the above benchmarks to the BDVA Reference Model layers and the corresponding modules of the I-BiDaaS integrated solution is shown in Table 7. It should be noted that as this work is in progress, there will be further analysis of the inclusion of further appropriate benchmarks to ensure the quality testing of the different modules.

Table 7. Mapping between BDVA reference model's horizontal concerns and I-BiDaaS modules and applicable benchmarks

Applicable Benchmark	BDVA reference model layer	Corresponding I-BiDaaS module (s)
ABench	Data visualization and user interaction	Advanced visualization module (AEGIS+SAG); User interface (AEGIS)
HiBench, SparkBench, BigBench, BigBench V2, ABench	Data analytics	Batch processing module (UNSPMF+BSC); Streaming analytics module (SAG+FORTH)
HiBench, SparkBench, BigBench, BigBench V2, ABench	Data processing architectures	APAMA CEP, CPU Accelerated Analytics
To be defined	Data protection	FORTH commodity cluster –privacy preservation through commodity hardware (Intel SGx); DFP (IBM) for generation of realistic synthetic data when real data cannot be uploaded to cloud or similar systems

[47] YSB, https://github.com/yahoo/streaming-benchmarks
[48] BigBench, https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench
[49] Ahmad Ghazal et al. 2017: BigBench V2: The New and Improved BigBench. ICDE 2017: 1225-1236
[50] Todor Ivanov 2018, Rekha Singhal: ABench: Big Data Architecture Stack Benchmark. Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, Berlin, Germany, April 09-13, 2018

YCSB, TPCx-IoT HiBench, SparkBench, BigBench, BigBench V2, ABench	Data management	COMPSs runtime (BSC); ATOS resource management and orchestration module
BigDataBench	The Cloud and HPC (efficient usage of Cloud)	ATOS resource management and orchestration module

Using such benchmarks the quantitative testing of the I-BiDaaS system quality in parts and as a whole proceeds as follows. For each of the identified use case, sets of specific workload types (such as graph analytics, artificial intelligence, data warehouse, NoSQL, streaming) are selected. Then, the testing continues by a gradual increase of the complexity through: (a) experimentation with applicable benchmarks (see Table 6); (b) combinations of workloads which are more representative of the specific use cases; and (c) complex end-to-end benchmarking.

3 Minimum Viable Product (MVP)

3.1 MVP use case description

3.1.1 Use case rationale, description and data flows with reference to I-BiDaaS architecture

For the use case of I-BiDaaS MVP, the "Analysis of relationships through IP address" from CAIXA is selected. This is a synthetic dataset generated by CAIXA and IBM that aims to be as realistic as possible to the data used in CAIXA for establishing new customer relationships based on the connections of its customers to the online banking services.

The main objective of this use case is to validate and test the I-BiDaaS architecture and tools, and in particular, verify if the generated synthetic data can provide the same or additional insights. Moreover, it will also exam if the amount of relationships detected in the real data by CAIXA can be also detected by using synthetic data.

This will open the possibilities to CAIXA to break data silos and foster innovation in a sector that is very sensitive to data security aspects, speeding up the bureaucracy and flexibility to elaborate proof-of-concepts (PoCs) in order to evaluate new developments and the usage of novel tools.

3.1.2 Dataset selection and use case context

In the MVP use case selection process, some specific factors were taken into account. First of all, it had to be selected from one of the use cases already specified in D1.3 (Positioning of I-BiDaaS). Considering that set, another requirement was specified by the project technology providers. The use case also should be adaptable to test I-BiDaaS MVP architecture in batch and stream processing. That restricted very much the selection of the use case, and there was another constraint that was basically the availability and simplicity of the different datasets of the defined use cases. Hence, it was decided to select the "Analysis of relationships through IP address" use case provided by CAIXA as the one for the MVP. It was considered the best option because it is based in a single-table dataset but at the same time allows to test the different features demanded by technology providers.

As detailed in D2.1 (Data assets and formats), CAIXA stores information about their customers for security and fraud prevention reasons, verifying if the patterns of the customers interactions with the bank are legit or not (e.g. avoiding fraudulent bank transfers). For example, CAIXA extracts customers' relationships and store them by analyzing the data directly provided by the customers, such as family relations, same home address, etc. However, most of the social relationships of the users cannot be found only with this information directly provided by the customers.

Therefore, CAIXA also collects and analyses the information from the operations that customers perform (bank transfer, check their accounts, etc.) using channels such as mobile apps or online banking. That allowed CAIXA to find new kind of relationships between customers using their connection information (e.g. IP address). For example, a new "residence" relationship has been built considering concurrent connections to online banking using the same IP by two or more customers. This relationship should link people that live in the same house or connect usually from the same networks. The IP relation is somehow a physical relation, however, there are too many types of accesses making difficult to categorize

the relationship. The proposed way to find the relation tries to maximize the possibilities of a close relation, by defining a set of filters that limit those connection concurrencies and find only those ones that are trustworthy relationships between the two or more customers. The main constraints specified are:

- At first place, we discard all the IPs where a lot of users have been connected. In this way, we are discarding connections from public spaces such as libraries, campus, etc. but also aggregation bank tools that connect in the name of a lot of users and is not a clear relation.
- After that, we propose to establish relations in pairs of users for simplicity reasons. Then we set that the same 2 users have to be at least on 2 different IPs during a certain period of time (e.g. last month). At the end is a simple process that must take groups of a maximum of 2 users with the same IP, and at least two different IP for those same users in a specific period.

Consequently, in the MVP use case, a batch process is proposed first in order to discover the "residence" relationships between the customers that connected during the period of time under analysis.

Moreover, a second phase of the MVP use case is defined in order to complement it with stream data analytics, and continuing with the usual data processing that CAIXA performs in daily operations.

Once the relations between customers are established (and periodically updated), they are used for a first-step validation of any bank transfer executed through the different channels that CAIXA offers (e.g. online banking, ATM, etc.). In a real scenario, the bank transfers will be analyzed in order to find any relationship between customers that indicate that it is a legit transfer. Hence, in the framework of the MVP, an additional dataset is generated emulating real-time bank transfers' records as well. These records contain the sender and receiver identifiers and the amount of transferred money. In this second phase of the MVP use case, this generated data is analyzed looking for transactions between costumers that do not have "residence" relationship.

3.1.3 Use case data formats

As further explained in D2.1 (Data assets and formats), the main generated dataset provides synthetic data of connections of users to the online banking systems during a certain period of time (e.g. last month records). Extracting information from these customers' connections, a "residence" relationship between customers can be built, as a part of the social graph of the bank customers. The data will be synthetically generated taking as a pattern the real data coming from a set of restricted tables (relational database), which contain information related to the customers, their connections and the different operations they perform.

