

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**MÁSTER UNIVERSITARIO EN INGENIERIA DE
REDES Y SERVICIOS TELEMÁTICOS**

TRABAJO FIN DE MÁSTER

**Design and implementation of a monitoring framework for a
DevOps life-cycle based on semantic techniques and the
OSLC standard**

**VÍCTOR ÁLVAREZ PROVENCIO
2022**

TRABAJO DE FIN DE MASTER

Título: Diseño e implementación de un framework de monitorización para el ciclo de vida DevOps basado en técnicas semánticas y en el estándar OSLC

Título (inglés): Design and implementation of a monitoring framework for a DevOps life-cycle based on semantic techniques and the OSLC standard

Autor: Víctor Álvarez Provencio

Tutor: Álvaro Carrera Barroso

Departamento: Departamento de Ingeniería de Sistemas Telemáticos

	Presidente:	—
MIEMBROS DEL TRIBUNAL CALIFICADOR .	Vocal:	—
	Secretario:	—
	Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes



TRABAJO DE FIN DE MÁSTER

Design and implementation of a monitoring framework for a
DevOps life-cycle based on semantic techniques and the OSLC
standard

2022

Resumen

El uso de la metodología DevOps ha tenido un importante incremento en empresas del ámbito tecnológico. Esto se debe a la agilidad que proporciona al ciclo de vida de un producto software así como la facilidad de integración entre dos de los campos mas importantes en las empresas tecnológicas: desarrollo y operaciones. Por un lado, una relación mas fluida entre los componentes de estos dos campos supone una mayor rapidez de gestión ante cambios inesperados, así como permite la automatización de despliegues o procesos que pueden ser de vital importancia para responder ante una subida de demanda de servicios por parte de los usuarios.

En la arquitectura tecnológica de estas empresas suelen utilizarse mas de una herramienta DevOps, cada una para la fase del ciclo de vida para la que sea útil. Esto supone la necesidad de tener estas herramientas interrelacionadas entre sí de manera interoperable y sencilla de gestionar. Esto no siempre resulta posible ya que cada herramienta puede tener sus propias configuraciones e independientes del resto de herramientas. Por otro lado, tener correctamente monitorizadas estas herramientas para poder responder ante cambios anómalos o ante cambios en otras herramientas de la arquitectura, se convierte en uno de los pilares fundamentales de estas empresas.

Bajo este contexto, este trabajo plantea una solución al problema de integración de las diferentes herramientas DevOps mediante el uso del estandar abierto OSLC y los principios de la web semántica. Por otro lado, este trabajo plantea un framework de monitorización de métricas software de la herramienta Stackstorm, así como de métricas sociales mediante la extracción y análisis de datos procedente de fuentes sociales como Twitter. Este trabajo forma parte del proyecto SmartDevOps¹ cuya función es integrar herramientas de DevOps con tecnologías semánticas y enfoque Big Data y este trabajo pretende servir de ejemplo para un caso de uso concreto del mismo.

Palabras clave: OSLC, DevOps, Stackstorm, SmartDevOps, Linked Data, Web Semántica

¹<https://smartdevops.gsi.upm.es>

Abstract

The adoption of the DevOps method has increased considerably in technology companies. This is due to the agility it provides to the life cycle of a software product, as well as the ease of integration between two of the most important fields in technology companies: development and operations. On the one hand, a more fluid relationship between the components of these two fields means faster management of unexpected changes, as well as allowing the automation of deployments or processes that can be of vital importance to respond to an increase in demand for services by the IT company demand for services from users.

In the technological architecture of these companies, more than one DevOps tool is usually used, each for the lifecycle phase for which it is useful. This implies the need to have these tools interrelated with each other in a way that is interoperable and easy to manage. This is not always possible, since each tool can have its own configurations and be independent of the rest of the tools. On the other hand, having these tools correctly monitored to be able to respond to anomalous changes or changes in other tools of the architecture becomes one of the fundamental pillars of these companies.

In this context, this work proposes a solution to the problem of integration of different DevOps tools through the use of the open standard OSLC and the principles of the semantic web. On the other hand, this work proposes a framework for monitoring software metrics of the Stackstorm tool, as well as social metrics by extracting and analyzing data from social sources such as Twitter. This work is part of the SmartDevOps project whose function is to integrate DevOps tools with semantic technologies and Big Data approach, and this work is intended to serve as an example for a specific use case.

