CROSSMINER
Developer-Centric Knowledge Mining from Large Open-Source Software Repositories

Project Number 732223

D8.9 Integrated Platform - Final Version

Version 1.0
18 October 2019
Final

Public Distribution

Unparallel Innovation

Project Partners: **Athens University of Economics & Business, Bitergia, Castalia Solutions, Centrum Wiskunde & Informatica, Eclipse Foundation Europe, Edge Hill University, FrontEndART, OW2, SOFTEAM, The Open Group, University of L'Aquila, University of York, Unparallel Innovation**

PROJECT PARTNER CONTACT INFORMATION

Athens University of Economics & Business Diomidis Spinellis Patision 76 104-34 Athens Greece Tel: +30 210 820 3621 E-mail: dds@aueb.gr	**Bitergia** José Manrique Lopez de la Fuente Calle Navarra 5, 4D 28921 Alcorcón Madrid Spain Tel: +34 6 999 279 58 E-mail: jsmanrique@bitergia.com
Castalia Solutions Boris Baldassari 10 Rue de Penthièvre 75008 Paris France Tel: +33 6 48 03 82 89 E-mail: boris.baldassari@castalia.solutions	**Centrum Wiskunde & Informatica** Jurgen J. Vinju Science Park 123 1098 XG Amsterdam Netherlands Tel: +31 20 592 4102 E-mail: jurgen.vinju@cwi.nl
Eclipse Foundation Europe Philippe Krief Annastrasse 46 64673 Zwingenberg Germany Tel: +33 62 101 0681 E-mail: philippe.krief@eclipse.org	**Edge Hill University** Yannis Korkontzelos St Helens Road Ormskirk L39 4QP United Kingdom Tel: +44 1695 654393 E-mail: yannis.korkontzelos@edgehill.ac.uk
FrontEndART Rudolf Ferenc Zászló u. 3 I./5 H-6722 Szeged Hungary Tel: +36 62 319 372 E-mail: ferenc@frontendart.com	**OW2 Consortium** Cedric Thomas 114 Boulevard Haussmann 75008 Paris France Tel: +33 6 45 81 62 02 E-mail: cedric.thomas@ow2.org
SOFTEAM Alessandra Bagnato 21 Avenue Victor Hugo 75016 Paris France Tel: +33 1 30 12 16 60 E-mail: alessandra.bagnato@softeam.fr	**The Open Group** Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of L'Aquila Davide Di Ruscio Piazza Vincenzo Rivera 1 67100 L'Aquila Italy Tel: +39 0862 433735 E-mail: davide.diruscio@univaq.it	**University of York** Dimitris Kolovos Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325167 E-mail: dimitris.kolovos@york.ac.uk
Unparallel Innovation Bruno Almeida Rua das Lendas Algarvias, Lote 123 8500-794 Portimão Portugal Tel: +351 282 485052 E-mail: bruno.almeida@unparallel.pt	

DOCUMENT CONTROL

Version	Status	Date
0.1	Table of Content	26/08/2019
0.2	Contributions to sections 2 and 3	23/09/2019
0.3	Updates to sections 2	7/10/2019
0.5	Added section1, 4, 5 and 6	11/10/2019
0.8	Small updates	14/10/2019
1.0	Final version after partner reviews	18/10/2019

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This deliverable reports the work carried under Task 8.2, aiming to the integration of the technical components developed in work packages 2-7.

To this end, we have identified all the components that have been integrated and the requirements that have been satisfied for enabling their integration. Moreover, we identified and described extra components that were developed to support the integration and management of the integrated platform: API Gateway, Authentication Server and Administration Dashboard. As CROSSMINER reutilizes components from OSSMETER, in this deliverable is also reported the improvements performed to those components.

In this document, are also presented the methodologies and infrastructures that support the development and testing of CROSSMINER products. Besides, it is also identified and described the recommended and supported approach for deployment of the CROSSMINER platform, which consists in deploying the platform as a distributed application based on containers and orchestrated by the Docker Compose tool. To support future reconfigurations of the platform, it is provided a description of the Docker images and the configurations used.

1 INTRODUCTION

1.1 PURPOSE OF THE DELIVERABLE

This deliverable has as main objective to give an understanding about the status of the CROSSMINER architecture and the development of the CROSSMINER components at Month 33. The main focus is not on how each component was developed but on how the integration of the components was performed.

The deliverable also describes how the CROSSMINER platform can be deployed and the corresponding resources that are needed to this end.

1.2 STRUCTURE OF THE DOCUMENT

The deliverable is organized as follows:

- Section 2 presents an overview of the CROSSMINER architecture and highlights the main changes with respect to the previous OSSMETER platform;
- Section 3 reports the development work, which has been done to integrate the software components produced in the context of the different work packages, as well as, the improvements operated on the OSSMETER components.

  Section 4- presents the recommended Platform Deployment procedure, providing a guide showing how CROSSMINER products can be deployed and configurated;
- Section 5 – presents an overview of the available documentation containing information about how to use the CROSSMINER technologies both from a user and developer point of view.

1.3 RELATIONS WITH OTHERS CROSSMINER DELIVERABLES

This deliverable progresses the work started in *D8.1 – Establishment of the Architectural Guidelines* and improved in *Deliverable D8.2 - Integrated Platform - Interim Version*, by tracking the changes and updating the CROSSMINER architecture. Moreover, this deliverable tracks the development carried up in work packages 2, 3, 4 and 5 for supporting the integration and deployment of the components developed in those workpackages.

2 CROSSMINER IMPLEMENTATION

In this section, we update and discuss the changes operated on the CROSSMINER architecture initially specified in deliverable D8.1 - *Architecture Specification of the Integrated Platform* and updated in Deliverable D8.2 - *Integrated Platform - Interim Version*. Aligned with the architecture, the development of each CROSSMINER component is mapped across the several R&D work packages.

2.1 INTEGRATION APPROACH

As described in D8.1 and D8.2, the CROSSMINER platform uses various measurement tools developed in the context of the OSSMETER FP7 project. In CROSSMINER additional measurement tools have been added by following the same architectural

approach, which was conceived in OSSMETER. These tools were adapted and improved in the context of CROSSMINER activities and were included in the **Metric Platform** module, aggregating all the metric providers and metric execution infrastructures. Figure 1 presents the logical architecture of the CROSSMINER platform, identifying the functional modules that compose the platform. In this figure different components are distinguished: components from OSSMETER planned to be used "as-is"; components developed in the context of CROSSMINER activities, and components originally from OSSMETER that required a refactoring in order to meet the specific needs of CROSSMINER.

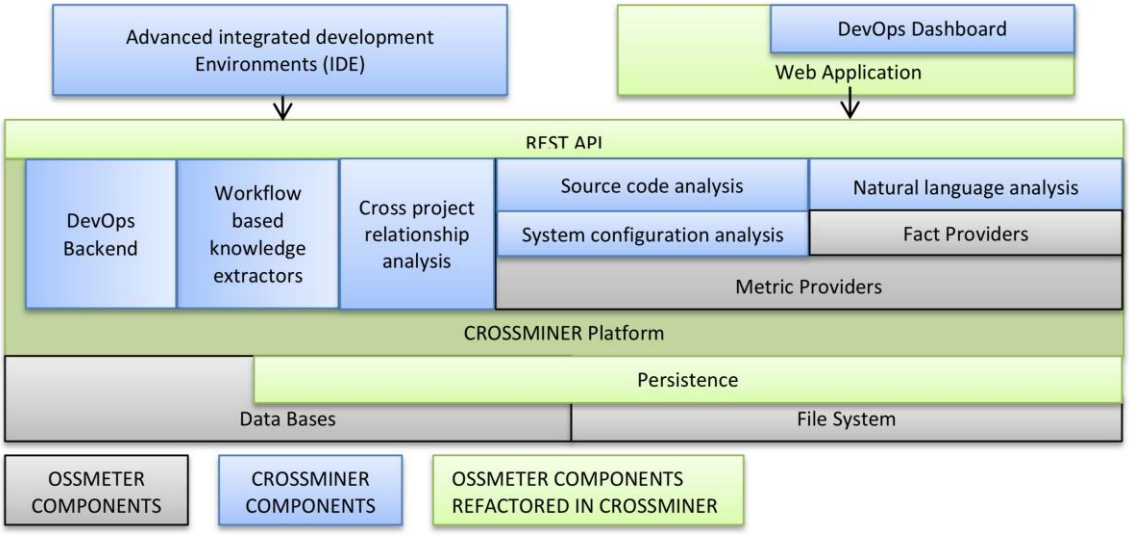

Figure 1 – CROSSMINER logical architecture

During the development of the CROSSMINER tools some extra modifications to fix the OSSMETER components where required to fix some bugs detected during tests. These modifications are listed in Table 1.

Table 1 - Improved Components

Component Name	Short description
Project Delta Manager	This component is in charge of managing the delta creation for retrieving the data from different sources. This delta mechanism in the base for computation of several metric providers. Some bugs on this component compromised the accuracy of some metric providers.
Metric Providers	Some metric providers developed during OSSMETER required the fix of bugs related with data accuracy in specific conditions.
Factoid Metric Provider	This component proposes a series of metrics that, through the presentation of facts, summary data processed in the

Component Name	Short description
	transient metric provider regarding bug trackers and communication channels. Due to modifications to underpinning metric providers some were modified to fix minor bugs and inaccuracies.

2.2 CROSSMINER COMPONENTS

2.2.1 Source code analysis tools

WP2 developed analysis tools to extract and store actionable knowledge from the source code of a collection of open-source projects. WP2 worked also on effectively separating reusable generic language-agnostic and language-dependent analyses from bespoke and context-specific analyses in separate modules, via intermediate models and query language(s). Developments on these tools are reported in D2.1, D2.2, D2.3, D2.4, D2.5, D2.6, D2.7 and D2.8.

Table 2 - Components developed in WP2

Component Name	Short description	Deliverables	Available at
Metadata miner	The **Metadata miner** component is responsible for extracting actionable information from the metadata of software components (OSGi configuration files – MANIFEST.MF, and Maven configuration files – pom.xml). It stores the information in generic models that are later exploited by the **Dependency miner** and **API miner** components.	D2.1 / D2.2 / D2.3 / D2.4 / D2.5	Metric Provider in Metric-platform: https://github.com/crossminer/scava/tree/master/metric-platform
Dependency miner	The **Dependency miner** component exploits the information extracted by the **Metadata miner** component to detect bad practices in the use of dependency-management frameworks. It also includes a set of metrics that exploit the information stored in the generic models and produces data regarding the number, nature, and management of dependencies in software projects which are later displayed in bespoke dashboards.	D2.1 / D2.2 / D2.3 / D2.4 / D2.5	

API miner	The first sub-component of the **API miner,** FOCUS, is a context-aware collaborative-filtering recommendation system that exploits cross-relationships between software projects to suggest API method calls and API usage patterns that assist developers in learning and using APIs; FOCUS is integrated with the Knowledge Base and CROSSMINER IDE to provide developers with recommendations tailored to the particular context they are faced with. The second sub-component of the **API miner**, Maracas, is a source code and bytecode analysis framework that automatically analyses changes in the APIs of software projects to assist developers in the migration of their code when the APIs they rely on evolve; Maracas is integrated with the CROSSMINER platform and dashboards as a set of metrics allowing to understand how OSS projects evolve.	D2.6 / D2.7 / D2.8	**FOCUS** - Integrated into Knowledge-Base: https://github.com/crossminer/scava/tree/master/knowledge-base **Maracas** - Platform Extension in Metric-Platform : https://github.com/crossminer/scava/tree/master/metric-platform/platform-extensions/org.eclipse.scava.maracas

2.2.2 Natural language analysis tools

Work package 3 consists in the development of several components that are used in CROSSMINER to analyse English texts such as bug issues, forums threads, commit messages or software documentation. The components developed in WP3 are presented in Table 3. Each component is composed of several tools which perform key tasks required for the processing and analysis of Natural Language data. The data is retrieved from a varied number of sources, like GitHub, Eclipse Forums, Stackoverflow, HackerNews among others. The analysis proposed by Work Package 3 is based on Natural Language Processing and Text Mining techniques. The outcome of the analysis is used to generate quality metrics and to enrich indexes that can be exploited through queries, dashboards or recommenders. More details regarding these tools can be found throughout the deliverables D3.1, D3.2, D3.3, D3.4, and D3.5.