The generation of this synthetic dataset without the sensitive data of its customers will allow CAIXA to use it to speed up the process of sharing data with its external providers for deploying and validating proofs-of-concept of different use cases (e.g., potential fraud based on customers' relationships).

For the sake of simplicity, the volume of the dataset for the MVP is reduced to 1 million rows (our early tests show that this as a reasonable number), previously stored it in a SQLite database. The initial structure of the dataset defined in D2.1 contains a single table with 5 columns. However, in order to minimize the volume of information required to identify possible relationships between customers, only three columns of the table from the IP Address dataset are needed (FK_NUMPERSO, PK_ANYOMESDIA, IP_TERMINAL).

Table 8. IP Address dataset

Attribute	Description	Format	Example
FK_NUMPERSO	Identifier of the Person.	NUMBER	34523454
PK_ANYOMESDIA	Date (YYYYMMDD) of the connection of the user.	NUMBER	20180823
IP_TERMINAL	IP Address of the connection of the user.	VARCHAR2	10.8.2.22
FK_COD_OPERACION	Code of business operation done by the user.	VARCHAR2	CA.OFI.ContrataActivoConSol.MenuConSolicitud
PK_COD_ESTADO_OP	Code of the status of the operation done by the user.	VARCHAR2	KO

3.1.4 Bank Transfer Dataset

In the second phase of the MVP use case, a dataset is continuously generated emulating the bank transfer logs collected by CAIXA infrastructure. The main relevant information from these logs can be defined in traces containing only the sender id, receiver id and time of the transfer. The structure of the dataset will be the following.

Table 9. Bank transfers dataset

Attribute	Description	Format	Example
FK_NUMPERSO_ORIGEN	Identifier of the Sender of the Transfer.	NUMBER	34523454
FK_NUMPERSO_DESTINO	Identifier of the Receiver of the transfer.	NUMBER	34523454
PK_TSINSERCION	Date and time when the operation is executed.	TIMESTAMP	04/03/18 00:00:20,074000 00

3.2 MVP Architecture

As presented in section 3.1.2, the MVP use case is divided in batch processing and stream processing. In the batch processing subcase, the data flow is the following:

- A file of synthetic data in SQLite format is created by TDF. It is a single table database with the structure presented in Table 8.
- The file is imported in a central Cassandra Database as a Cassandra table with the same structure.

- Hecuba runs on the table and calculates the user relations between people. Results (pair of user ids) are stored in a new table in Cassandra and a file in CSV format.

In the stream processing subcase, the data flow is the following:

- Random transactions of users (from the pool of user ids created in the first synthetic dataset) are created, in the format depicted in Table 9. Each transaction is published as a simple string with comma separated values via MQTT in a topic called "caixa_transaction_pair" in UM.
- An APAMA application:
  - loads the user relations (groups) that were created in the batch processing subcase in memory
  - subscribes to caixa_transaction_pair topic and checks every incoming transaction to see if the pair of users are in a group
  - Every transaction that is between a "related" pair of users is sent as a new message to another topic called "caixa_transaction_related".

The Visualization Component provides a user interface for controlling experiments (an experiment is an instantiation of a specific use case) and for acquiring access to the results of the experiment.

To speed up the realization of the MVP and to integrate components that were lacking appropriate interfaces for the integration, an orchestrator service was created, especially for the MVP. The orchestration service acts as a middleware between the Visualization Component and the rest of the I-BiDaaS platform. It is worth to mention that the actual resources for the MVP implementation (including the allocation, configuration and maintenance of the required virtual machines) has been provided by ATOS.

The overall MVP architecture is presented in the following figure:

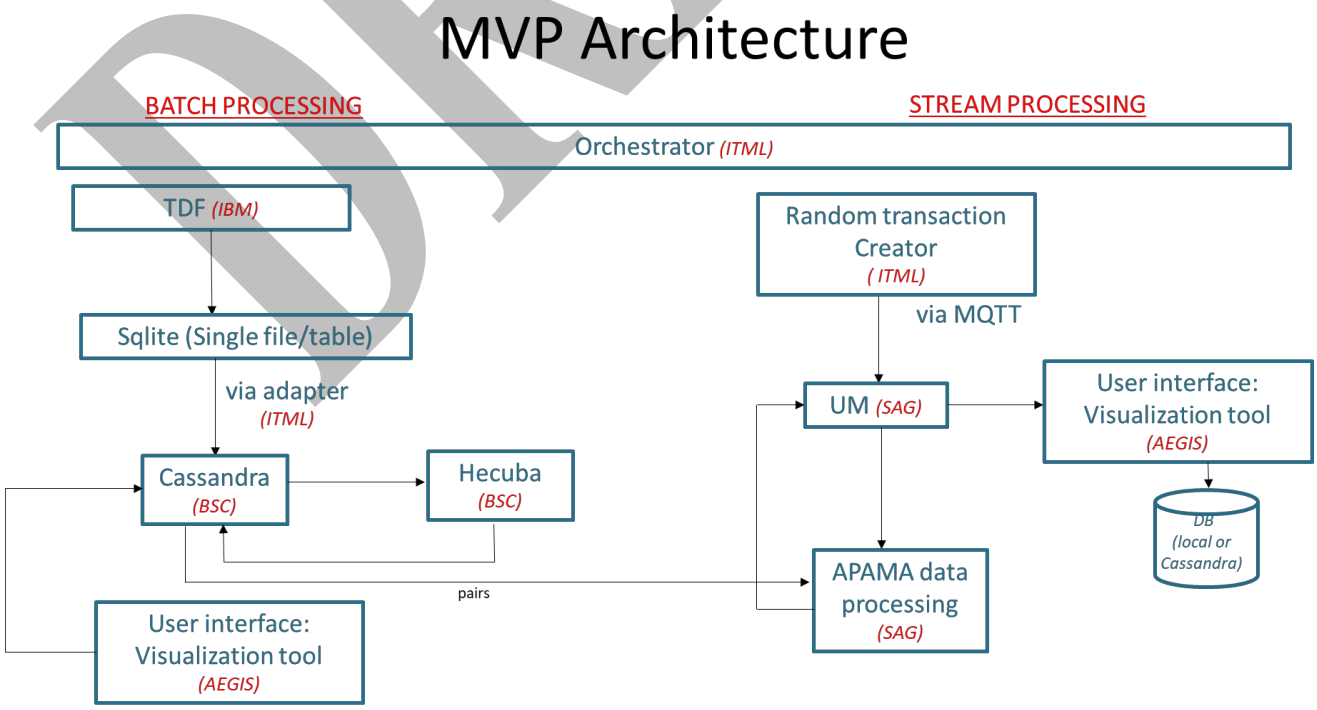

Figure 9. MVP architecture

3.3 MVP components and technologies

3.3.1 Universal Messaging

The CAIXA use case considered for the MVP specifies that data is fabricated according to the schemas defined by CAIXA with IBM's TDF. The data is written to a database and optionally simultaneously to a MQTT broker. In our case, this is Software AG's Universal Messaging component.