Keywords: OSLC, DevOps, Stackstorm, SmartDevOps, Linked Data, Semantic Web

Agradecimientos

A mis padres.

Agradecer también al personal del Grupo de Sistemas Inteligentes, en especial a Álvaro Carrera Barroso, por la ayuda todo este tiempo y a Jorge y a Guillermo por hacer el trabajo mas ameno.

A los trabajadores y trabajadoras de la Universidad Politécnica de Madrid.

Contents

Resumen	VII
Abstract	IX
Agradecimientos	XI
Contents	XIII
List of Figures	XVII
Listings	XIX
1 Introduction	1
1.1 Context	1
1.2 Motivation	3
1.3 Project goals	4
1.4 Structure of this document	5
2 Enabling Technologies	7
2.1 Development and Operations (DevOps)	8
2.1.1 StackStorm	10
2.1.2 Airflow	11

2.1.3	MongoDB	13
2.1.4	Apache Kafka	14
2.1.5	The Elastic Stack	16
2.1.5.1	Elasticsearch	16
2.1.5.2	Kibana	18
2.1.5.3	Logstash	18
2.2	Semantic Web	19
2.2.1	Linked Data	21
2.2.2	RDF (Resource Definition Framework)	22
2.2.3	Protégé	24
2.2.4	Social metrics and sentiment analysis	25
2.2.4.1	GSICrawler	26
2.2.4.2	Senpy	27
2.2.5	Apache Jena Fuseki	29
3	Semantic models	31
3.1	Social Models	31
3.1.1	Extracting data with GSICrawler	32
3.1.2	Sentiment analysis with Senpy	33
3.1.2.1	Senpy Annotations	34
3.1.2.2	SLIWC - LIWC dimensions represented as a SKOS taxonomy	35
3.1.2.3	Morality - MFT concepts represented as a SKOS taxonomy	38
3.1.2.4	Onyx Ontology	39

3.2	Software Models	40
3.2.1	OSLC Core	40
3.2.1.1	Service Provider	45
3.2.1.2	OSLC Resources	46
3.2.2	OSLC Automation Specification	47
3.2.3	OSLC Events and Actions	49
3.2.4	Tracked Resource Set Specification	52
3.2.5	Stackstorm-OSLC semantic model	53
4	Architecture	57
4.1	Overview of the Architecture	58
4.2	Architecture of StackStorm OSLC Adapter	62
4.3	Architecture of Social Monitoring	64
5	Prototype implementation	67
5.1	Scenario deployment	68
5.2	Stackstorm OSLC Adapter Case Study	72
5.2.1	Monitoring module	74
5.2.2	Graph Manager	75
5.2.3	Kafka instance	80
5.3	Social metrics Case Study	81
5.3.1	Airflow	81
5.3.2	GSICrawler	82
5.3.3	Preprocessing	84

5.3.4	Senpy	85
5.3.5	Data Storage and Visualization	86
6	Conclusions and Future Work	89
6.1	Conclusions	89
6.2	Achieved Goals	90
6.3	Future Work	91
A	Project Impact	93
A.1	Introduction	93
A.2	Social Impact	93
A.3	Economic Impact	94
A.4	Environmental Impact	94
A.5	Ethical Implications	94
B	Project Budget	97
B.1	Introduction	97
B.2	Human Resources	97
B.3	Physical Resources	98
B.4	Total Costs	99
C	Stackstorm ontology	101
	Bibliography	105

List of Figures

2.1	Example of DevOps lifecycle	8
2.2	Stackstorm Architecture	10
2.3	Apache Airflow Architecture	12
2.4	Apache Airflow label management example	12
2.5	MongoDB Architecture	13
2.6	Example Apache Kafka event diagram	15
2.7	Partitions in Kafka topics	16
2.8	Elasticsearch Architecture Example	17
2.9	Elastic Stack Architecture	18
2.10	Logstash Architecture Example	19
2.11	Linked Data principles	22
2.12	RDF Structure Description	23
2.13	RDF Graph Example	23
2.14	Protégé interface example	25
2.15	GSICrawler Web Interface	26
2.16	Senpy Architecture	27
2.17	Senpy Web Interface	28
2.18	Apache Jena Diagram	29