Table 3 - Components developed in WP3

Component Name	Short description	Deliverables	Available at
Platform Channel Manager	This component is used to manage the readers related to communication channels such as NNTP Newsgroups, IRC, Eclipse Forums.	D3.2, D3.4, D3.5	Platform Extension in Metric-platform: https://github.com

Platform Issue Tracker Manager	This component is in charge of handling the readers that connect to bug tracking systems. Examples of bug trackers are: GitHub, GitLab, JIRA, Bitbucket.	D3.2, D3.4	/crossminer/scava/tree/master/metric-platform/platform-extensions
Platform Software Documentation Resources Manager	This component handles the readers that are in charge of retrieving the information regarding software documentation.	D3.5	
Platform Social Media Manager	It is in charge of managing the readers related to social media such as HackerNews.	D3.2, D3.4	
Platform Question-Answer Manager	This component manages the readers related to question-answer websites, such as those related to StackExchange websites.	D3.2, D3.4	
Natural Language Processing	This component comprises all the tools created for analysing text in natural language. The tools developed include: Topic clustering, Emotion Classifier, Tokeniser, Lemmatiser, Documentation Classifier.	D3.1, D3.3. D3.4, D3.5	Metric Provider in Metric-platform: https://github.com/crossminer/scava/tree/master/metric-platform
Transient Metric Provider	This component, shared with WP2 and WP4, comprises multiple metrics for processing the data retrieved by the readers. In the case of WP3, this component is in charge of calling the natural language tools and process the data in a way that other metrics can exploit them. As well, we provide some metrics related data directly related to the bug trackers and communication channels, i.e. not processed by natural language tools, such as number of users, number of threads in a forums or statistics regarding days of bug trackers activity.	D3.4, D3.5	
Historic Metric Provider	Similar to the Transient Metric Provider, this component includes several metrics created by WP2, WP3 and WP4. In the case of WP3, these metrics are used to create a historic background of how the data evolves through the time.	D3.4, D3.5	

Indexes	This component is responsible for managing the indexing of the data in CROSSMINER.	D3.2, D3.4, D3.5	
Suggestion Recommender	This component, done in collaboration with WP6, provides snippets and discussions regarding the code being created by developers. The data for creating this recommender comes from the data indexed by the **Indexes** component.	D3.5	Integrated into Knowledge-Base: https://github.com/crossminer/scava/tree/master/knowledge-base

2.2.3 System configuration analysis tools

WP4 developed source code analysis tools to gather and analyse system configuration artefacts and data in order to provide quality assessment about the system configuration code. The quality aspects that will emanate from the analysis will lead to actionable insights to improve the quality of a software system holistically. The outcomes of the analysis tools will be the base of the DevOps DashBoard, which will present in a comprehensive manner the quality metrics and issues of an OSS project, along with the configuration dependencies that it has with other projects/libraries. The progress on the development of the System Configuration Code Analyser can be followed in D4.1 and D4.2 and D4.4 and of the DevOps Dashboard in D4.3.

Table 4 - Components developed in WP4

Component Name	Short description	Deliverables	Available at
System Configuration Code Analyser	Previous called System Configuration Code Smell Miner, is a component used to analyse Puppet and Docker based projects and to detect smells and quality issues that can be translated into metrics to be integrated in CROSSMINER metric platform. Also, it is used to extract the dependencies (at configuration level) of each project.	D4.1, D4.2, D4.4	Metric Provider in Metric-platform: https://github.com/crossminer/scava/tree/master/metric-platform
DevOps Dashboard	It presents the outcomes of the **System Configuration Code Analyser** in easily comprehensible and conceivable visualizations that can lead the user to actionable insights about the quality of a project.	D4.3	Integrated into web-dashboards: https://github.com/crossminer/scava/tree/master/web-dashboards

2.2.4 Workflow-based knowledge extractors

The development of bespoke analysis and knowledge extraction tools is simplified by the contribution of a framework that shields engineers from technological issues (e.g. API rate limits, network failures, caching) and allows them to concentrate on the core analysis tasks instead.

Crossflow is a general-purpose, language-agnostic distributed workflow execution platform which architecture is composed of the following components:

- The Crossflow *metamodel* (latest version depicted in D5.6) is used to create Crossflow *models*, i.e. created by the use a dedicated graphical editor (reported in D5.5), that capture the structure of workflows and are employed for the generation of source code.

- The Crossflow *code generator* reads Crossflow metamodel-conforming models and produces source code that may subsequently be compiled or interpreted for execution. Currently, Crossflow supports the languages Java, Python, and R.

- The Crossflow *runtime* represents the execution engine for workflow models and has been implemented in Java and Python. The Crossflow runtime for R employs a Java-based interpreter for R scripts.

- Additionally, the *packaging*, *upload*, and *orchestration* of Crossflow workflow implementations is supported by both the Crossflow API and graphical UI.

Table 5 - Components developed in WP5

Component Name	Short description	Deliverables	Available at
Workflow language	Domain-specific language that is referred to as Crossflow and enables the construction of workflows that are semantically composed of *tasks* and *streams* that act as consumers and producers of messages and channels for passing messages among tasks, respectively. Moreover, a workflow may have *sources* and *sinks* representing specialized tasks that provide inputs and outputs to workflows, respectively.	D5.1, D5.2	Workflow tool: https://github.com/crossminer/scava/tree/master/crossflow
Workflow development tools	Eclipse-based graphical and textual editors that both support the creation of workflows, i.e. conforming to the Crossflow language captured by the Crossflow metamodel, as well as UI components for triggering code generation and workflow packaging, uploading, and execution.	D5.2, D5.3, D5.5	

Workflow code generators	Crossflow code generators creating orchestration code that joins individual workflow tasks for parallelized and distributed execution; base classes for workflow tasks; stubs for implementation classes, i.e. extending base classes, that may be supplemented with dedicated execution logic by developers; and packaged code archives that may be executed on a number of distinct physical machines either programmatically or by the employment of the Crossflow web UI.	D5.5	
Workflow execution engine	Engines for distributed and parallel execution of workflows; currently supporting workflow implementations in Java, Python, and R.	D5.4, D5.6	
Distribution	Distribution of the Crossflow execution engine and the web UI as well as Apache ActiveMQ and Tomcat in form of a Docker image for seamless deployment.	D5.5	
Workflow web UI	Web-based interface for the deployment (i.e. by offering the capability to upload packaged workflows), execution (i.e. by instantiating the workflow execution engine) and monitoring of workflow models (i.e. by visualizing the state of workflow executions during runtime).	D5.5	

2.2.5 Cross-project relationship analysis tools

By exploiting cutting-edge information retrieval techniques, Work-Package 6 has built recommender systems for mining software repositories. Tools and techniques have been conceived to assist software developers in implementing their projects by providing real-time recommendations, which are relevant for the developer context. The recommendation engines are fed with metadata curated from different OSS forges and communication channels. Based on the CROSSMINER mining tools, developers are able to select open source software and get real-time recommendations, which are summarized as follows:

- find a set of similar OSS projects to the system being developed, with respect to different criteria, e.g., external dependencies, or API usage;

- recommend components that similar projects have included, for instance, a list of external libraries;

- recommend code snippets that show how an API is used in practice. These snippets provide developers with a deeper insight into the usage of the APIs being included;

- suggest additional sources of information, e.g., technical documents, tutorials, communication channels, etc., that are relevant to the code being developed, for instance by mining external experiences from StackOveflow;

- identify API changes and their consequences: changes of libraries will have a certain effect on the depending projects. It is necessary to notify developers and recommend amendments to preserve program.

Table 6 - Components developed in WP6

Component Name	Short description	Deliverables	Available at
Recommender	It produces recommendations in response to user requests. This component is extensible in order to add the management of specific types of artifacts. Example of recommendations are code examples that can be examined to solve the particular problem at hand, documentation or StackOverflow posts that are relevant for the current development context, third-party libraries, API function calls, etc.	D6.3, D6.5	Integrated into Knowledgebase: https://github.com/crossminer/scava/tree/master/knowledge-base
ClusterCalculator	It calculates clusters of analysed artifacts. To this end, similarity functions are used as implemented by the SimilarityCalculator component, which oversees managing the execution of both atomic and composed similarity calculations.	D6.2, D6.4, D6.3, D6.5	
SimilarityCalculator	It implements similarity functions, which underpin the definition and the development of the different kinds of recommendations provided by the KB.	D6.2, D6.4	
KnowledgeBaseScheduler	It triggers the calculation of similarities and clusters thus the execution of the available similarity functions.	D6.5	
KnowledgeBase	It is responsible for setting up the other components, as well as for executing them;	D6.1, D6.3, D6.5	https://github.com/crossminer/scava/tree/master/knowledge-base

2.2.6 Advanced integrated development environments

The advanced integrated development environments consist of two components: Eclipse IDE and the Web-based Dashboard. They are summarized below.

2.2.7 Eclipse IDE

Advanced integrated development environments (WP7): Development of extensions for the Eclipse IDE that allow developers to use the CROSSMINER knowledge-base and analysis tools directly from the development environment. The IDE extensions also include features for monitoring the developer activity while they work on a given OSS project. Thus, the IDE issues alerts or recommendations, and collects user feedback which helps developers to improve their productivity. Depending on the context, recommendations can include suggested code snippets, patterns, fixes, and blogs and QA posts to coding issues, suggestions to use alternative APIs or components. The WEB-based Dashboard component provides high-level, global overview of the projects showing aggregated, evolutionary and statistical data. The GitHub integration component enables GitHub users to easily include their projects in a CROSMINER analysis.

Development on web-based dashboards can be checked in D7.1 (requirements) and D7.9 (final version), and details about the Eclipse-based CROSSMINER IDE can checked in D7.1 (requirements), D7.6, D7.7, and D7.10 (final versions). The GitHub integration can be checked in D7.8.

2.2.8 Web-based Dashboard

The purpose and scope of the Web-based Dashboard is to provide a platform to create dynamic dashboards able to show the data collected in CROSSMINER. The main activities of the Web-based Dashboard can be grouped in 4 categories: collection, enrichment, consumption and automation, which are summarized below. Details about the Web-based dashboards can be checked in D7.1 and D7.9.

- **Collection.** CROSSMINER is considered as an external data source for GrimoireLab, thus a *Perceval backend* able to fetch projects and recommendations data from the CROSSMINER API has been built. The project data consists of the list of projects available in CROSSMINER plus their corresponding metrics, while the recommendations are a list of projects similar to the ones analysed. The extraction of projects, metrics and recommendations is achieved by using different categories, which allow to drive the execution of Perceval.

- **Enrichment.** The CROSSMINER data fetched by the Perceval backend is processed by *Scava2es*. First, the data is parsed depending on its category, then it is converted to a common format to easy manipulation and visualization operations, finally the data is stored to the ElasticSearch database.

- **Consumption.** The enriched data stored in ElasticSearch, can be consumed using Kibana[1] dashboards, which allow to focus on different facets of the

[1] Kibana is a tool, which provides visualization capabilities (e.g., scatter plots and pie charts) on top of the content indexed on the Elasticsearch database.

projects such as users, sentiment and emotions, source code insights (dependencies and code quality), configurations, quality models and top topics.

- **Automation.** A set of Shell scripts, executed every 5 minutes, is in charge of triggering the collection and enrichment components plus uploading the Web dashboards to Kibana. It allows to automatically reflect the CROSSMINER data to the Web dashboards with a reasonable delay.

Table 7 - Components developed in WP7

Component Name	Short description	Deliverables	Available at
Developer Activity Monitoring	Component used to record the developers' activity during the development of some software. It is integrated in the CROSSMINER Eclipse IDE plug-in that transparently records configurable developer activity data and sends computed metrics to the CROSSMINER server for further processing.	D7.4, D7.7	Integrated into Eclipse-based IDE: https://github.com/crossminer/scava/tree/master/eclipse-based-ide
IDE Integration Services	This component is used by the CROSSMINER Eclipse IDE plug-in to communicate with the CROSMINER knowledge-base server using the CROSSMINER API.	D7.5, D7.6	
Eclipse-based CROSSMINER IDE	This component is an extension of the Eclipse IDE, offering several features of the CROSSMINER platform to the developers directly from the Eclipse IDE. These include code recommendations, library suggestions, and Q&A posts.	D7.2, D7.10	https://github.com/crossminer/scava/tree/master/eclipse-based-ide
GitHub Integration	This component allows GitHub project owners to have their projects automatically analysed by a CROSSMINER platform server.	D7.8	https://github.com/crossminer/scava/tree/master/github-integration
Web-based Dashboard	This component provides a high-level view of the projects analysed by the CROSSMINER platform. These include summaries, evolutionary data, and statistical analysis.	D7.3, D7.9	https://github.com/crossminer/scava/tree/master/web-dashboards
Perceval backend	It fetches the data from the CROSS-MINER API. Such a data includes project information, metrics available	D7.9	Integrated into Web-based Dash-board:

	via the visualization and raw end-points, factoids and recommendations		https://github.com/crossminer/scava/tree/master/web-dashboards
Scava2es	It processes the data collected by the Perceval backend, coverts it to a standard format, which simplifies inspection and visualization operations, and stores it to the Elasticsearch database.	D7.9	
Kibiter dashboards	They allow to visualize and inspect the data stored in Elasticsearch by leveraging on the Kibiter component.	D7.9	Integrated into Web-based Dashboard: https://github.com/crossminer/scava/tree/master/web-dashboards
Automation script	It is in charge of triggering the collection and enrichment components plus uploading the Web dashboards to Kibana. It allows to automatically reflect the CROSSMINER data to the Web dashboards with a reasonable delay.	D7.9	In Scava-Deployment to be distributes as Docker image: https://github.com/crossminer/scava-deployment/tree/master/dashboard

2.3 ARCHITECTURE UPDATE

During the development of the CROSSMINER platform, some changes have been operated with the respect to the OSSMETER platform as discussed in the following.

In D8.2, we modified how the **Social Media Manager** and **Question-Answer Manager** would interact with CROSSMINER components. Specifically, we expected to link them to a **Project-Related Data Manager**, which in turn would be related to the **Project Analyser**. In this deliverable, we change again this portion of the architecture due to three reasons:

- The **Social Media Manager** and **Question-Answer Manager** are not related to any particular project. Thus, these cannot be linked, even through another manager, to the project analyser.