Even though a detailed description of the Universal Messaging component will be described in D2.4 and D2.6, (Universal Messaging Bus interim and final versions respectively) we will provide here a brief summary since the other deliverables are due M18 and M30, respectively. Universal Messaging is a Message Orientated Middleware product that guarantees message delivery across public, private and wireless infrastructures. Universal Messaging has been built from the ground up to overcome the challenges of delivering data across different networks. It provides its guaranteed messaging functionality without the use of a web server or modifications to firewall policy.

The Universal Messaging realm server is a heavily optimized Java process capable of delivering high throughput of data to large numbers of clients while ensuring latencies are kept to a minimum. In addition to supporting the client types described below, the Universal Messaging realm server has a number of built in features to ensure its flexibility and performance remains at the highest levels.

Universal Messaging clients support synchronous and asynchronous middleware models. Publish Subscribe and Queues functionality are all supported and can be used independently or in combination with each other. Universal Messaging clients can be developed in a wide range of languages on a wide range of platforms (Java, C# and C++ running on Windows, Solaris and Linux are all supported). Mobile devices and Web technologies such as Silverlight all exist as native messaging clients.

A Universal Messaging Realm is the name given to a single Universal Messaging server.

Universal Messaging realms can support multiple network interfaces, each one supporting different Universal Messaging protocols.

A Universal Messaging Realm can contain many Channels or Message Queues. Universal Messaging provides the ability to create clusters of realms that share resources within the namespace. Cluster objects can be created, deleted and accessed programmatically or through the Universal Messaging Enterprise Manager.

Objects created within a cluster can be accessed from any of the realms within the cluster and the Universal Messaging ensures that the state of each object is maintained by all realms within a cluster. The clustering technology used within Universal Messaging ensures an unsurpassed level of reliability and resilience.

Realms can also be added to one another within the namespace. This allows the creation of a federated namespace where realms may be in different physical location, but accessible through one physical namespace.

Universal Messaging Realms can support multiple communications interfaces, each one defined by a protocol a port. Universal Messaging Realms can be configured to bind to all network interfaces on a machine or specific ones depending on configuration requirements.

Specific SSL certificate chains can be bound to specific interfaces thus insuring clients always authenticate and connect to specific interfaces.

A Universal Messaging Realm is the name given to a single Universal Messaging server. It may contain many Channels or Message Queues while supporting multiple network interfaces each one defined by a protocol port. An interesting feature of the realms is that they can be added one to another within the namespace, allowing the creation of a federated namespace where realms may be in different physical location, but accessible through one physical namespace. The tool can be configured to bind to all network interfaces on a machine or specific ones depending on configuration requirements. Specific SSL certificate chains can be bound to specific interfaces thus insuring clients always authenticate and connect to specific interfaces.

In order to provide your clients with a service that is highly available, clustering is recommended. Clusters enable transparency across your clients. If one server becomes unavailable, the client will automatically reconnect to another realm within the cluster. All cluster objects within the realm are replicated among all cluster realms and their state is maintained exactly the same across all realm members. Therefore, whenever a client disconnects from one realm and reconnects to another, they will resume from the same position on the newly connected realm.

When a client provides a list of RNAMES as a comma separated list, if each entry in the list corresponds to realm that is a member of the cluster, then the client will reconnect to the next realm in the cluster list.

3.3.2 Batch Processing

Generally, the I-BiDaaS batch processing module is a software component that is responsible for machine learning and batch analytics within the I-BiDaaS platform. It contains two sub-modules: 1) COMPSs (COMP Superscalar) programming model, and 2) Advanced Machine Learning sub-module (UNSPMF). The former is a task-based programming model that allows for a simplified development of implementations of algorithms that are to be run over a distributed infrastructure (cluster, gird, or cloud). The latter is an actual pool of machine learning and analytics algorithms implemented within the COMPSs programming model and Python, incorporating external codes like Message Passing Interface (MPI), when needed. The algorithmic pool will be evolving as the platform lives, and the implementations will be available in the I-BiDaaS knowledge repository[51] for re-use. Current machine learning algorithm implementations in the pool, developed by UNSPMF in COMPSs (Python version) with BSC support, include K-nearest neighbours, K-means, decision trees, random forests, and sparse regression using the alternating direction method of multipliers (ADMM). In addition, BSC has developed a batch analytics algorithm crafted for the MVP use case scenario, as detailed below.

The Batch Processing components included in the MVP are one application derived from the use case and the technological components that this application required with the corresponding extensions: Hecuba, Apache Cassandra and the TDF tool.

The goal of the application is to find relations between people, given a set of connections to IP addresses. The way to find the relation try to maximize the possibilities of a close relation.

[51] https://github.com/ibidaas/knowledge_repository

At first place, all the IP addresses where a lot of users have been connected are discarded. In this way, connections from public spaces such as libraries, campus, etc. are being discarded, but also aggregation bank tools that connect in the name of a lot of users and is not a clear relation. The maximum number of connections from a specific IP is set as the public space threshold.

There is a relation between two users when they have connected from two different IP addresses in a month, discarding the relations with public IP addresses. In the end, the application must take groups of a maximum of two users. The main data structures are the following:

- IPConnections: the goal of this data structure is to register which users have connected to a given IP address. It is implemented as a dictionary indexed by IP address and with a set of user identifiers as a value.
- DailyPairs: the goal of this data structure is to keep the information about all the connections of each user in a month. It can be implemented as a nested dictionary. The outer dictionary is indexed by month and the inner dictionary is indexed by the user identifier. The value of the inner dictionary is a set of the user connections, connection being a pair composed of the day and the IP address.

The algorithm performs the following steps:

- First of all, the algorithm iterates over all the data in order to create the data structures, only considering those IP addresses that are not from known aggregation bank tools.
- Secondly, using the IPConnections structure, the public IP addresses can be detected. They are the IP addresses with a length of the set of users greater than the public space threshold. The public IP addresses are kept because in the next step it will be used to discard the relations with public IP addresses.
- At last, in order to find the relations, the algorithm uses the DailyPairs structure. For each month, it intersects the set of connections of each user with that of all the others except the set of connections of the users already computed.