3.1	Senpy Annotations Ontology Diagram	35
3.2	SLIWC Ontology Diagram	37
3.3	Morality Ontology Diagram	38
3.4	Onyx Ontology Diagram	39
3.5	OSLC integration diagram	41
3.6	OSLC domains diagram example	42
3.7	OSLC Core Concepts and Relationships	44
3.8	Relationships between Automation Resources	48
3.9	OSLC Actions domain	51
3.10	OSLC Events domain	51
3.11	Stackstorm-OSLC model	54
3.12	Stackstorm Rule Semantic Model	55
3.13	Stackstorm Actions - OSLC Semantic Model	56
4.1	SmartDevOps platform general architecture diagram	58
4.2	SmartDevOps OSLC Layer	59
4.3	Stackstorm-OSLC General Diagram	60
4.4	Stackstorm OSLC Specific Adapter Diagram	63
4.5	Social Metrics Pipeline	64
5.1	Stackstorm MongoDB Architecture	68
5.2	Stackstorm OSLC adapter modules	72
5.3	Stackstorm-OSLC endpoints	73
5.4	Monitoring module diagram	74

5.5	Graph Manager Stackstorm-OSLC	76
5.6	Graph Manager interaction with Kafka	78
5.7	Graph Manager ST2Logs endpoint	79
5.8	Kafka Producer/Consumer Diagram	80
5.9	Pipeline Flowchart	82
5.10	GSICrawler input-output example	83
5.11	Data Preprocessing Example	84
5.12	Output example with Senpy and LIWC, MFT plugins	86
5.13	Kibana Dashboard with number of tuits and timeline for hashtags	87
5.14	Kibana Dashboard with popular hashtags	87
5.15	Kibana Dashboard with Sentiment Analysis	88
5.16	Kibana Dashboard with Data Location	88
5.17	SPARQL Endpoint	88

Listings

2.1	N3 Representation Example	24
3.1	Code for mapping tuits into schema ontology	33
3.2	Example of OSLC Service Provider representation in N3	46
3.3	Example of OSLC Resources List by make a HTTP GET into a queryBase	46
3.4	Example of OSLC Automation Result representation in N3 notation	49
3.5	Example of a changeLog in TRS in N3	53
5.1	Part of code from the DockerCompose file for MongoDB ReplicaSet deployment.	69
5.2	Script for the creation of a Replica Set of MongoDB.	70
5.3	Part of code from the DockerCompose file for KAFKA deployment.	71
5.4	Example of an update rule log from the Stackstorm MongoDB	74
5.5	Stackstorm representation as a OSLC Service Provider	76
5.6	Stackstorm rules representation as a OSLC Resources	77
5.7	OSLC Action representation in N3 for updating a Stackstorm rule	78
5.8	SPARQL Query for extracting the OSLC Action type	79
5.9	DAG code example for a daily execution	81

Introduction

In this section, the context, objectives of the project and an introduction to the developed system will be related.

1.1 Context

In recent years, technology companies, due to the increased demand for features and services by users, have invested much of their activity in improving communication between their two main areas: development and operation. Therefore, in order to maintain the quality of service, the use of DevOps methodology is increasing. These methodologies or sets of practices consist of the use of certain tools that are capable of automatically and easily interconnecting all parts of the life cycle of a software product [1]. Therefore, DevOps tools are becoming a fundamental pillar of the activity of these technological companies. This increase brings as a consequence a set of challenges for these technology companies, which can range from achieving interoperability of these tools to maintaining proper monitoring of them to avoid catastrophes or unexpected results.

On the one hand, one of the main challenges that a company may face when interconnecting different DevOps tools is that each of them may have their own characteristics that make compatibility with the others difficult, either because they are from different manufacturers or because they have incompatible configurations. Therefore, the use of open standards that allow their interoperability, regardless of the manufacturer or the configurations they have, could be an important step forward. This project proposes the use of OSLC (Open Services for Lifecycle Collaboration) as an open standard based on the semantic web and Linked Data principles as a common vocabulary for these tools. This will allow to set aside the limitations mentioned above, as well as establish a way to be able to include other DevOps tools in the future without the need for detrimental adjustments to system configurations. Therefore, this project will propose an architecture based on the use of the OSLC standard and the semantic web for the interoperability of DevOps tools.