- The pace in which data are generated and updated in the sources related to the corresponding managers is different to the development pace in projects. In other words, a delta approach is not adequate for retrieving information from social media or question-answer sources in CROSSMINER.

- A successful collaboration between WP3 and WP5 allowed us creating workflows, which can be used to process and index the information from the **Social Media Manager** and **Question-Answer Manager**.

In consequence, the **Social Media Manager** and **Question-Answer Manager** are connected directly to the **Workflow API** rather than to the **Project Analyser** through, the now non-existent, **Project-Related Data Manager**. This has been described in detail in D3.4.

Another change in the architecture is related to the **Indexes** component. The objective of this component is to store raw and processed data analysed by the platform, such as issue tracker comments, emails, and forum posts. Originally, these would be stored in **Data Storage**. However, as it was explained in deliverables D3.2 and D3.4, Elasticsearch is the library used by CROSSMINER to index the data, and it is also used by WP7 for supporting the development and management of Web-based dashboards. Thus, the component **Indexes** has been unlinked from the **Data Storage** component and linked to the **Elasticsearch Data Storage**. This change allowed us not only to avoid a duplication in the tools and libraries used in CROSSMINER, but also to make available the indexes in the dashboards, either through user queries or interfaces created by WP7.

Related to the previous change, there is a modification in the components linked to the Knowledge Base. Originally, the recommenders presented in D3.5, were expected to be connected uniquely to the **Data Storage** as **Indexes** were linked to this component. However, since the change in **Indexes**, now the **Knowledge Base** makes use of the **Elasticsearch Data Storage** too. This change allows us having a recommender, represented in the **Suggestion Recommender** component and explained in D3.5, that makes use of the data indexed by the platform.

Furthermore, we have removed the **Statistics Recorder** component, which originally was going to be used for a relevance feedback process regarding the **Suggestion Generator** component. The main reason for removing this component is the different and evolving data that is used inside the recommender. Specifically, the data used by the Suggestion Generator (presented in D3.5) comes from the indexes, which in turn comes from diverse readers (e.g. GitHub Bug Tracker, Eclipse Forums, StackOverflow, SYMPA mailing lists). This data is changing constantly and might not be uniform, i.e. some data might be richer depending on the metrics selected by the user. This makes harder to create a relevance feedback process that takes into account the diversity of the data to manage. Besides, we expect that the method utilised for the recommender will fulfil users' expectations and they will not need to provide any kind of feedback for retrieving in the future the expected suggestions.

Two new components were added to the architecture: **API Gateway** and **Authentication Services.** The first was designed to provide the different CROSSMINER clients with one entry point that aggregates all the services provided by the other CROSSMINER components. The second component provides a mechanism to authenticate API clients and to define different authorization levels for different clients, allowing the CROSSMINER administration, when required, to protect specific resources. These components are described in more detail in Section 3 of this deliverable.

Figure 2 shows the current architecture of the CROSSMINER platform, which is an updated version of that presented in deliverable D8.3. In this figure the changes to the Social Media Manager and Question-Answer Manager connections are represented, as well as API Gateway and Authentication Services described before. A larger version of the architecture is provided in Appendix A: CROSSMINER Architecture.

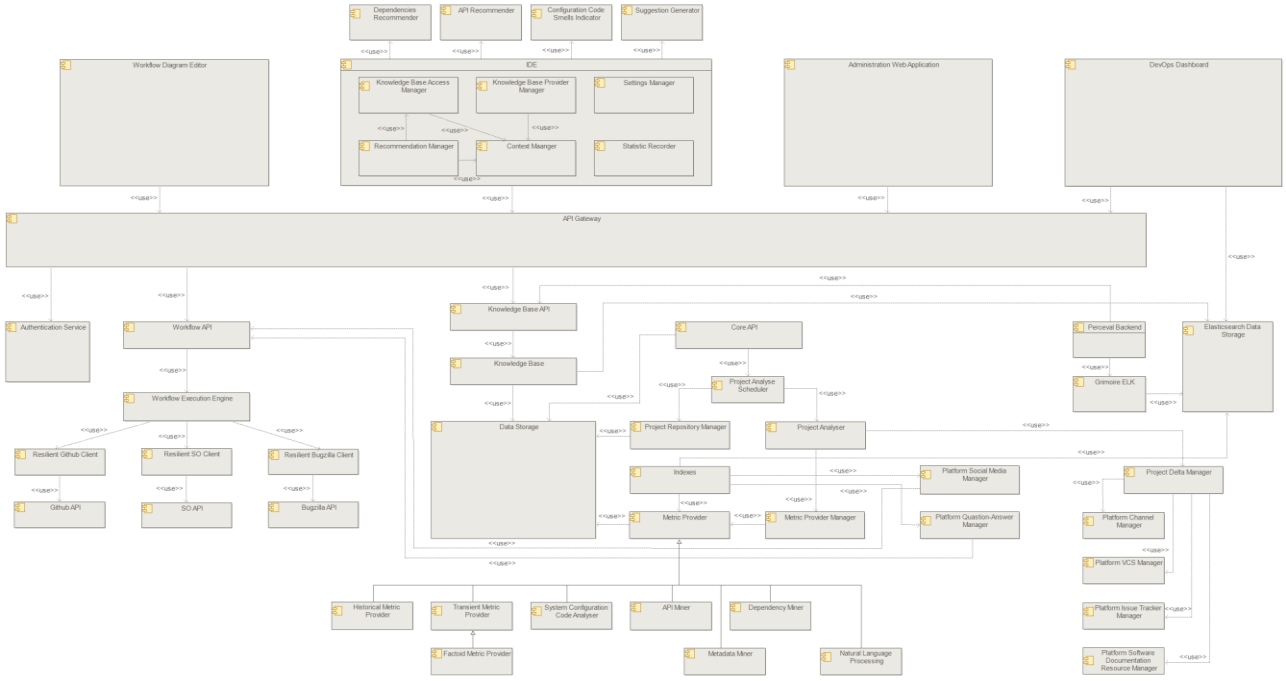

Figure 2 – CROSSMINER Component Architecture

2.4 ADDITIONAL IMPLEMENTATION TO SUPPORT USE CASES: RASCALARDUINO

RascalArduino is a new component developed in the context of **WP2 – Source Code Mining** by **CWI** to address specific requirements of **Use Case 1 – Arduino-based IoT devices** from **UNPARALLEL** (cf. D8.3). This use case encompasses the analysis of C/C++/Arduino code, which goes beyond the current capabilities of the CROSSMINER platform. RascalArduino[2] is a software component able to parse C/C++ source files (including Arduino) and to build processable Abstract Syntax Trees (ASTs) and M3 models. RascalArduino relies on CLAIR[3] (C/C++ Language Analysis In Rascal), a new Rascal component that exploits the Eclipse CDT front-end to parse source code in an efficient way and translate back the results as Rascal-processable ASTs and M3 models for further processing.

A number of metrics related to source code, dependencies, memory analysis, and clone detection have been defined on top of this architecture to support the use case. To ease the integration of RascalArduino with the internal tools developed by UNPARALLEL, RascalArduino stands outside the CROSSMINER platform as a standalone component that can be easily integrated with the IoT-Catalogue of UNPARALLEL through a dedicated REST API.

[2] https://github.com/crossminer/arduino-analysis
[3] https://github.com/cwi-swat/clair

3 INTEGRATION FACILITATORS

The CROSSMINER platform is a complex application that assembles several components, collaborating to provide an accurate analysis of open-source software. In order to handle this complexity, we worked to group these components into coherent units while trying to define how these groups would communicate with each other.

The integration task focuses on identifying these macro components, delineating their perimeters and administering interactions between them, as defined in Figure 3. We have identified six macro components that support the services provided by the CROSSMINER platform and the clients composed of applications that consume these services.

The additional Administration component has been integrated into the architecture to bring together the transverse management functions of the platform (configuration, user management, authentication, etc.). This component became necessary after the removal of the web application component of the OSSMETER platform used both for administrative and project analysis purposes. In CROSSMINER, dedicated and advanced DevOps dashboard are provided to support project analysis tasks.

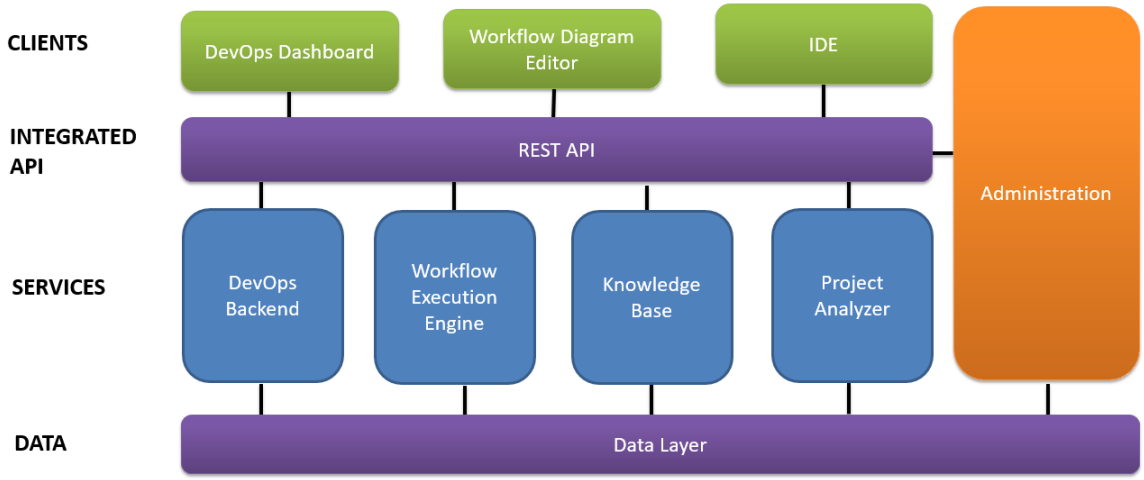

Figure 3 – CROSSMINER macro-components architecture

In order to manage the integration of these components, two main communication channels were identified: a REST API, which enables the communication between services and clients and inter-service communications; and the data layer, which groups the common data to all the platform services.

3.1 INTEGRATED API

3.1.1 Concept

The key concept of our integration approach is to provide a unified REST API that will allow various clients like the Eclipse IDE or the DevOps Dashboard and Administration WebApp to consume services provided by the different platform backend components.

In our integration approach, we have made the choice of not imposing a common technology for the implementation of the different services that compose the platform and did not make any assumptions about where and how CROSSMINER components

will be deployed. Our only requirement is that the components have to provide a REST API to expose their services and that all communication between services and clients or between services itself must be implemented through this API.

In order to integrate all the different components developed in the different work packages we have conceived the API Gateway by following a pattern, which is common in the microservices ecosystem.

The API Gateway is a single entry-point (and point of control) for frontend clients, which could be browser-based or mobile clients, as represented in Figure 4. It acts as a reverse web-proxy that redirects client's requests to services provided by the platform components. With this approach the client only has to know the URL of API Server to access to all services provided by the platform. At the same time, the REST API of the components can be refactored with no changes to the API provided by the API Gateway.

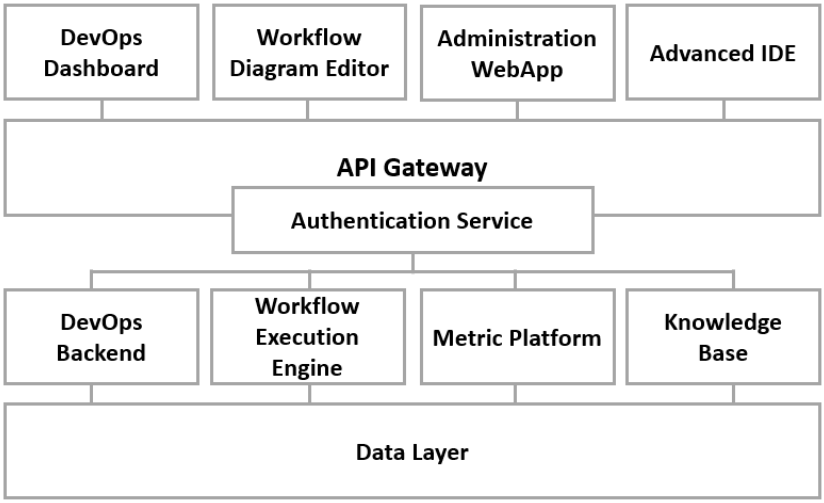

Figure 4 – Interaction between the API Gateway, CROSSMINER components and the API clients

The API Gateway has been implemented using the Netflix Zuul Reverse Proxy, an edge service component of Spring framework dedicated to microservices applications.

Finally, the API gateway is also in charge of managing the access load balancing and the authentication of CROSSMINER's clients through the authentication service as detailed in the next section.

The API Gateway component is available in CROSSMINER GitHub repository (https://github.com/crossminer/scava/tree/master/api-gateway).

3.1.2 Authentication Service via the API Gateway

Confronted with the need to secure access to services provided by platform components, we set up an authentication system at the level of the API Gateway. This approach allows us to secure a heterogeneous REST service bundle in a centralized way.

The CROSSMIER API Gateway is secured using JSON Web Tokens (JWT) mechanism, an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. The

authentication itself is managed by a sub-component of the Administration Application, which centralise the Rights Management for the whole platform. As for the authentication service, it is accessed through the API Gateway.