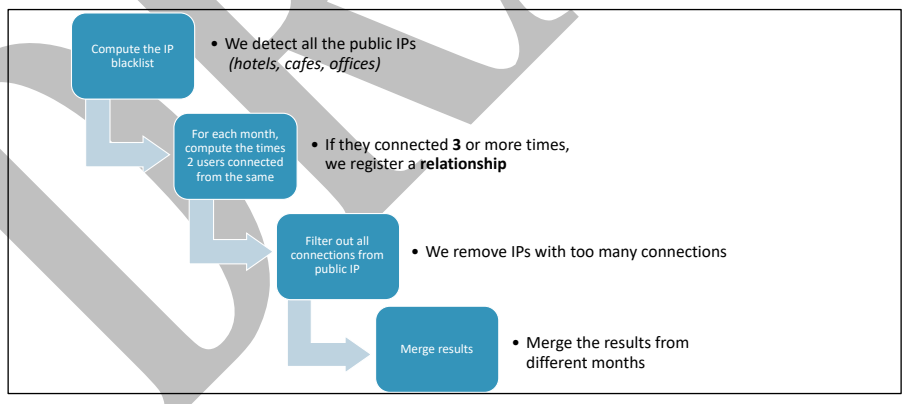

Figure 10. Algorithm to detect relations between users

We have developed three different versions for this application: the sequential version, the parallel version using COMPSs and the parallel version with Cassandra, using COMPSs and Hecuba. All these codes are available in a GitHub repository[52]. For all three versions, the

[52] IBIDaaS MVP repository. https://github.com/cugni/IBiDaaS-MVP/tree/master/batch

input data, describing all the connections, is assumed to be stored in Apache Cassandra and is accessed through the interface implemented by Hecuba. We plan to explore two different choices for TDF to provide data to the Batch layer of the I-BiDaaS platform: one is TDF storing data directly on the database and the other one is TDF communicating with Hecuba through Universal Messaging. For the MVP we are considering only the first option.

The sequential version of the application is just an implementation of the algorithm described above and depicted in Figure 10. In the following sections, we describe the two parallel versions implemented.

3.3.2.1 Parallel version using COMPSs

COMPSs programming model provides an easy and convenient way to parallelize sequential codes. The programmers just need to identify the tasks that can exploit the parallelism of the hardware and mark them using a python decorator. The COMPSs runtime automatically identifies the dependencies between tasks (between input/output parameters of different tasks) and implements transparently to the user the synchronization operations. In this initial version of the code, we have defined several parallel tasks, not only to exploit parallelism but also to benefit from the automatic detection of the dependencies. The main tasks of our algorithm are the following:

- Divide in chunks the input data (randomly), and for each chunk initialize in parallel the data structures with the input data
- Merge the results of the initialization
- Compute the blacklist with the disposable public IP addresses
- Identify the months that appear in the input data
- For each month, compute in parallel the relation between users
- Merge the information about relations detected for each month

Figure 11 represents the code that implements the initialization of the IPConnections and the DailyPair structure together with the corresponding merge phase, while Figure 12 represents the portion of the dependency graph of the COMPSs version of the application for this code fragment.

```
@task(returns=(dict,dict))
def chunk_aggr(partition):
    # A map of maps with format
    #  { 'june': { 'userA': {('127.0.0.1','15th'}]}}
    result = defaultdict(lambda: defaultdict(set))
    # A map of users for ip to ban public ips
    ip_connections = defaultdict(set)
    for ((FK_NUMPERSO, PK_ANYOMESDIA), (IP_TERMINAL, _)) in partition.items():
        if IP_TERMINAL not in KNOWN_AGGR_BANK_TOOLS_IP:
            day_of_the_month = PK_ANYOMESDIA[-2:]
            connection = (day_of_the_month, IP_TERMINAL)
            month = PK_ANYOMESDIA[:-2]
            result[month][FK_NUMPERSO].add(connection)
            ip_connections[IP_TERMINAL].add(FK_NUMPERSO)
    return result, ip_connections

@task(returns=dict)
def get_reduced_IPConnections(*ip_connections):
    return reduce(compute_IPConnections, ip_connections)

def compute_IPConnections(a_ip_connections, b_ip_connections):
    ip_connections = defaultdict(set)
    for ip in set(a_ip_connections.keys() + b_ip_connections.keys()):
        ip_connections[ip] = a_ip_connections[ip].union(b_ip_connections[ip])
    return ip_connections


def main():
(...)
daily_pairs, ip_connections = zip(*map(chunk_aggr), records.split()) #compute local
results

compss_barrier()
```

Figure 11. COMPSs version: Initialization of IPConnection and DailyPairs

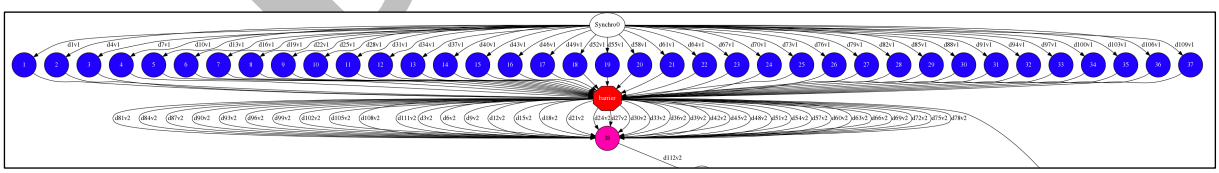

Figure 12. Dependency partial graph

Notice that, as the temporary data is stored in memory, it is necessary to add merge tasks to join the partial results. However, if we use a database to store this temporary data, we can omit the synchronization and the merge task, maximising the parallelism degree. Figure 13 illustrates three different approaches to dealing with this splitting-parallel computing-merging

scheme: the one that we have using with COMPSs programming model, the alternative using Spark/Hadoop engines and, finally, the one that can be obtained using Hecuba as the backend for the data.

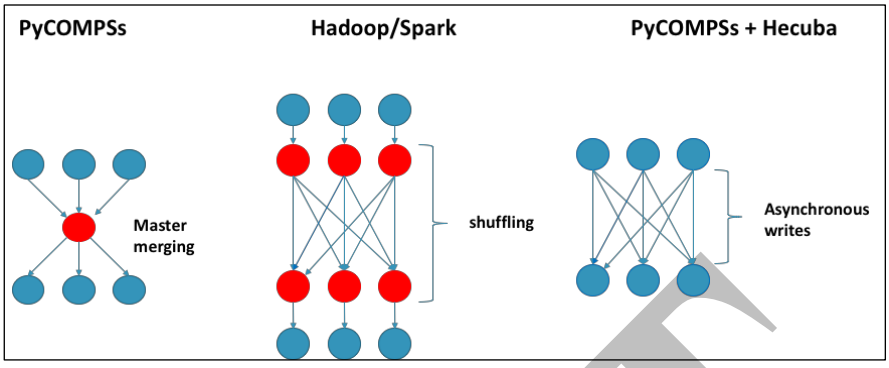

Figure 13. Split-compute-merge solutions

3.3.2.2 Parallel version using COMPSs and Hecuba

The third version of the application is a COMPSs application implemented using Hecuba as the data interface and Cassandra as data backend.