On the other hand, a bad monitoring of these tools could lead to an incorrect reaction to unexpected events. This could have unpredictable consequences in terms of both security and system performance. For this reason, it is necessary to create a monitoring framework capable of reacting to unexpected events. Therefore, this project will propose the design and development of a monitoring framework based on OSLC events and actions that will be able to export different types of metrics. Firstly, the monitoring of software metrics will consist of the design and development of an OSLC adapter for the DevOps Stackstorm tool whose function will be, under the principles of Linked Data, to monitor all the events produced in this tool. On the other hand, this framework will also be able to extract and analyze social metrics through social networks such as Twitter. This will be done by deploying a pipeline that will extract from Twitter information related to certain software, apply sentiment and emotion analysis algorithms, as well as other types of operations and expose it to the user through a Dashboard. The advantage of the framework being able to export both types of metrics will give a more global vision of the system's tools. On the one hand, it will be possible to monitor the events of the tool and on the other hand, it will be possible to complete this information with the feedback of the users with respect to a certain service or functionality. a given service or functionality.

In conclusion, this project proposes a way to develop a monitoring framework based on the OSLC standard and the advantages of Linked Data that responds to the new challenges arising from the increased use of DevOps methodologies, through the ability to export social metrics and software metrics of a given DevOps tool.

1.2 Motivation

This project is part of the SmartDevOps¹ project, a collaboration between the Intelligent Systems Group (UPM) and Taiger² for the integration of DevOps tools with semantic technologies.

Modern methodologies in technology companies require the use of tools and practices that automate the entire software development process. The objective of these methodologies is usually to cover the updates and evolution of the technological infrastructure. It is in this context that the use of DevOps tools (acronyms for "Development" and "Operations") becomes one of the most used methodologies to pursue the effective production of high-quality software systems. These tools are very useful, especially in Big Data systems, because of their ability to unite the two fields of development and operations needed to deal with large amounts of data. This is why most Big Data systems today are based on a type of DevOps methodology. Despite the increase in their use, these types of system have certain limitations when each of the tools used has its configurations, which are not always fully compatible with each other. Therefore, the use of an open standard such as OSLC (Open Services for Lifecycle Collaboration) that is common to all of them can free the system from this limitation based on the benefits of Linked Data.

On the one hand, adding a monitoring layer to the whole process is very beneficial for several reasons. This monitoring framework will be comprehensive and will have several benefits. On the one hand, the main one is that it allows us to react much faster to unexpected changes and allows administrators to manage these changes by collecting software metrics. These metrics consist of all the events that occur in the different DevOps tools that make up the system, to have a complete view of the events that occur in the system. On the other hand, this monitoring framework will be able to extract, analyze social metrics through social networks (mainly Twitter), and expose them through a dashboard easy to understand by users or developers.

In conclusion, the main motivation of this project will be to design and develop a monitoring framework based on the semantic web and the OSLC standard to allow integration of DevOps tools through the use of a common vocabulary. This framework will be able to monitor software metrics of the DevOps environment by monitoring all events produced by a DevOps tool and it will be able to monitor social metrics through Twitter.

¹<https://smartdevops.gsi.upm.es>

²<https://taiger.com>

1.3 Project goals

On the one hand, the OSLC adapter must be able to provide a monitoring layer to manage the events that occur in Stackstorm, as well as to detect unexpected changes in its operation. To do so, it must be able to obtain software metrics through monitoring based on OSLC actions and events.

On the other hand, this framework will be able to obtain social metrics by extracting real data from Twitter, related to a certain software or service. These metrics will be received by the analysis layer, which will be in charge of applying sentiment and emotion analysis algorithms, using semantic ontologies, to show in detail the user feedback to that software or service. A visualization layer will be added through which the user can see the result of all the operations performed on these social metrics. All this will be carried out through a pipeline formed by a set of software services that will allow the automatic and sequential execution of each of the tasks related to the extraction, analysis and visualization of social metrics.

In addition, the project will aim to enable the integration of different DevOps tools through the use of an open and common standard. In this case, OSLC (Open Services for Lifecycle Collaboration) will be used to overcome the limitations[2] that each DevOps tool that is part of the system may have, either because it comes from a different manufacturer or because it has specific configurations that make it incompatible with the others.