As exemplified in Figure 5, when the user successfully logs in using their credentials via the authentication server, a JSON Web Token will be returned and must be saved locally, instead of the traditional approach of creating a session in the server and returning a cookie. When the client requests an access to a protected service, the server's protected routes will check for a valid JWT in the Authorization header of the request, and if it is present, the user will be allowed to access the protected resources.

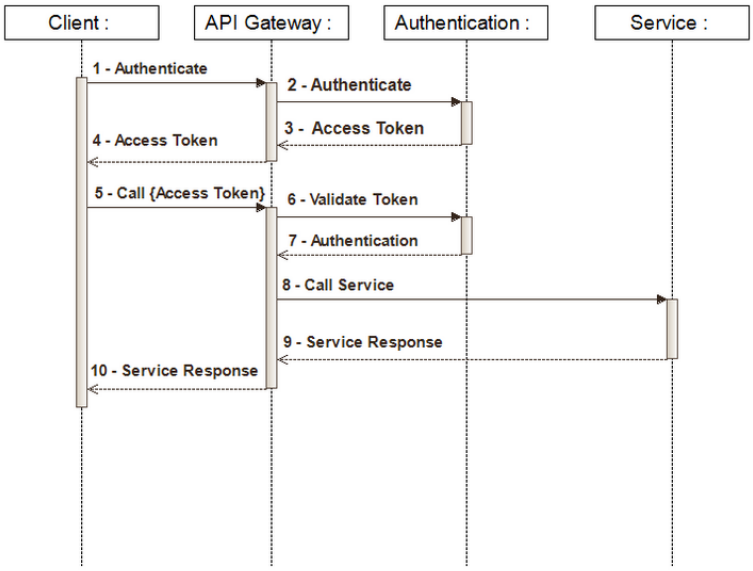

Figure 5 – Authentication protocol and access to protect services

3.2 ADMINISTRATION WEB APPLICATION

3.2.1 Functionalities

As part of the new CROSSMINER platform architecture, we decided to replace the web application from the OSSMETER project by new components. This web application assumed two roles: the visualization of analyses results of the open source projects by platform services and some administration services.

In our new architecture, the visualisation of projects analysis results is managed by the new DevOps Dashboard. Therefore, we had to integrate a new component containing the administrative functions of the deleted application: a new administration-oriented web application which is in charge of the authentication and management of users, configuration of open source project analysis and platform monitoring.

3.2.1.1 User Management and Authentication

The CROSSMINER Administration application implements user management and authentication service for the entire CROSSMINER platform. The platform is organized around four types of users:

- **The Platform User** is a basic user of the platform that can check out the generic content on the CROSSMINER platform including the list of the registered projects details.
- **The Analysis Manager** is in charge of the registration of projects to be analysed by the platform and of the configuration of specific project dashboard using the DevOps Dashboard Component.
- **The Platform Administrator** manages user rights and monitor all the analysis services provided by the platform.

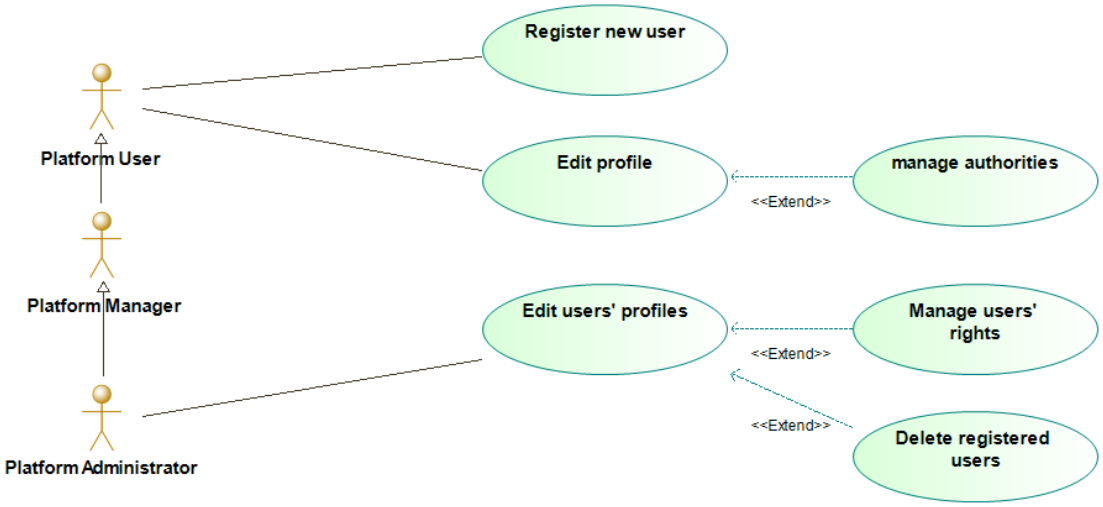

Figure 6 - CROSSMINER User Management Services

AWEB1: Register new user	**Primary Actor (s):** Platform User - Platform Manager - Platform Administrator
Description and Conditions	Allow to register new user on the platform. The registration process includes an e-mail checking process to finalize the registration.

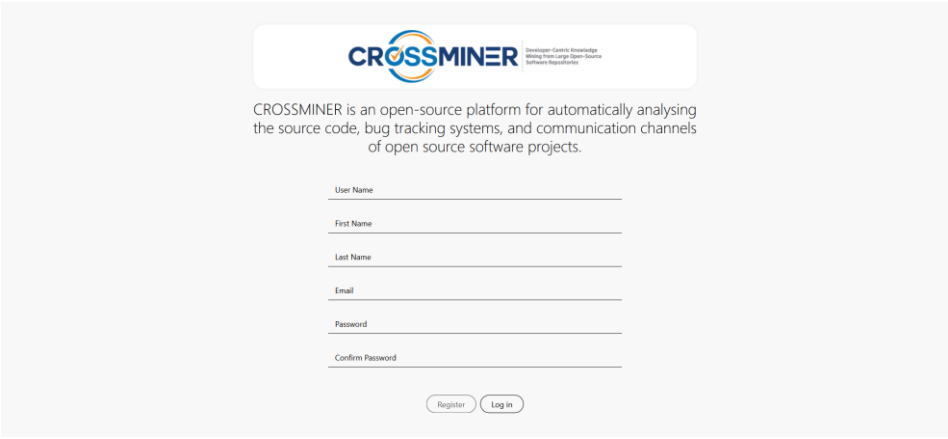

Figure 7 – User Registration

AWEB2: Edit profile	**Primary Actor (s):** Platform User - Platform Manager - Platform Administrator
Description and Conditions	Allow the current user to edit his account settings.

AWEB3: Manage authorities	**Primary Actor (s):** Platform User - Platform Manager - Platform Administrator
Description and Conditions	A registered user can generate some token-authorities to specify different authorities to monitor the developer's activities on the Eclipse IDE plugin side.

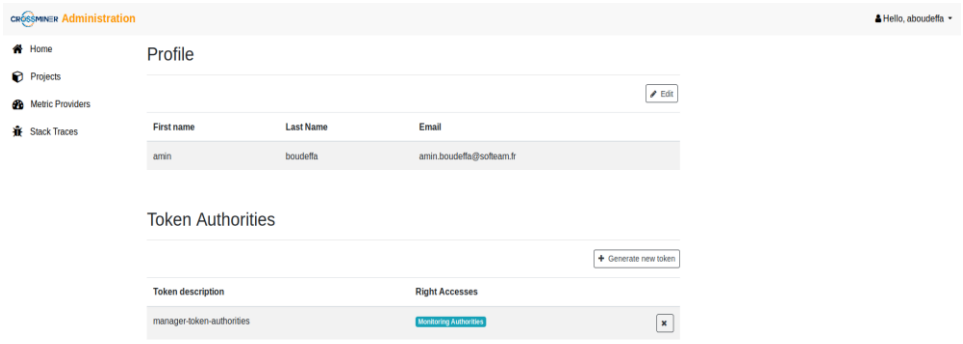

Figure 8 – Profile Edition and Authorization Management

AWEB5: Edit users' profile	**Primary Actor:** Platform Administrator
Description and Conditions	Allow to edit the registered users' account. All profile data or personal data will be edited during this operation.

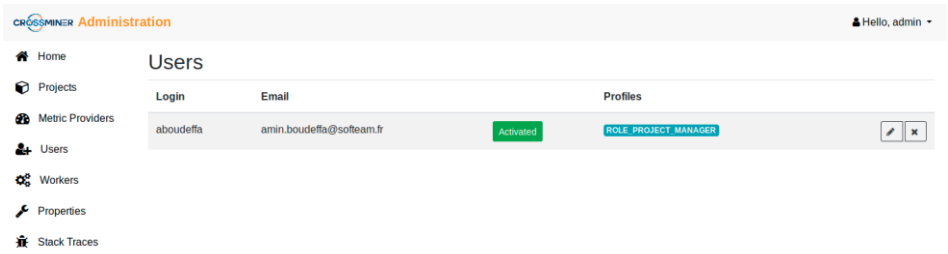

Figure 9 – Users Management

AWEB4: Manage users' rights	Primary Actor: Platform Administrator
Description and Conditions	A platform Administrator can change roles of a registered user with the aim of granting him new rights.

AWEB6: Delete registered user	**Primary Actor:** Platform Administrator
Description and Conditions	Allow to delete a registered user account. All profile data or personal data will be edited during this operation.

3.2.1.2 Project Analysis Configuration

The CROSSMINER Administration application allows to register open-sources project to be analysed by the platform services, as well as to manage and configure the analysis process in a fine-grain way by defining the analysis tasks that can be executed at will.

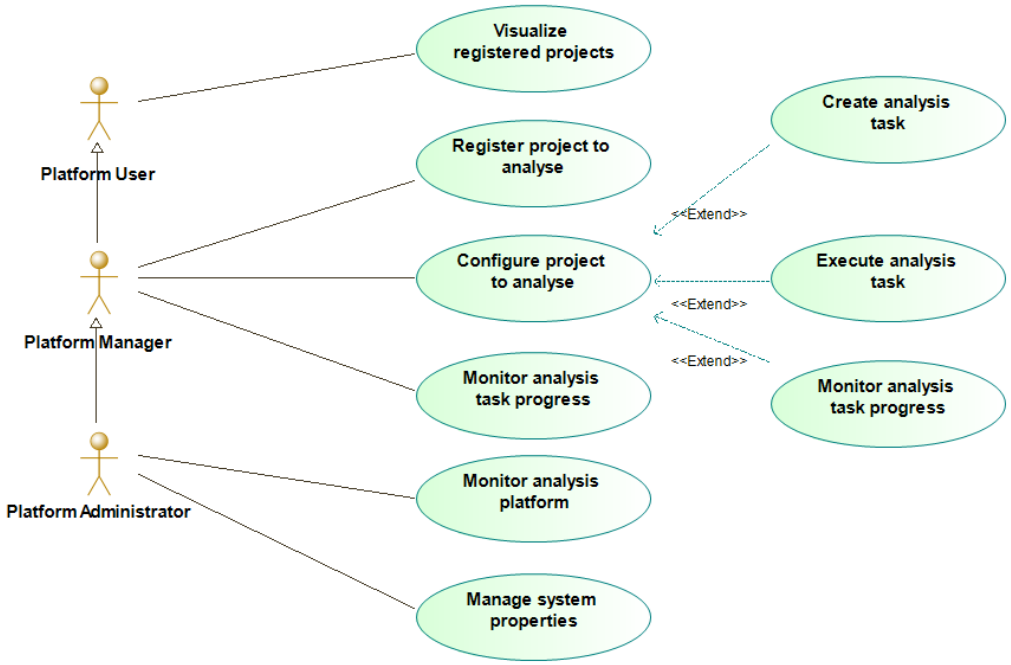

Figure 10 - Project Analysis Registration and Configuration

AWEB7: Visualize registered projects	**Primary Actor**: Platform User
Description and Conditions	Allow to check out the list of the registered projects details on the CROSSMINER platform.