As we have said in section 3.3.2.1, using Cassandra to store the data allows to delegate on the database the management of the global view of the data, avoiding explicit synchronization points and maximizing the parallelism degree. Notice that the interface to insert data in Cassandra is asynchronous with the execution of the application, overlapping this way the data storing with the computation.

To implement this application version, we have extended Hecuba to support new data types: sets and dictionaries of sets.

Hecuba allows users to access the data as regular python objects stored in memory. The only requirement is to define previously the class that supports each data structure, specifying the data types.

Figure 14 shows the definition of the classes that support the two main data structures of our use case.

```
class IPConnections(StorageDict):
'''
@TypeSpec <<ip_terminal:str>, users:set<int>>
'''
class DailyPairs(StorageDict):
'''
@TypeSpec<<month:str,user:int>,connections:set<day:str,ip:str>>
'''
```

Figure 14. Classes definition for the data structures

Both classes are implemented as dictionaries, that Hecuba converts into the suitable tables on the database. To achieve this transparent translation, it is necessary to make them inherit from the Hecuba class StorageDict. With the decorator @TypeSpec, the programmer can specify the type for both the key and the value of each element of the dictionary. Optionally, it is also possible to specify a name for each of these fields.

The keys of dictionaries of class IPConnections are of type string while the values are sets of integers (i.e. for each IP address we store the identifier of all users that have connected to it).

The key of dictionaries of class DailyPairs is a tuple composed of a string and an integer, while the value is a set of tuples, each of them composed of two strings.

Once defined these classes, the programmer can instance some objects and use the regular python operators on those instanced objects. Hecuba offers two options for the object instantiation: volatile objects or persistent objects. All volatile object can become persistent at any point by using the make_persistent method. Instantiation of persistent objects requires as a parameter a string to identify the data in the database.

Figure 15 shows the code of the initialization of the IPConnections and the DailyPairs structures. Notice that comparing this code fragment with the one showed in Figure 11, in this case, the merge is done inside the initialization function, as the object containing the IP connections is global and the synchronization is a responsibility of the database.

```
task()
def chunk_aggr(partition, ip_connections, daily_pairs):
   # A map of maps with format
   #  { 'june': { 'userA': {('127.0.0.1','15th'}]}}
   for ((FK_NUMPERSO, PK_ANYOMESDIA), (IP_TERMINAL, _)) in partition.items():
      if IP_TERMINAL not in KNOWN_AGGR_BANK_TOOLS_IP:
         day_of_the_month = PK_ANYOMESDIA[-2:]
         connection = (day_of_the_month, IP_TERMINAL)
         month = PK_ANYOMESDIA[:-2]
         try:
            daily_pairs[month, FK_NUMPERSO].add(connection)
         except KeyError:
            daily_pairs[month, FK_NUMPERSO] = {connection}
         try:
            ip_connections[IP_TERMINAL].add(FK_NUMPERSO)
         except KeyError:
            ip_connections[IP_TERMINAL] = {FK_NUMPERSO}

def main():
(...)
ip_connections = IPConnections( "test.IPConnections" )
daily_pairs = DailyPairs( "test.DailyPairs" )
map(lambda x: chunk_aggr(x, ip_connections, daily_pairs), records.split())
```

Figure 15. COMPSs + Hecuba version: Initialization code of IPConnections and DailyPairs

3.3.3 Streaming analytics

When a streaming use case is implemented, fabricated data is immediately grabbed by APAMA from UM and analyzed. More details are given in the APAMA related deliverables D4.1 “Real time complex event processing engine – design and approach” and D4.3 “Streaming analytics and predictions”. In fact, the former has the same due date as D5.2 (M12).

3.3.4 Visualization

The visualisation needs of the MVP are based on the Multipurpose Interface and interactive visualisation framework which are reported analytically in D2.3 (I-BiDaaS visualization and monitoring framework, and a multipurpose interface). Building upon the extensible visualisation framework, the MVP includes visualisations of the data that are generated in the context of the MVP use cases as described in the previous paragraph. More specifically, for the first use case, the goal of the visualisations is to display the various clusters of relationships and let the end-users quickly get an overview of the sizes of these groups as well as identify in what kind of relationships a specific transaction might be included. For the second use case, a visualisation of the generated alarms/events deriving from the stream processing component is available to let users know about detected events in real time.

Furthermore, a fraction of the Multipurpose Interface has been adapted to serve the needs of the MVP. As described in D2.3, the interface has been designed with the aim of letting users create their data analysis projects irrespectively of their technical background. This means that the interface will offer generic functionalities with respect to data loading, data manipulation, analytics algorithms, visualisations, etc. In the context of the MVP, higher development effort has been given on the functionalities required by the rest of the components and the MVP use case so as to produce a unified interface that showcases this first integrated version of the I-BiDaaS platform. These functionalities are described in the following paragraphs.

The Batch Processing use case is comprised of the following steps that can be executed by users via the provided interface:

- Define the dataset of transactions (predefined TDF model)
- Start the Batch Processing job
- Get notified when the job is finished
- Visualise results and statistics about the job

The following figures depict the parts of the Multipurpose Interface that offer the elements to perform the above steps as well as the visualisations used to present the results of the analysis to the users:

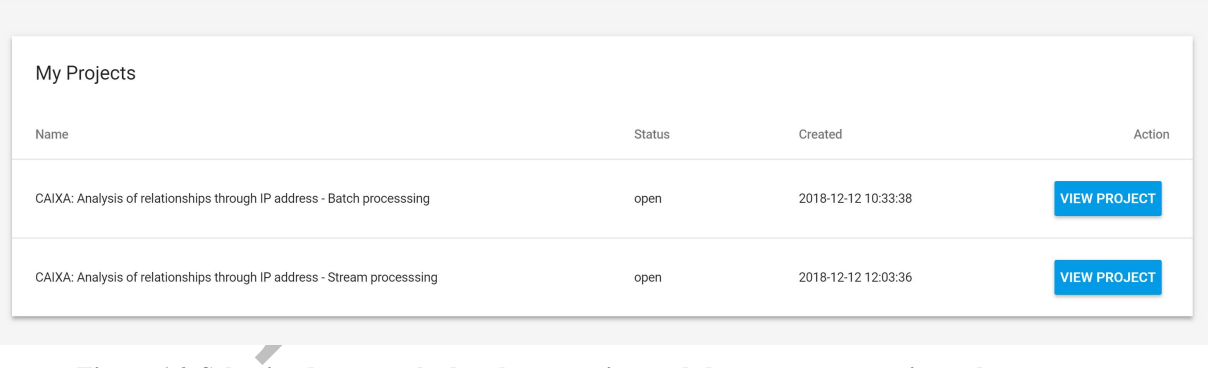

Figure 16. Selection between the batch processing and the stream processing subcase