In conclusion, the main objective of the project can be summarized in the following points:

- Design and development of a Stackstorm-OSLC adapter that allows the monitoring of all the events produced in it. In this project the events will consist of the modification (activation/deactivation), deletion or creation of Stackstorm rules.
- Design of an ontology to semantically model the Stackstorm rules to adapt it to OSLC concepts.
- Design and development of a module for the extraction and analysis of social metrics through Twitter.
- Design of an architecture that allows the integration of tools in a DevOps environment in the future, under the use of the OSLC standard and distributed communication through Kafka.

1.4 Structure of this document

The remainder of this document is structured as follows.

Chapter 2: *Enabling Technologies*. This chapter will explain the context on which this project is based, by explaining the most important technologies and tools used in the development of the system.

Chapter 3: *Semantic Model*. This chapter explains the entire semantic layer that is part of the project. On the one hand, all the ontologies used for the development of the social metrics extraction and analysis pipeline will be explained, as well as the software models such as OSLC with its different domains and classes, and the model created to semantically define the Stackstorm rules.

Chapter 4: *Architecture*. This chapter will explain the entire architecture of the project. All the modules involved in the Stackstorm-OSLC adapter and its monitoring framework, as well as all the software used for the social metrics pipeline and its interconnection, will be explained.

Chapter 5: *Prototype implementation*. This chapter will explain all the details of the technical implementation of the different modules for monitoring both the software metrics and the social metrics. This chapter does not intend to describe all the code used, but to explain to the reader the methodology used and the development carried out for each part of the system.

Chapter 6: *Conclusions* This chapter will explain all conclusions and future ideas to extend and improve this work in the future.

Appendixes Here we will document the semantic model to define the Stackstorm rules according to the principles of the Semantic Web and Linked Data. Here, different impacts of the project, such as ethical or environmental impact, will be explained.

Enabling Technologies

This section will explain the main concepts necessary to understand the development of the project, as well as the specific tools and services used for the project. First, the general technologies necessary to understand the project will be described. These general technologies can be separated into two; DevOps and Semantic Web. These general technologies will have as a subsection the respective services and technologies that will be used in the project and that can be encompassed within them. Of the tools discussed in the following, not all will have the same function. That is, some of them will be part of the pipeline that will allow the project to extract social metrics for subsequent analysis, indexing, and visualization. Others, such as StackStorm, will be a major part of the OSLC standard and its integration with other tools through the use of Linked Data.

Before we begin to describe the DevOps technologies used in the project, it is necessary to frame this project within the framework of DevOps practices and methodologies. This overview will serve to understand not only the purpose of the project but also the usefulness of using the specific services that will be described. It is also necessary to explain the importance of Linked Data and how its benefits form an important part of the usefulness of the project, as well as its own development.

2.1 Development and Operations (DevOps)

The DevOps concept will be vital for understanding the project. DevOps tools have become a mainstay of technology companies. The use of these tools leads to more fluid ongoing communication, collaboration, integration, visibility, and transparency between the two main camps in these companies; development teams (Dev) and their counterparts in technology operations (Ops). This seamless communication between the two domains brings many benefits to the user, who can receive the service in significantly less time. With the increase in user demand for new services and functionalities, this benefit becomes one of the key benefits in today's technology companies' lifecycle. DevOps concept corresponds to the combination of the words "Development" and "Operations" and refers to a set of practices whose objective is to streamline processes so that they move more smoothly from the creation of a functionality or software improvement (development) to the deployment in a production environment (operation) so that it can be used by the user. Therefore, the seamless relationship between the two fields enables faster delivery of value to the user.