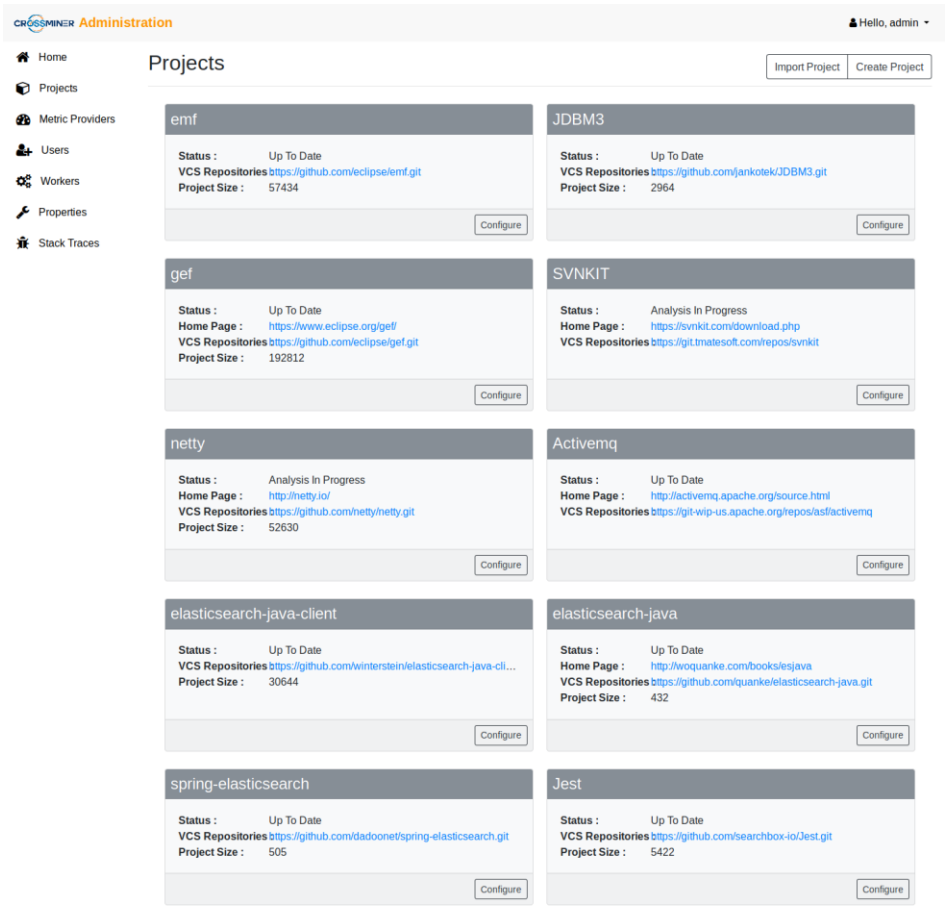

Figure 11 - Projects Visualization

AWEB8: Register project to analyse	**Primary Actor**: Platform User - Platform Manager
Description and Conditions	Allow to register a project on CROSSMINER platform for analysis purposes: • Provide the URL of the project repository (GitHub, Eclipse forge); • Define URLs of auxiliary data sources (issue tracker, message boards, etc.).

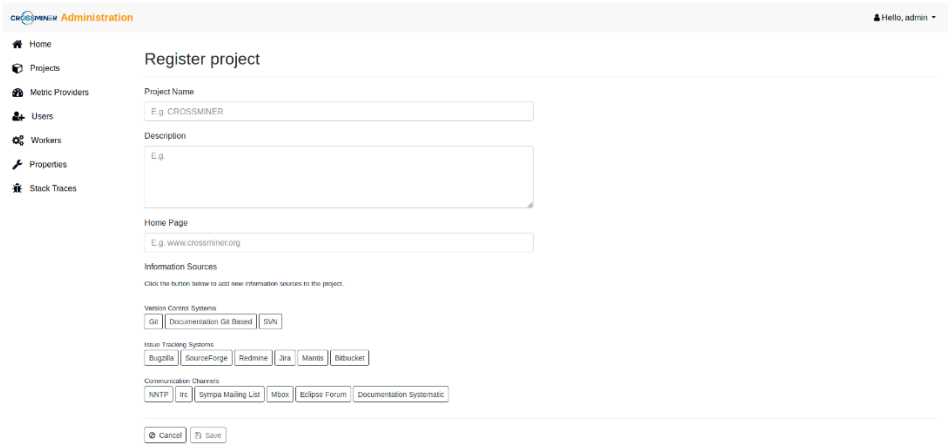

Figure 12 - Project Analysis Registration and Configuration

AWEB9: Configure project to analyse	**Primary Actor**: Platform User - Platform Manager
Description and Conditions	Allow to configure the registered project by applying a set of features: • Edit the project properties • Delete the registered project alongside the associated analysis data. • Manage analysis task within the project.

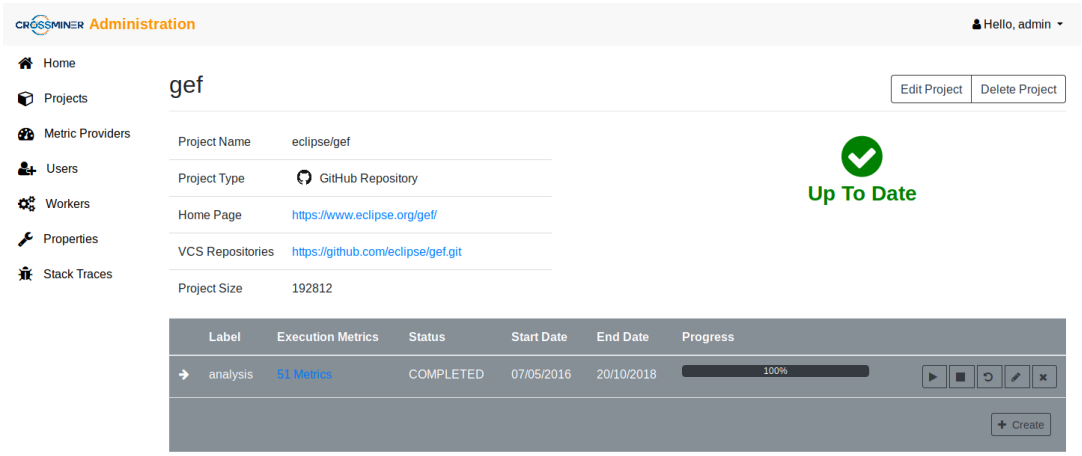

Figure 13 - Project Analysis Configuration

AWEB10: Create analysis task	**Primary Actor**: Platform User - Platform Manager
Description and Conditions	Schedule the execution of a registered analysis task. The task will be pushed into the platform analysis system including: • Select the list of metric providers that will be executed during a project analysis; • Select the Start and the End Date of the day to day analysis task; Define if the task will be executed once or if it is scheduled to be executed periodically.

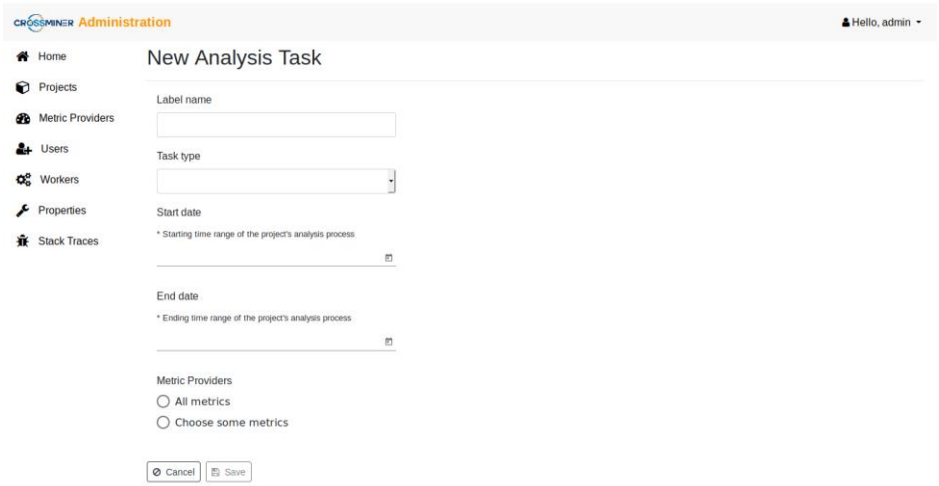

Figure 14 – Analysis Task Creation

AWEB11: Execute Analysis Task	**Primary Actor**: Platform User - Platform Manager
Description and Conditions	Schedule the execution of a registered analysis task. The task will be pushed into the platform analysis system.

AWEB12: Monitor Analysis Task Progress	**Primary Actor**: Platform User - Platform Manager
Description and Conditions	Access to information related to the status of the analysis task (Pending, In Progress, Up to Date). In the cases of task in progress, provide information about the progress of the task (%, estimated end date).

AWEB13: Monitor analysis platform	**Primary Actor**: Platform Administrator
Description and Conditions	Track the entire works progress including metric providers associated to the analysis task and the pending analysis tasks.

AWEB14: Manage system properties	**Primary Actor**: Platform Administrator
Description and Conditions	Allow to configure generic configurations applied to the metric-platform, e.g., GitHub OAuth tokens …

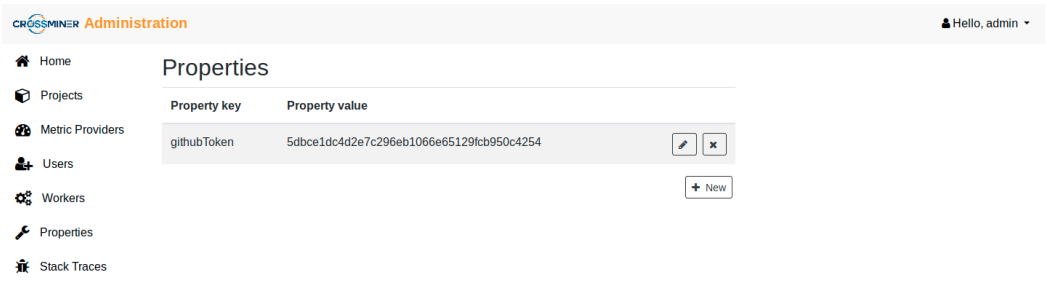

Figure 15 – Properties management

3.2.2 Technical Stack

The CROSSMINER Administration component is a web application based on Angular 6[4] and Bootstrap 4[5]. The delivered application needs to be deployed on a standard web server like Nginx[6], NodeJS[7] or Tomcat[8].

[4] https://angular.io
[5] https://getbootstrap.com
[6] https://www.nginx.com
[7] https://nodejs.org/en/
[8] https://tomcat.apache.org/

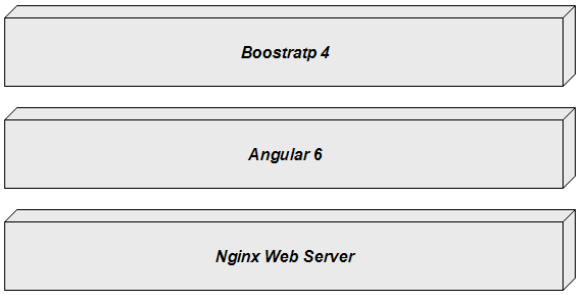

Figure 16 - CROSSMINER Administration Technological stack

The Administration application uses the REST APIs provided by the several CROSSMINER components to access analysis functions and the resulting data. The web application goes through the API Gateway component to access these services using a centralised and secured access point (more details about the API Gateway component are available in the dedicated chapter of this document).

The Administration application consume services from two CROSSMINER platform components:

- **The Authentication Service** which provide services related to user management, access right management and authentication.
- **The CROSSMINER Metric Platform** which allows to manage registration of open-source projects to be analysed by the platform, the execution of analysis tasks defined by the platform administration and allows the access to measurements resulting from the execution of analysis process.

The access to the platform data layer is performed using the intermediary of the REST services.

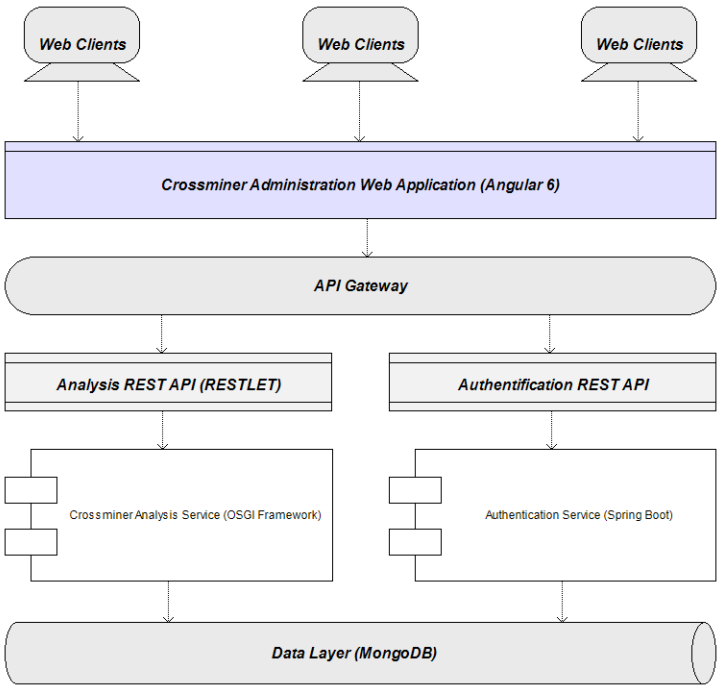

Figure 17 - Component architecture of CROSSMINER Administration

3.3 INTEGRATED PLATFORM

3.3.1 Component Overview

The CROSSMINER project extends the platform developed in OSSMERTER with new capabilities like new metric analysers, interproject analysis and library recommendations. To achieve this goal, a platform has been developed to support decision makers in the process of discovering, comparing, assessing and monitoring the health, quality, impact and activity of open-source software. Some of the components developed during the OSSMETER project were integrated to CROSSMINER.

In the early phases of CROSSMINER, we identified several axes of improvements of the components inherited from the OSSMETER project:

- **Centralized Architecture:** The OSSMETER application has been designed as a centralized application to facilitate its deployment. The discovery of scalability issue has pushed the architects to split the application into several components materialized by threads that are handled manually.

- **Scalability issue in project analysis process:** The analysis process of an open-source project involves execution of more than 300 metrics applied over the dataset for each day of the project, starting from the date of last analysis to the current date. Due to the potential dependencies between metrics of 2 successive temporal iterations, the execution of day by day analysis can't be parallelized. These constraints, in correlation with the low scalability of the architecture have the consequence of not allowing the parallelization of the behaviour when analysing a project.

- **Coarse-grained configuration:** In OSSMETER application, the analysis process of an open-source project is not configurable. All the metrics providers are always solicited during an analysis.

- **Limited level of feedback related to the analysis process:** Analysis of the history of an open-source project can take several days. The lack of information to the end users related the progress of the analysis process is problematic for the computation of this duration.