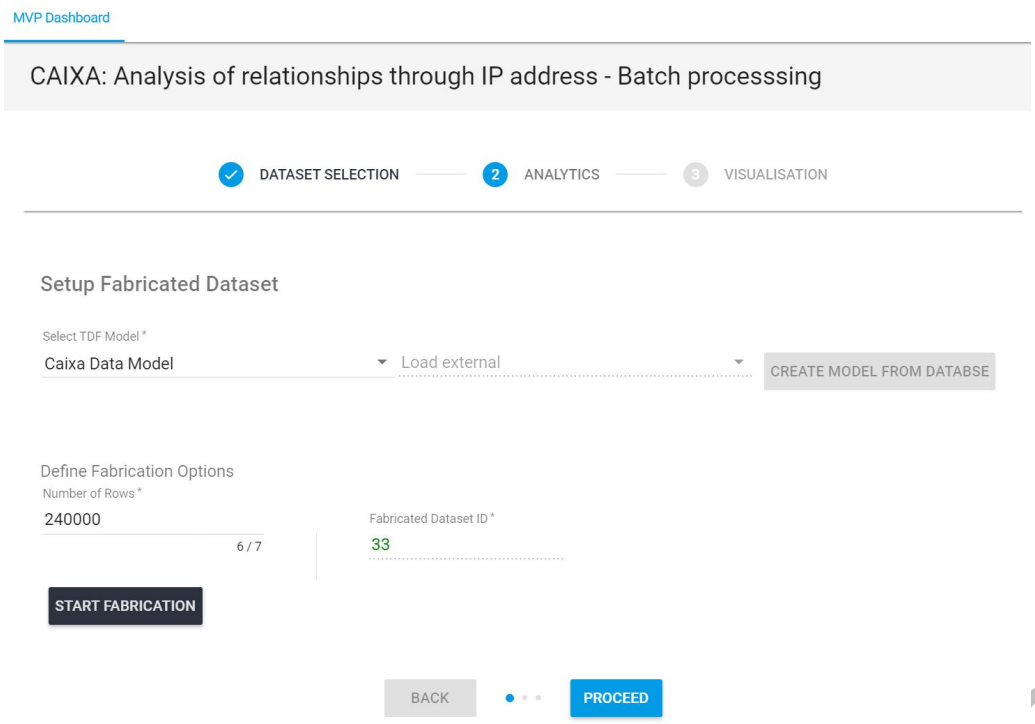

Figure 17. Select batch TDF model

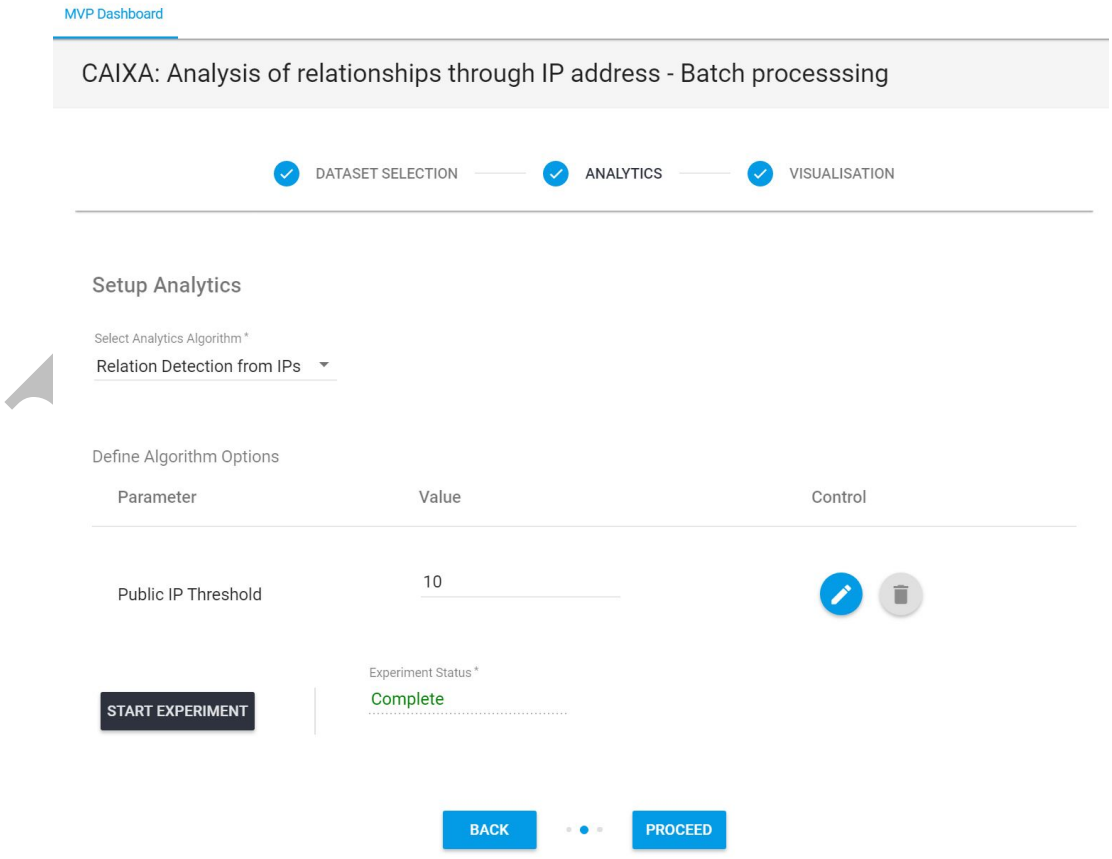

Figure 18. Select algorithm

Figure 19. Visualise relationships

The Streaming use case follows a similar pathway in the interface. Its steps include:

- Start Streaming of transactions
- Visualise streaming processing results
- View historical results – Older alarms caught by the system