DevOps methodology includes several phases of the software product lifecycle, and each of them will have several characteristic tools with specific functionalities. [3]

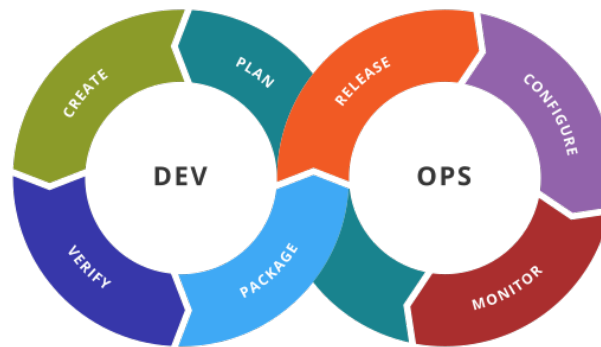


Figure 2.1: Example of DevOps lifecycle

DevOps methodology can be applied to very different delivery models but must be adapted to the environment and the product architecture[4]. This requires a fairly high integration capability of the different tools that address specific functionalities at each stage. For example, in the case of highly secure cloud-based delivery, these delivery models need specific architecture and hardware changes.

There are several methodologies involved in DevOps such as Infrastructure as Code, Continuous Integration, and Continuous Deployment (CI/CD), Cloud-Native DevOps, Microservices with DevOps, and DevSecOps. It is an evolving field that uses various open source and enterprise tools. With the growing popularity of cloud and SaaS services, DevOps methodologies are available as a managed service from the cloud provider. These DevOps methodologies cover all phases of the lifecycle of a software project, which will be important to introduce to understand the need for this project and the advantage of integrating a DevOps tool with the semantic web. These phases are: [5]

1. **Project Tasks Planning:** This mainly involves the project management aspects of the software life cycle.
2. **Development:** This includes the development of the code, as well as the collaboration of the development team in it.
3. **Building the Code:** Consists of transforming the code into an executable and deployable element.
4. **Deploy the Code:** The deployment in the test or production server, which is the project or software execution environment.
5. **Test the Deployed Resources:** Testing for the quality assurance of the code deployed on the test servers.
6. **Package the artifacts:** There may be a phase involved that packages deployable artifacts ready for production that can be shared through various channels.
7. **Version-Release Cycle:** This includes the lifecycle of enhancements and new releases.
8. **Scaling and Upgrading:** This involves performance optimization and infrastructure scaling for load balancing of deployable artifacts or services.
9. **Support and Maintenance of the Services:** This involves production support and monitoring, problem escalation, problem fixing, and resolutions for services or applications.

As mentioned above, each phase of a product life cycle will have different tools or services with functionalities specific to that phase. In this project, event-based task automation tools, such as StackStorm or ELK-based tools, will be used for information visualization or indexing (Elasticsearch and Kibana).

2.1.1 StackStorm

StackStorm is a platform for the integration and automation of services and tools[6]. Is event-driven automation framed within Infrastructure as Code (IaC). This means that it can deal with infrastructure management and provision through code rather than manual processes[7]. That is, all parts of StackStorm can be programmed and configurable through programming languages such as Python or through.YAML files. Stackstorm has a modular architecture that allows its components to communicate via the message bus. The main architecture of this tool can be seen in the figure below.

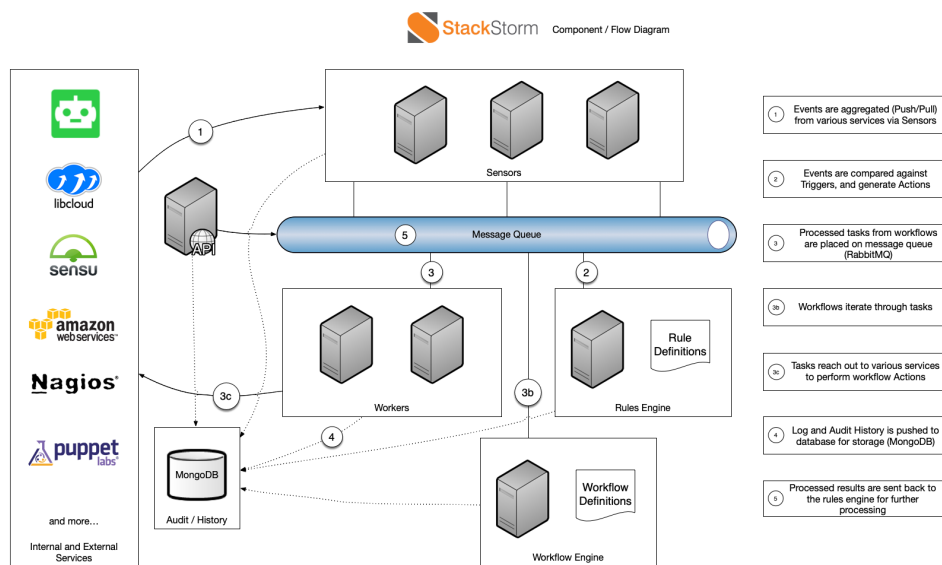


Figure 2.2: Stackstorm Architecture

The complete architecture consists of other components, such as **triggers**, **workflows**, **packs**, **actions**, **rules** and **audit trail**. The components of interest for the project will be explained below[6]:

- **Triggers:** StackStorm representations of external events. These triggers can be a consequence of generic events, such as webhooks or timers, or triggers associated with external tools. Custom triggers can also be defined through the Python development of the plugin for a given sensor.

- **Actions:** StackStorm outbound integrations. These actions can be generic (REST calls, ssh, etc.), integrations with other tools (OpenStack, Docker, etc.), or custom actions. These actions can be invoked manually by the user via the CLI or API, or used and called part of rules.
- **Rules:** They associate triggers with actions by applying matching criteria. When a trigger occurs as a result of an event (external or internal), a certain action is triggered.
- **Audit trail:** Stackstorm's architecture has a way of recording and storing the details of what the tool produces. This log output allows auditing by a client or an external administrator to prevent anomalous behavior. This audit trail allows integration with other visualization tools such as Logstash, which will be very useful for the project.

Stackstorm will have fundamental relevance in the project since it will be the DevOps tool whose events will be monitored by the monitoring framework.

2.1.2 Airflow

Airflow is an Apache platform to create, schedule, and monitor workflows on a schedule basis. It is a way to automate the execution of workflows in a scalable, dynamic, and extensible way. It is fully written in Python and one of its main features is the integrations it has with other tools with Microsoft Azure or Amazon Web Services.

As it is an open source platform, it can be perfectly adapted to the needs or characteristics of the architecture [8]. Airflow creates workflows as **directed acyclic networks (DAGs)** of tasks, and the scheduler is responsible for executing the tasks on multiple workers following the specified dependencies. A DAG (Directed Acyclic Graph) is a data circuit defined in Python code. Each DAG represents a sequence of tasks to be executed organized to indicate the relationships between tasks in the Airflow user interface. These DAGs can be as complex as the administrator wants. Airflow can be used for any batch data circuit, so its use cases are as diverse as they are diverse.

Due to its extensibility, this platform is particularly well suited for organizing tasks with complex dependencies on multiple external systems. By writing circuits in code and using the various available plugins, Airflow can be integrated into any dependent system from a unified platform for management and monitoring[9]

An Airflow installation generally consists of the following components: [10]

- **Scheduler:** Handles both triggering scheduled workflows, and submitting Tasks to the executor to run.
- **Executor:** Handles running tasks. Most production-suitable executors push the execution of tasks out to workers.
- **Web Server:** User interface to inspect, trigger, and debug the behavior of DAGs and tasks.
- **Metadata Database:** Used by the scheduler, executor, and web server to store the state.