As part of the Integration Task, we worked to address with these problems by updating the architecture and adapting the analysis algorithms of open-source projects, modifying the communication channels between components, integrating a new administration web application and replacing the authentication mechanism implemented in OSSMETER.

3.3.2 Integration Platform API

Following the integration philosophy of the CROSSMINER platform, the Integrated Platform exposes all of these services through a REST API. This API offers different services related to the analysis of open source repositories such as the configuration of an analysis task, the monitoring of all the analysis processes or the access of metrics resulting from the analysis process.

Consumed mainly by the CORSSMINER platform's front-end components, the API is also intended to enable the automation of analysis operations and their integration with continuous integration systems such as Jenkins thus making it possible to strengthen the integration of the CROSSMINER platform in business processes.

In order to facilitate its integration, the API is fully documented through the SWAGER standard:

https://crossminer.github.io/scava-docs/developers-guide/api-reference-guide/metric-platform-api/

You will find below a sample of the key services provided by the API:

Project Configuration	Set of services which allow to administrate the analysis projects.	
/projects/create	POST	Create a new analysis project for a specific open source repository
/projects	GET	List analysis projects
/search	GET	Search a specific project using a query

Analysis tasks	Set of services which allow to administrate analysis tasks of a project	
/analysis/task/create	POST	Create a new analysis task in an analyses project by specifying metrics to execute.
/analysis/task/start	POST	Run an analysis task
/analysis/task/promote	POST	Increase priority of this analysis task on worker waiting queue.
/analysis/workers	GET	Retrieve the list of workers (execution process) configured on the analysis platform

Metric Providers	Set of services which allow to access to analysis results	
/raw/metrics/p/{projectid}	GET	Retrieve the list of metrics which has

		been computed for a specific project
/projects/p/{projectid}/m/{metricid}	GET	Retrieve values of metric a specific metric on a specific project
/analysis/metricproviders	GET	Retrieve the list of Metric Providers Kind of computable metrics) available on the analysis platform
/factoids	GET	Retrieve the list of Factoides (Kind of high level metrics) available on the analysis platform

3.3.3 Axes of improvements

3.3.3.1 Component granularity improvement

To address scalability and architectural issues identified in early phases of the project, we refactored the component model and the deployment schema of the actual application. By this operation, we have identified components with more restrained perimeters and ensured the scalability of each of these components.

Project Analyser: New component that supplants the Platform component. The role of this component is to compute the dependency graph of the unitary treatments required to analyse a project. From these unitary tasks, we differentiate the tasks of calculation of daily changes realized by the information managers and the tasks of computation of metrics performed by the metric providers. Once this graph is computed, the component will take care of the sequencing and parallelization of the execution of these different tasks.

Project Data Manager: New scalable component in charge of the calculation of a daily a list of changes on analysed project sources and other raw data.

Metric Provider Manager: Refactoring of Metric provider component in order to allow it to compute a partial list of metrics that depends on each other instead of calling all of the metrics providers.

This new division of the analysis platform services offers the possibility to modify the algorithm of projects analyses, to parameterize its behaviour and paralyze some of these operations.

3.3.3.2 Inter-Component Communications

In the platform architecture inherited from OSSMETER project, the inter-component communication system was based on the recording of tasks in a shared database and did not offer the necessary flexibility. In the new version of this platform, we have redesigned the remote communication system between components based on the REST services to manage synchronous communications.

To facilitate the integration between the heterogeneous (technologic heterogeneity and functional independences) components of the CROSSMINER platform, we decided to

recommend the usage of a common approach to expose public services to clients and to the other components: the deployment of a REST API. The REST approach is a set of architectural constraints for implementing synchronous web services over HTTP and provide interoperability between systems on the Internet.

3.3.3.3 Redesign of Administration Services

In order to address the Coarse-grained configuration issue and the lake of information available for project managers and platforms administrators, we decided to do a complete replacement of the platform administration web application. Previously integrated to the metric dashboard, the administration interface is now an independent application.

Coupled with the refactoring of the project analysis algorithm provided by the analysis platform, the new application now allows to precisely configure the analysis of each open-source project repository by defining the list of metrics providers executed, the execution period and the scheduling of the analysis task. Information views about the state of each analysis process and the load of the different components of the platform analysis are now available for project managers and platform administrators. More information related to the Administration application is available in the dedicated section of this document.

3.3.3.4 New Authentication Mechanism

Based on the patterns developed in the microservices architectures, we decided to implement a central authentication mechanism for the entire platform as an independent service. This authentication service brings together in one component all the functionalities necessary for the management of the users of the platform and the authentication mechanism:

- User Management: creation, validation, deletion of user accounts;
- Rights Management: management of the level of authorisation of all users, using a role-based system;
- Authentication: deployment of a centralised authentication system for all services and web application provided by the CROSSMINER platform based on JSON Web Tokens (JWT) technology.

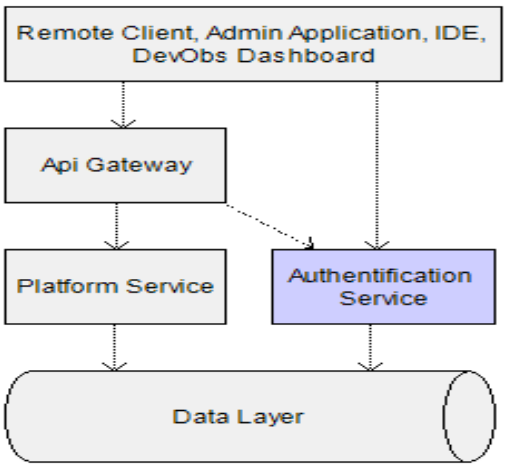

Figure 8 - Integration of the Authentication Service in CROSSMINER Architecture

The Authentication service is directly used by remote clients, the administration application, the integrated development environment and the DevOps Dashboards. Coupled with the API Gateway component, it provides also an authentication mechanism for all REST services provided by the platform's components.

3.4 CONTINUOUS INTEGRATION

Continuous Integration (CI) is one of the good practice setup for the industrialisation of the CROSSMINER product. It ensures that all changes can be built and tested independently of the local environment of the developer and provides automatic feedback on branches and pull requests. Continuous Integration also builds the official deliverables and serves as a common build reference for all partners.

In this chapter, we describe the development guidelines followed by the consortium, as well as the setup of the infrastructures required for enabling the use of Continuous Integration.

3.4.1 Development Guidelines

Several guidelines were defined to guide the CROSSMINER consortium in the development of the several components of the CROSSMINER platform. These guidelines specify a common strategies and good practices that all CROSSMINER developers must adhere to ensure coherent collaborative development of CROSSMINER software components. The guidelines can be grouped in two main categories:

3.4.1.1 Source Code Repository:

All the source code of CROSSMINER components must be reside in a single repository, where components can be contained in different subdirectories. This will ensure that every developer has a working copy of the entire project. All code is available in the following a repository: https://github.com/crossminer/scava

The version control used is the Git, where different branches will be used to separate and manage different stages of development. The branch model will follow a master-development-feature approach, represented in Figure 9. In this model, the **master** branch is the main branch, which is always clean, builds and runs all the tests

successfully. Other branches will be created for the different **features/components**, which will be eventually merged with the develop (**dev**) branch. When the **dev** reaches a state where all the components can be built without errors and it is verified that all the integrated components work and interact as expected, then the conditions are met for a merge with the **master** branch.

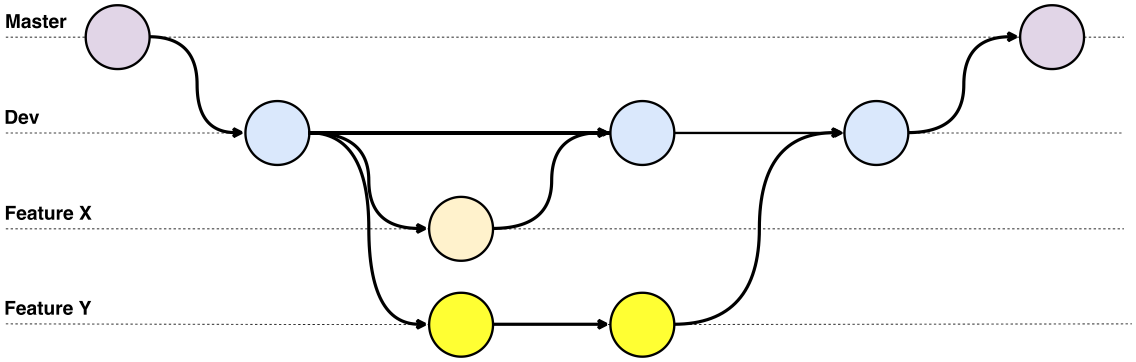

Figure 9 – Example of the Git branch model used

3.4.1.2 Building automation

Every CROSSMINER component must be able to be built in an automated way. This is a requirement for the process of CI, necessary for the automation of the testing and deployment activities. Since different components are implemented using different technologies, each component can use its own building technology. The selection of the building technology is of the responsibility of development team working on the component but requiring that the chosen technology must be supported by the CI server used.

3.4.2 Building and Testing

A Jenkins instance[9] was setup to support the automatic building and testing of CROSSMINER software. Jenkins[10] is one of the most used CI open-source engines, both in the industry and the free software community. Furthermore, it is the official build engine at the Eclipse forge[11], and the Eclipse Foundation offers Jenkins-based CI services for its hosted projects. This will enable us to easily switch to the Eclipse infrastructure, or to any hosting that uses Jenkins. In our setup, we used the most recent version of Jenkins, and the new Jenkins pipeline language for the reproducible definition of CI processes. As represented in Figure 10, it was decided to setup the building and testing jobs at the component level in order to watch the individual evolution of the components' build status and stability. All jobs are then organised into a common Jenkins build definition file that builds the full product, runs the tests and produces binaries for the various available platforms.

[9] http://ci5.castalia.camp:8080/
[10] https://jenkins.io/
[11] https://ci.eclipse.org/

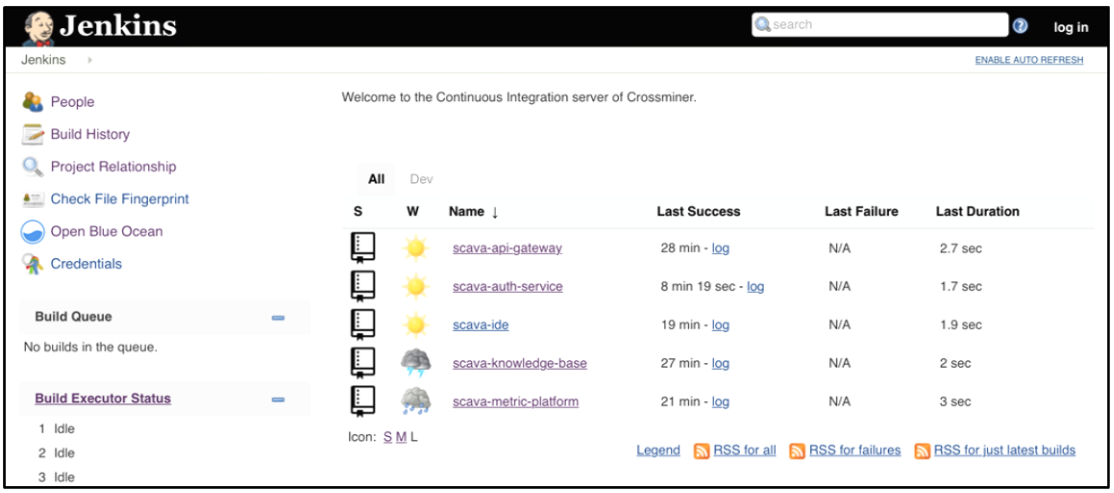

Figure 10 – Screenshot of the Jenkins dashboard showing the different CI jobs

Figure 11 shows the pipeline definition in the Blue Ocean view of Jenkins. This definition specific the workflow of CI tasks, specifying the building of the individual components and publishing steps.

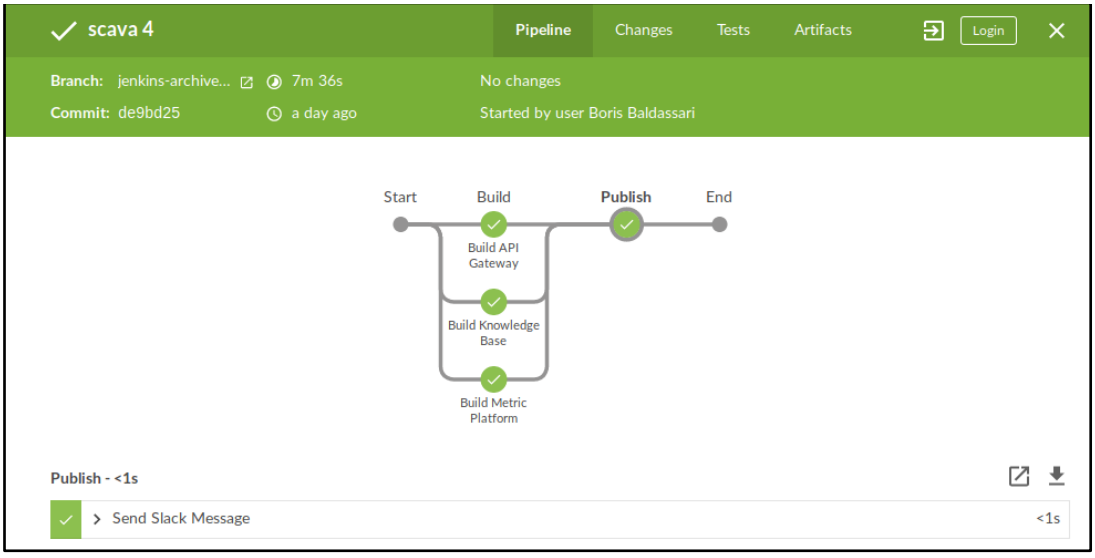

Figure 11 – Definition of the workflow of CROSSMINER CI process

A feedback loop was implemented to bring the build results back to GitHub[12] after changes (changes are pulled every hour to spare resources). This provides a quick check for committers and reviewers, increasing both ease of use and reliability. Figure 12 shows the build results in the GitHub interface.