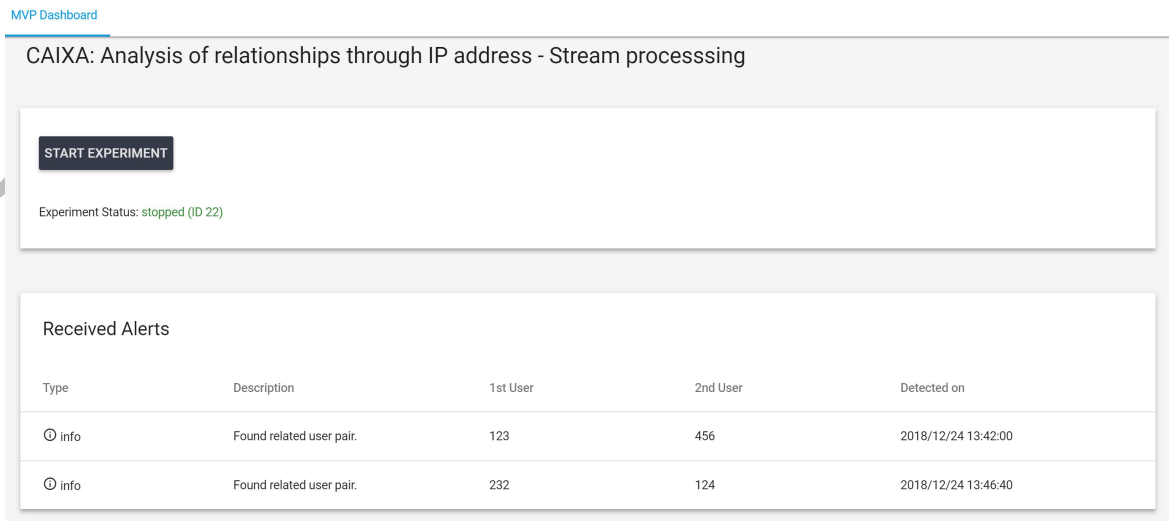

Figure 20. Visualise captured events

The last step of the Streaming use case is about visualising alerts having been captured in the past and stored to the platform. Therefore, the visualisation resembles the one depicted in Figure 20 with the difference being only the source of the alerts. In the real-time case, alerts come from the MQTT service which the interface listens to, whereas in the latter case events come from the internal storage using the relevant REST API call offered by the platform.

3.3.4.1 Technologies

Both sub-modules of the Visualisation framework, namely the Advanced Visualisation Toolkit (AVT) and MashZone make use of AngularJS[53] framework to provide interactive and dynamic web applications that present data visualisations to end users. The multipurpose interface also relies on AngularJS to provide the dashboard functionalities required by the MVP and generally by the entire I-BiDaaS integrated platform.

The communication with the underlying components, as also described in the architecture definition in D1.2, follows a service-oriented approach with heavy use of RESTful APIs that are exposed from all the underlying components. The Interface makes use of these APIs to trigger actions on the backend as well as to get status notifications for running jobs and finally retrieve the results of the data analytics algorithms. It must be noted that other messaging protocols, such as MQTT, have been also used in order to offer real-time information about system operations. For example, in the streaming use case of the MVP, the identified alerts coming from the stream processing tool (APAMA) are picked by the visualisation component by subscribing to the Universal Message Bus through a MQTT listener service.

3.3.5 MVP Orchestration

To speed up the realization of the MVP and to integrate components that were lacking appropriate interfaces for the integration, an orchestrator service was created, especially for the MVP.

The orchestration service is implemented as RESTful web service and has the following functionality:

Table 10. MVP Orchestration

Method	Url	Description
GET	/project/findByStatus	Finds Projects by status
GET	/project/experiments/{id}	Find project experiments by project ID
GET	/project/{id}	Find project by ID
POST	/project/{id}	Updates a project in the store
DELETE	/project/{id}	Deletes a project and all its experiments
GET	/experiment/run/{id}	Run a new experiment for a specific project
GET	/experiment/{id}	Get experiment status by id
PUT	/experiment/stop/{id}	Stop/cancel experiment by id
GET	/experiment/download/{id}	Get the results of a completed experiment

[53] https://angularjs.org/

The definition of the orchestrator API is provided in Appendix 2.

The orchestration service is used by the Visualization Component that acts a user interface for the rest of the I-BiDaaS platform. Future versions of this orchestration service will be incorporated in the Resource Management and Orchestration Component.

4 Conclusions and next steps

The current deliverable presents the first version of the "Big-Data-as-a-Self-Service Test and Integration Report". The work gives important perception towards the release of the envisioned I-BiDaaS solution, ensuring a smooth and effective integration of the separate I-BiDaaS components, taking into consideration their availability, interoperability potential, scalability and performance requirements. Moreover, this document presents the analysis of the MVP based on a CAIXA's related use case. The main objective of this use case is to validate and test the I-BiDaaS architecture and to kick-start the integration between the I-BiDaaS components. The findings obtained from the current study are summarized below:

- Prevision of the preliminary analysis of the testing and integration plan for the I-BiDaaS platform (Chapter 2).
- Representation of the main tools and components developed on this stage of investigation (Chapter 2).
- Definition of the basis for the integration of developed components and provision of the hardware infrastructure used in the frame of the I-BiDaaS project (Chapter 2).
- Description of the MVP use case, provided by CAIXA, by giving the main context of the use case (which is to reveal relations between bank customers for security checks, given a set of their IP address connections to online banking services) (Chapter 3).
- Illustration of the use case dataset selection process, data formatting and data flowing processes (Chapter 3).
- Bringing in the MVP architecture including both batch and streaming processes and demonstrating the integration between the I-BiDaaS components and their distinct technologies (Chapter 3).

Future work, starting from M13, will expand the integration of the I-BiDaaS platform with respect to the MVP version in Figure 9, including the tools and technologies by FORTH, ATOS and UNSPMF, and it will include also CAIXA, CRF and TID's use cases experiments according a revised experimental protocol as lead by UNIMAN. Moreover, as the various use cases that will be implemented in the I-BiDaaS platform become well-defined, a complete list of E2E tests will be prepared and utilized for the validation of I-BiDaaS platform integration.

Appendix 1 – Hardware specifications Template

COMPONENT REQUIREMENTS

Component (Algorithm, Model, Sub system, etc.) Specification Template

Overview			
Component name:	[A short name to be used as an identifier for the component]		
Created by:	[Name of the person/partner who created the component spec]	Document version:	[This document's version]

Software Description
[Provide the description of the software used for this component]

Component requirements	
Platform requirements:	[A list of the platform dependencies for the component]
Software dependencies, libraries required:	[A list of software required for the operation of the component]
Software license:	[Note if the software is opensource, whether it can be installed in a common BIDaaS server, etc]
Hardware requirements:	[A list of hardware requirements for the operation of the component (memory, CPU, disk space…)]

Additional Information	
Notes and Issues:	[List any additional comments about this component, remaining open issues or TBDs (To Be Determined) that must be resolved.