[12] https://github.com/crossminer/scava/pull/6

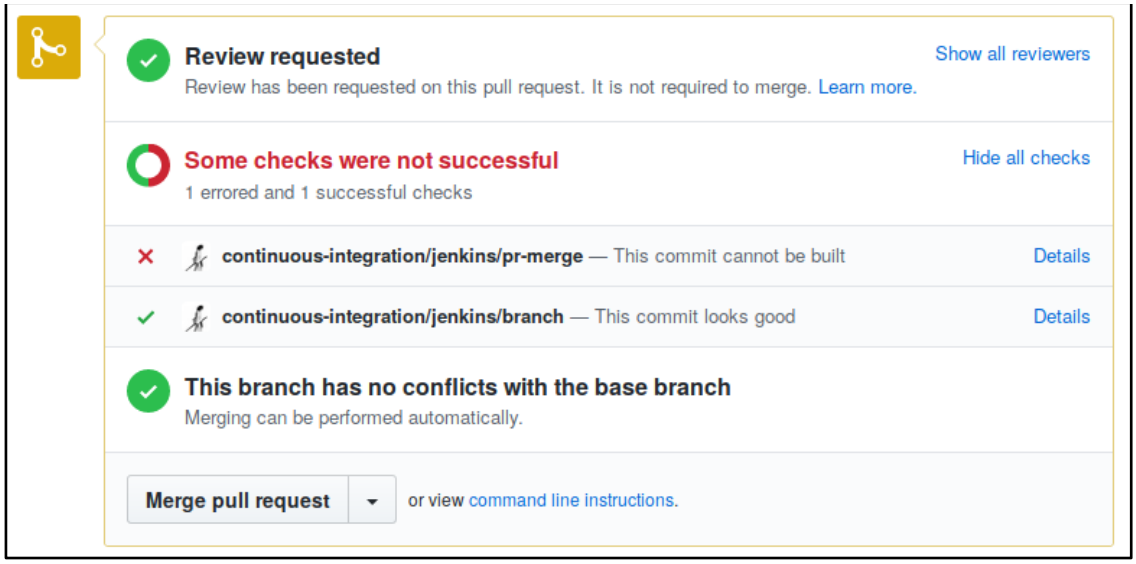

Figure 12 – Feedback of the CI server about the state of branch in "Pull Request"

4 PLATFORM DEPLOYMENT

4.1 OVERVIEW

Each of the CROSSMINER components has its own procedures and requirements for building and deployment. In order to simplify this process, CROSSMINER consortium provides a Docker setup to create deployments of the whole platform. Such Docker provides some interesting advantages like:

- Easy to deploy procedure;
- Portability;
- Scalability;
- Some level of configurability of the setup (selection of the components to be deployed);
- Independent execution environment for each component, each one with their specific dependencies, decreasing the probability of conflicts.

Using Docker, CROSSMINER platform is deployed as a distributed application, where Docker images are created to run one or more components. The current CROSSMINER application consists of 11 Docker images that communicate between each to accomplish their designed functions. The images and their communication flows are represented in Figure 13.

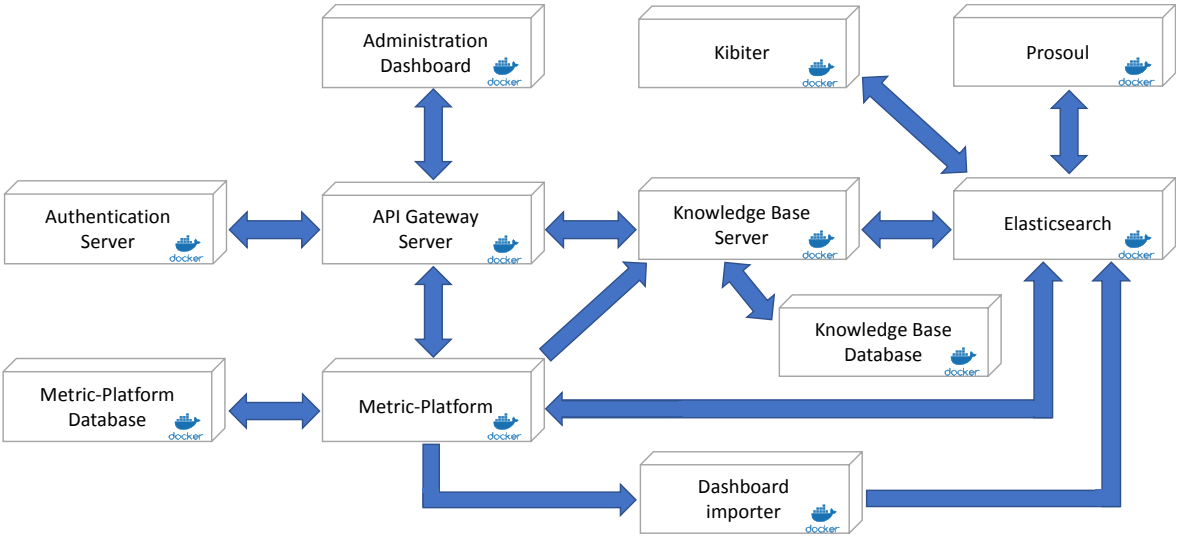

Figure 13 – Diagram of CROSSMINER as a Docker distributed application

Each Docker image can aggregate several components of the CROSSMINER architecture (Figure 2). The functions provided by each Docker image is summarised below:

- Metric-Platform – Image with the components of the Metric Platform[13] and some metric providers developed in CROSSMINER. This Docker image can be run as different containers (1 master and multiple slaves to compute metrics).

- Metric-Platform Database – MongoDB image to store the data of the metric platform. Currently runs as one container but may be replaced by a replication or sharded MongoDB cluster if needed.

- Administration Dashboard – Dashboard in charge of the authentication and management of users, import and configuration of open-source project analysis and the monitoring of the platform's execution process.

- Knowledge Base Service – Image aggregating the components implementing the Knowledge Base[14]. It is used to run the cross-project analyses and provides the API to give access computed data.

- Knowledge Base Database – MongoDB image to store the data from the analyses performed by the Knowledge Base Service.

- Dashboard Importer – Image with the Perceval Backend and the Grimoire ELK components to extract and enrich data from project analysis and provide it to the Dashboards through the support of the Elasticsearch data storage. The Perceval backend[15] is an extension to the Perceval data collection tool[16] that will collect data from the Knowledge Base and Metric Platform databases using the API REST.

[13] https://github.com/crossminer/scava/tree/dev/metric-platform
[14] https://github.com/crossminer/scava/tree/dev/knowledge-base/org.eclipse.scava.knowledgebase
[15] https://github.com/crossminer/scava/tree/dev/web-dashboards/perceval-scava
[16] https://github.com/grimoirelab/perceval

- API Gateway Server – The entrypoint to access to the CROSSMINER analysis components that provides authentication and access authorisation mechanisms through the Authentication Server.

- Authentication Server – Server responsible for the management of the users of the CROSSMINER platform. It is the main actor on validation of the authentication attempts and decision about the access permissions of each user.

- Elasticsearch – Consists on the database used to store the output of the CROSSMINER Data Importer and NLP related indexes.

- Kibiter -Is in charge of starting the Kibiter components used to visualize and interact with the Web-based dashboards.

- Prosoul - Provides CROSSMINER means to assess quality models based on the metrics collected by the platform.

The CROSSFLOW tools are deployed in a setup independent of the rest of the CROSSMINER technologies. It consists on a web interface the provides the functionalities to load workflow packages, execute them by deploying instances of the Workflow execution engine and execution monitoring. This tool is provided as a independent Docker image[17] available at Docker Hub. The usage of this tool is very specific to the scenario that the CROSSMINER user needs to implement and therefore it was not included to the generic deployment of CROSSMINER technologies. As such the deployment of this tool will not be covered in this deliverable. More Information can about this tool can be found in deliverable D5.5 and on CROSSMINER's official documentation.

4.2 DEPENDENCIES

The following table identifies the software dependencies that must be installed on each Docker Image to allow to successful deploy and run the respective CROSSMINER components. The Metric-Platform requires the installation of Python3, Puppeteer18, PDM19 and Puppet-lint20 to be able to execute the metric providers related with the System Configuration Code Smell Miner component, and the installation of Maven to perform the analysis of Maven configurations. The Dashboard Importer requires an environment with Python 3 installed to run Perceval. Moreover, Dashboard Importer has as dependencies Grimoire21 and Perceval22. These dependencies are downloaded and built by scripts in the Perceval CROSSMINER, requiring the installation of Git and build tools like the GCC. Similar requirements are needed to run the Prosoul. *Knowledge Base Service*, *API Gateway Server* and *Authentication Server* just require an environment with Java 8. The *Administration Dashboard*, since is a Javascript application, just requires a HTTP server to make it available.

[17] https://hub.docker.com/r/crossminer/crossflow
[18] https://github.com/tushartushar/Puppeteer
[19] https://pmd.github.io
[20] http://puppet-lint.com
[21] https://github.com/chaoss/grimoirelab-toolkit/#egg=grimoirelab-toolkit
[22] https://github.com/chaoss/grimoirelab-perceval/#egg=perceval

Table 8 - Execution dependencies for each Docker image

Docker Image	Dependencies installed
Metric-Platform	Java 8 (JDK) Python 3 Maven Puppeteer PDM Puppet-lint
Administration Dashboard	HTTP Server (Nginx)
Knowledge Base Service	Java 8 (JDK)
Dashboard Importer	Python3 Git C compiler (GCC)
API Gateway Server	Java 8 (JRE)
Authentication Server	Java 8 (JRE)
Prosoul	Python3 Git C compiler (GCC)

Elasticsearch and *Kibiter* are provided as a Docker images available at Docker Hub. The Metric-Platform Database and Knowledge Base Database correspond to instances of MongoDB[23] database

4.3 CONFIGURATIONS

The deployment of the CROSSMINER platform is performed by using the Docker Compose[24] tool. This tool uses a YAML file (typically named docker-compose.yml) to define the images to be run and to specify the configurations about how these images will run and interact. With a single command - `docker-compose up` – containers are generated for all images, private networks are setup between the containers, storage volumes are linked to containers, network ports are exposed., etc. Containers are grouped in services and service names are used as hostnames to address communication between containers. Eleven services where defined in the docker-compose.yml for the CROSSMINER deployment. These services are described below.

4.3.1 admin-webapp

This service starts the Administration Dashboard on to allow the managing of administrative aspects of the platform, such as the managing of projects and analysis tasks to be executed. By default, the *admin-webapp* is exposed on port 5601 on the localhost. This container depends on the API Gateway server and must be configured with the address public address for that service.

[23] https://www.mongodb.com
[24] https://docs.docker.com/compose/overview/

```
admin-webapp:
    build: ./web-admin
    environment:
        - API_GATEWAY_VAR=http://localhost:8086
    depends_on:
        - api-server
    networks:
        - default
    expose:
        - 80
    ports:
        - "5601:80"
```

Listing 1 – Description admin-webapp service

4.3.2 api-server

This service executes the API Gateway service, entrypoint for the REST API of CROSSMINER. The it requires the execution of the metric platform and Authentication server to support its activities. By default, the gateway is exposed in port 8086.

```
api-server: # API gateway to route the access of REST APIs
    build: ./api-gw
    depends_on:
        - oss-app
        - auth-server
    networks:
        - default
    expose:
        - 8086
    ports:
        - "8086:8086"
```

Listing 2 – Description api-server service

4.3.3 auth-server

Authentication service used by the *api-server*. By default is expose on port 8085.

```
auth-server: # Server responsible for the authentication of the
    build: ./auth
    entrypoint: ["./wait-for-it.sh", "oss-db:27017", "-t", "0", "--", "java", "-
jar", "scava-auth-service-1.0.0.jar" ]
    depends_on:
        - oss-db
    networks:
        - default
    expose:
        - 8085
    ports:
        - "8085:8085"
```

Listing 3 – Description api-server service

4.3.4 oss-app

The oss-app service corresponds to the containers running the Metric-Platform as ‘master’ and providing the REST API to interact with the platform. This service starts after the metrics’ database (*oss-db*) and expose the port 8182 to the Docker internal network in order to allow the containers in the network to consume the API provided by this service. It requires also requires the availability of an Elasticsearch container to compute some metrics.

```
oss-app: #Deploys a container with the OSSMETER platform configured to act as
master and to run the REST API server
    build: ./metric-platform
    entrypoint: ["./wait-for-it.sh", "oss-db:27017", "-t", "0", "--",
"./eclipse", "-master", "-apiServer", "-worker", "w1", "-config",
"prop.properties"]
    depends_on:
        - oss-db
        - elasticsearch
    networks:
        - default
    expose: #exposes REST API port
        - 8182
    ports:
        - "8182:8182"
```

Listing 4 – Description of oss-app service

In order for this application to work on this setup, the file `prop.properties` was modified to `mongo_hosts=oss-db:27017`, to specify how to connect to the container running the database.

4.3.4.1 oss-slave

This service is similar to the *oss-app*, where containers run the Metric-Platform but with the 'slave' configuration (no job schedule nor REST API). Multiples containers can be deployed in the context of this server and will communicate with the existing 'master' of the service oss-app. Before starting, this service needs to wait for the start of the services *oss-db* and defines *oss-app* as dependence.

```
oss-slave: #Deploys a container with the OSSMETER platform configured to act as
master and to run the REST API server
    image: scava-deployment_oss-app
    entrypoint: ["./wait-for-it.sh", "oss-db:27017", "-t", "0", "--",
"./eclipse","-worker", "w2", "-config", "prop.properties"]
    depends_on:
        - oss-app
    networks:
        - default
```

Listing 5 – Description of the oss-slave service

4.3.5 oss-db

This service consists in a MongoDB container to store the data collected and computed by the Metric-Platform. This container exposes the port 27017 to allow the other services to connect to it. Optionally this port can also be mapped to the host machine to allow the use of DB management tools. A volume can be associated to the container in order to persist the data on the filesystem of the host machine.

```
   oss-db: # data storage service
        image: mongo:3.4 #current setup uses mongodb
        networks:
            - default
        expose:  #exposes database port to oss-web and oss-app
            - 27017
        volumes: #creates volume on container
            - ~/oss-data:/data/db
 # Uncomment the port map below to give access to external mongo database visu-
alizers
        ports:
            - "27017:27017"
```

Listing 6 – Description of the oss-db service

4.3.6 kb-service

This service runs the container with the Knowledge-Base components. It requires the service *kb-db* to be already running in order to store the computed data and the service oss-*app* as it will be used as data source. The REST API provided is exposed in the port 8080.

```
    kb-service: #deploys the Knowledge Base services
        build: ./KB  # directory to the corresponding Dockerfile
        depends_on:
            - kb-db #only requests for kb-db service to be launched before this
service. DB may still not be ready when this service starts
            - oss-app
        networks:
            - default
        # maps the port 8080 in the localhost to port 8080 of container
"HOST:CONTAINER"
        ports:
            - "8080:8080"
```

Listing 7 - Description of the kb-service

The following configuration was specified in the file `application.properties` to allow this container to communicate with the containers of the services *kb-db and oss-app*.

```
spring.data.mongodb.host=kb-db
spring.data.mongodb.port=27017
spring.data.mongodb.database=CROSSMINER
lucene.index.folder=/home/root/CROSSMiner_lucene/
ossmeter.url=http://oss-app:8182/
```

Listing 8 - Configuration of the kb-service

4.3.7 kb-db

This service consists in a MongoDB container to store the data collected and computed by the Knowledge-Base. This container exposes the port 27017 to allow the other services to connect to it. Optionally this port can also be mapped to the host machine to allow the use of DB management tools. A volume can be associated the containers ran by this server in order to persist the data on the filesystem of the host machine. Optionally, a Docker image can be built to run a set of steps to restore a dump that will load previously computed data into the DB.

```
    kb-db: # data storage service for the Knowledge Base services
        build: ./KB-db #restores a dump of the of te
     #    image: mongo:3.4 #current setup uses mongodb
        networks:
            - default
        expose:  #exposes database port to oss-web and oss-app
            - 27017
#        volumes: #creates volume on container
#            - ~/kb-data:/data/kb-db
 # Uncomment the port map below to give access to external mongo database visu-
alizers
        ports:
            - "27018:27017"
 #   Current database setup consists in a single db instance. May be changed to
a
 # replication deployment to increase fault tolerance and to better handle the
number of
 # read and write operations
```

Listing 9 – Description of the kb-db service

4.3.8 dashb-importer

This service runs the container with the Perceval backend for CROSSMINER. This service must wait for the start of the service *oss-app*, *elasticsearch* and *kibiter* because it requires the REST API provided by these systems to have access to the CROSSMINER data, and it uploads the obtained data to Elasticsearch and configure the dashboards running on *kibiter*.

```
build: ./dashboard
    entrypoint: ["./wait-for-it.sh", "oss-app:8182", "-t", "0", "--",
                 "./wait-for-it.h", "elasticsearch:9200", "-t", "0", "--",
                 "./wait-for-it.sh", "kibiter:5601", "-t", "0", "--",
                 "./starter.sh"]
    depends_on:
        - oss-app
        - elasticsearch
        - kibiter
    networks:
        - default
    environment:
        - DASHDEBUG=0
        - BULKSIZE=500
        - NO_LOOP=0
```

Listing 10 – Description of the dashb-importer service

4.3.9 elasticsearch

This service is started at the beginning of the docker-compose and does not depend on other services. It sets up the Elasticsearch database used to store the output of the *dashb-importer* and NLP related indexes. It uses an image hosted on Docker Hub[25].

```
elasticsearch:
    image: acsdocker/elasticsearch:6.3.1-secured
    command: /elasticsearch/bin/elasticsearch -E network.bind_host=0.0.0.0 -
Ehttp.max_content_length=500mb
    networks:
       - default
    expose:
       - 9200
    ports:
       - "9200:9200"
    environment:
       - ES_JAVA_OPTS=-Xms2g -Xmx2g
       - ES_TMPDIR=/tmp
```

Listing 11 – Description of the elasticsearch service

4.3.10 Kibiter

This service is in charge of starting the Kibiter components used to visualize and interact with the Web-based dashboards. It is based on an image hosted on the Docker Hub[26] and depends on *elasticsearch* service.

[25] https://hub.docker.com/r/acsdocker/elasticsearch
[26] https://hub.docker.com/r/acsdocker/grimoirelab-kibiter

```
kibiter:
    image: acsdocker/grimoirelab-kibiter:crossminer-6.3.1
    networks:
        - default
    depends_on:
        - elasticsearch
    expose:
        - 5601
    ports:
        - "80:5601"
    environment:
       - ELASTICSEARCH_URL=https://elasticsearch:9200
```

Listing 12 – Description of the Kibiter service

4.3.11 Prosoul

This service provides CROSSMINER to assess quality models based on the metrics collected by the platform. It starts after the *elasticsearch* and *kibiter* services.

```
prosoul:
    build: ./quality-model
    ports:
        - "8000:8000"
    depends_on:
        - kibiter
        - elasticsearch
    environment:
        - ALLOWED_HOSTS=prosoul localhost 127.0.0.1
        - ES_URL=https://admin:admin@elasticsearch:9200
        - KIBITER_URL=https://kibiter:80
        - KIBITER_HOST=http://localhost:80
        - DASHDEBUG=0
        - SYNC_ES_SCAVA=0
        -
QM_URL=https://raw.githubusercontent.com/Bitergia/prosoul/master/django-
prosoul/prosoul/data/qmodel_crossminer.json
        - QM_NAME=crossminer
```

Listing 13 – Description of the dashb-importer service

5 DOCUMENTATION

The CROSSMINER consortium compiled a set of documentation to provide all the CROSSMINERS stakeholders with the information about how to interact with the all CROSSMINER technologies. A repository was created to store and organize this information. You can find it here: https://github.com/crossminer/scava-docs. A site generator, MkDocs[27], was configured to use the information in this repository and automatically generate and update a website: https://scava-docs.readthedocs.io.

Figure 14 presents the home page of website generated. The documentation is divided into 4 sections, each one targeting a different type of information.

[27] https://www.mkdocs.org

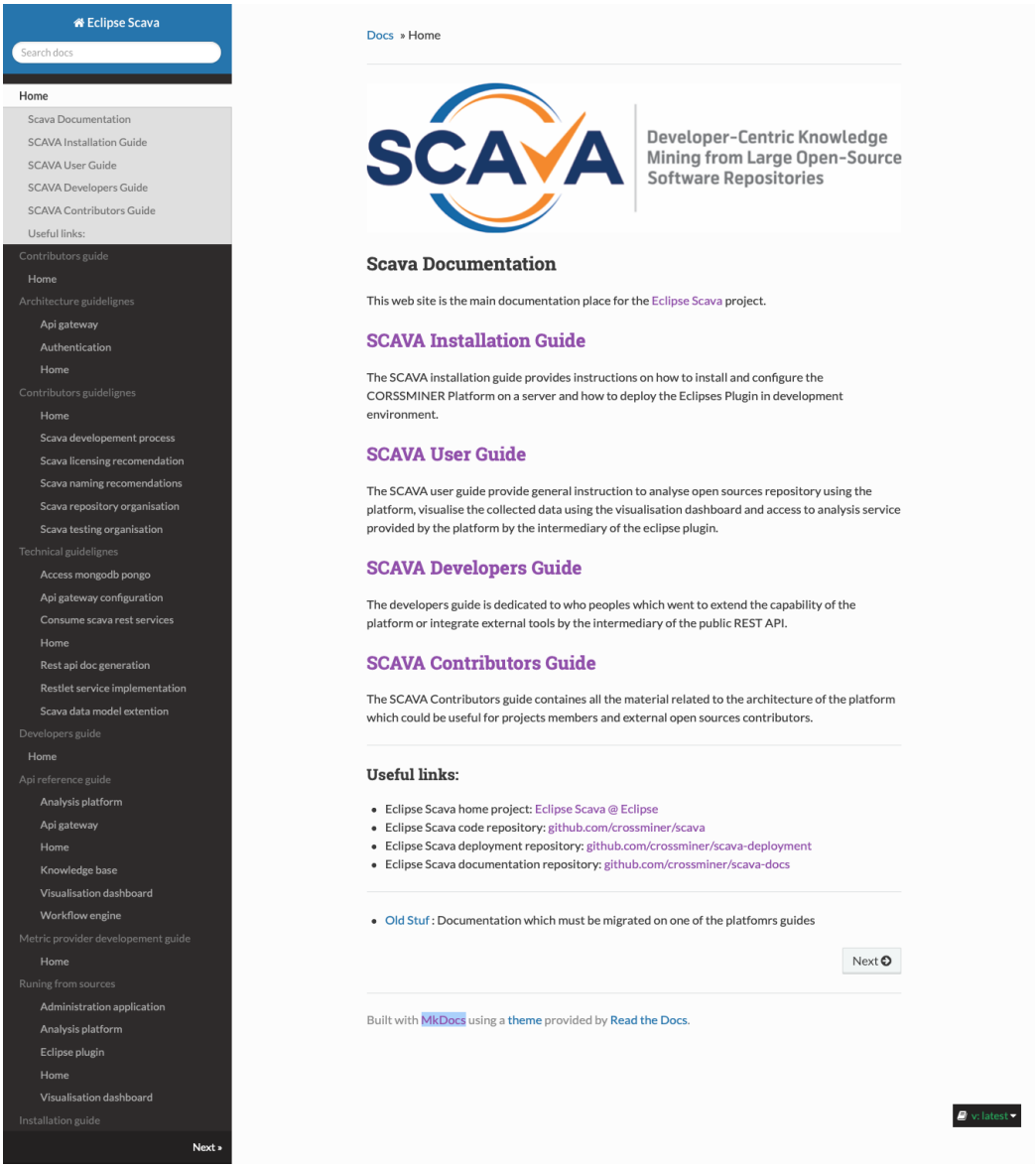

Figure 14 – CROSSMINER Documentation website

The purpose of each section is:

1. Installation Guide providing information about the steps and requirements for the installation and configuration of CROSSMINER technologies;

2. User Guide consisting in a collection of guides about how to use CROSSMINER technologies;

3. Developers Guide, which provides information oriented to developers, explaining how the capabilities of CROSSMINER technologies can be extended and how external tools can be integrated with CROSSMINER technologies; and

4. Contributors Guide providing descriptions about the internal development methodologies followed in CROSSMINER and providing

guidelines to external entities about how they can also contribute for CROSSIMER.

The Documentation website also provides a search bar, making it easy and quick to search for any specific content.

6 SUMMARY

This document provided an update to the CROSSMINER Architecture initially presented in D8.3. These changes correspond to some minor modifications on the relations between some components related with the Natura Language Analysis and to the addition of new components to manage and regulate the access of clients to the APIs provided by CROSSMINER components. The functionalities of these components, API Gateway and Authentication Services, are then described. Details are also given concerning the implementation decisions and technologies used.

This document also provided an update about the current status of integration between the CROSSMINER components and the technologies from the OSSMETER projects used and improved in CROSSMINER.

It was also presented the development guidelines proposed to consortium concerning the development of the CROSSMINER components and about the building and testing environment that is being used to continuously test the stability and functionalities of the tools developed.

It is presented the deployment setup proposed and supported by the CROSSMINER consortium, which consists of container-based distributed application based on the Docker Compose technology. This deployment solution provides a simple way for deploying and configuring a CROSSMINER instance with a high-level of portability.

At last, it was described the documentation prepared by the CROSSMINER consortium to provide anyone interested in CROSSMINER technologies with information about how to install, used, integrate and develop, and how to contribute to the open-source project.

APPENDIX A: CROSSMINER ARCHITECTURE