Appendix 2 – Orchestration Service Definition

```
 swagger: "2.0"
info:
  description: "This is the I-BiDaaS MVP Orchestrator Service"
  version: "1.0.0"
  title: "I-BiDaaS Orchestrator"
  contact:
    email: "vchatzi@itml.gr"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
host: "ibidaas-db.itml.gr"
basePath: "/api"
tags:
- name: "project"
  description: "I-BiDaaS projects"
- name: "experiment"
  description: "I-BiDaaS projects have multiple experiments"
schemes:
- "http"
paths:
  /project/findByStatus/{status}:
    get:
      tags:
      - "project"
      summary: "Finds Projects by status"
      operationId: "findProjectssByStatus"
      produces:

      - "application/json"
      parameters:
      - name: "status"
        in: "path"
        description: "Status values that need to be considered for filter"
        required: true
        type: "array"
        items:
          type: "string"
          enum:
```

```
        - "open"
        - "closed"
        default: "open"
    responses:
      200:
        description: "successful operation"
        schema:
          type: "array"
          items:
            $ref: "#/definitions/Project"
      400:
        description: "Invalid status value"
/project/experiments/{id}:
  get:
    tags:
    - "project"
    summary: "Find project experiments by project ID"
    description: "Returns an array of experiments"
    operationId: "getProjectExperimentsById"
    produces:

    - "application/json"
    parameters:
    - name: "id"
      in: "path"
      description: "ID of project to return experiments for"
      required: true
      type: "integer"
      format: "int64"
    responses:
      200:
        description: "successful operation"
        schema:
          type: array
          items:
            $ref: "#/definitions/Project"
      400:
        description: "Invalid ID supplied"
      404:
        description: "Project not found"
/project/{id}:
  get:
    tags:
    - "project"
```

```
    summary: "Find project by ID"
    description: "Returns a single project"
    operationId: "getProjectById"
    produces:

    - "application/json"
    parameters:
    - name: "id"
      in: "path"
      description: "ID of project to return"
      required: true
      type: "integer"
      format: "int64"
    responses:
      200:
        description: "successful operation"
        schema:
          $ref: "#/definitions/Project"
      400:
        description: "Invalid ID supplied"
      404:
        description: "Project not found"
  put:
    tags:
    - "project"
    summary: "updates a project in database"
    description: ""
    operationId: "updateProject"
    produces:
    - "application/json"
    parameters:
    - name: "id"
      in: "path"
      description: "ID of project that needs to be updated"
      required: true
      type: "integer"
      format: "int64"
    - name: project
      in: body
      description: The project to create.
      schema:
        type: object
        required:
          - name
```

```
        properties:
          name:
            type: string
    responses:
      405:
        description: "Invalid input"
  delete:
    tags:
    - "project"
    summary: "Deletes a project"
    description: ""
    operationId: "deleteProject"
    produces:

    - "application/json"
    parameters:
    - name: "id"
      in: "path"
      description: "Project id to delete"
      required: true
      type: "integer"
      format: "int64"
    responses:
      400:
        description: "Invalid ID supplied"
      404:
        description: "Project not found"
/project:
  post:
    tags:
    - "project"
    summary: Add a new project
    parameters:
      - in: body
        name: project
        description: The project to create.
        schema:
          type: object
          required:
            - name
          properties:
            name:
              type: string
    responses:
```

```
      405:
        description: "Invalid input"

/experiment/run/{id}:
  get:
    tags:
    - "experiment"
    summary: "Run a new experiment for a specific project"
    description: "Run a new experiment for a specific project"
    operationId: "runExperimentForProject"
    produces:
    - "application/json"
    parameters:
    - name: "id"
      in: "path"
      description: "ID of project to run experiment for"
      required: true
      type: "integer"
      format: "int64"
    responses:
      200:
        description: "successful operation"
        schema:
          $ref: "#/definitions/Experiment"
      400:
        description: "Invalid ID supplied"
      404:
        description: "Project not found"
/experiment/{id}:
  get:
    tags:
    - "experiment"
    summary: "Get experiment status by id"
    description: "Get experiment status by id"
    operationId: "getExperimentById"
    produces:

    - "application/json"
    parameters:
    - name: "id"
      in: "path"
      description: "ID of project to run experiment for"
      required: true
      type: "integer"
```

```
      format: "int64"
    responses:
      200:
        description: "successful operation"
        schema:
          $ref: "#/definitions/Experiment"
      400:
        description: "Invalid ID supplied"
      404:
        description: "Experiment not found"
/experiment/stop/{id}:
  put:
    tags:
    - "experiment"
    summary: "Stop/cancel experiment by id"
    description: "Stop/cancel experiment status by id"
    operationId: "stopExperimentById"
    produces:

    - "application/json"
    parameters:
    - name: "id"
      in: "path"
      description: "ID of experiment to stop/cancel"
      required: true
      type: "integer"
      format: "int64"
    responses:
      200:
        description: "successful operation"
        schema:
          $ref: "#/definitions/Experiment"
      400:
        description: "Invalid ID supplied"
      404:
        description: "Experiment not found"
/experiment/download/{id}:
  get:
    tags:
    - "experiment"
    summary: "Get the results of a completed experiment"
    description: "Get the results of a completed experiment"
    operationId: "downloadExperimentResults"
    produces:
```

```
    - "application/json"
    parameters:
    - name: "id"
      in: "path"
      description: "ID of experiment to get results for"
      required: true
      type: "integer"
      format: "int64"
    responses:
      200:
        description: "successful operation"
        schema:
          $ref: "#/definitions/ExperimentResult"
      400:
        description: "Invalid ID supplied"
      404:
        description: "Experiment not found"
      405:
        description: "Experiment not completed"


definitions:
  Project:
    type: "object"
    required:
    - "name"
    properties:
      id:
        type: "integer"
        format: "int64"
      name:
        type: "string"
        example: "Caixa use case 1"
      status:
        type: "string"
        description: "project status"
        enum:
        - "open"
        - "closed"
    xml:
      name: "Project"
  Experiment:
    type: "object"
```

```
  required:
  - "name"
  properties:
    id:
      type: "integer"
      format: "int64"
    projectid:
      type: "integer"
      format: "int64"
    date:
      type: "string"
      example: "201812111031"
    status:
      type: "string"
      description: "project status"
      enum:
      - "running"
      - "completed"
      - "stopped"
  xml:
    name: "Experiment"
ExperimentResult:
  type: "object"
  required:
  - "data"
  properties:
    description:
      type: "string"
      example: "A zipped file containing the results of the experiment"
    referall:
      type: "string"
      description: "A url to download results for"
      example: "http://ibidaas-db.itml.gr/results/23234512435.zip"
  xml:
    name: "ExperimentResult"
```