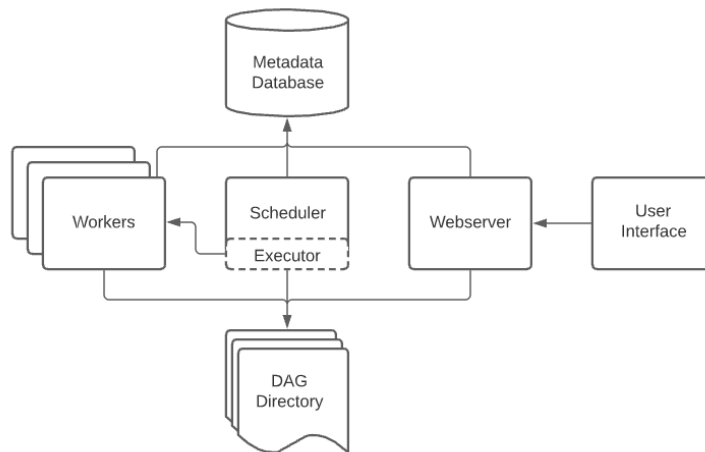


Figure 2.3: Apache Airflow Architecture

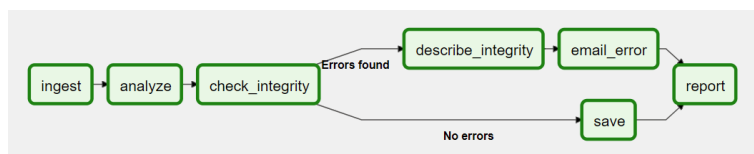


Figure 2.4: Apache Airflow label management example

In the case of this project, Apache Airflow will be in charge of defining tasks and producing their continuous execution depending on the context. When workflows are defined as code, they become more maintainable, versionable, testable, and collaborative. In the case of this project, it will be in charge of interrelating the social metrics gathering task, the pre-processing task, the sentiment analysis task, the indexing task, and finally the visualization task.

2.1.3 MongoDB

MongoDB is a document database, distributed at its core and free to use, offering an advanced query and indexing model. It offers drivers for more than 10 languages, of which we will be interested in its development in Python. The MongoDB document model offers greater scalability and flexibility, which is an advantage for processing large amounts of data. Documents stored in MongoDB are similar to JSON, as fields can vary between documents, and the data structure can be changed over time to suit each use case. Queries, indexing, and real-time aggregation offer several advantages when working with these data.[11]

MongoDB allows for horizontal scaling. This means that the information can be spread across several servers so that each server has a part of the complete data set. This process is called sharding. The set of servers that contains the entire data set is called a sharded cluster. To ensure high availability of the cluster, each shard is configured as a replicated cluster. This ensures the fault tolerance of each shard separately, regardless of which shard stores a particular piece of data. When a query is made on a fragmented set, it must go through a routing process that indicates where to look for the data within the set. [12]

Therefore an example of the basic architecture of MongoDB in the project could be:

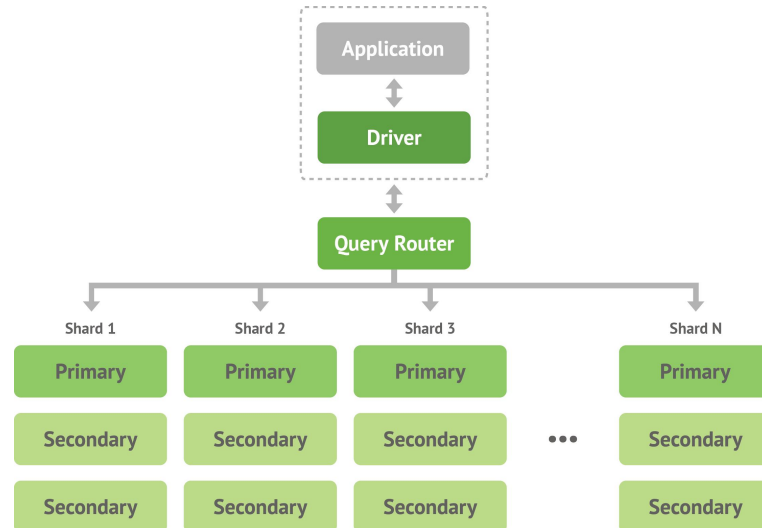


Figure 2.5: MongoDB Architecture

The main advantages of MongoDB are: [13]

- **Availability:** Replication capacity in the face of errors and self-healing recovery.
- **Scalability:** This allows horizontal scalability to increase the number of nodes de-

pending on the data load demanded by the user or application.

- **Workload isolation:** Ability to run operational and analytical workloads in the same cluster.
- **Locality:** Ability to place data on specific devices and in specific geographies for governance, class of service, and low-latency access.

Another interesting aspect that will be used in this project will be the ability of MongoDB to perform replication. Replication allows MongoDB to store the same data in different instances, so that if one of the nodes becomes corrupted, the system will be able to access any of the other nodes. In this project, a replica set will be managed to obtain this decentralization. The importance of this database in the project lies in the fact that, as previously mentioned, the logs of the events and actions produced by/in Stackstorm are stored in MongoDB. Therefore, it will be useful to understand how this works, as well as the development in Python of an active monitoring layer of MongoDB, since in this way we will be able to actively monitor Stackstorm.

2.1.4 Apache Kafka

Apache Kafka is an open-source distributed event streaming platform used for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.[14].

Is a distributed system consisting of servers and clients communicating over a high-performance TCP network protocol[15].

- **Servers:** Kafka runs as a cluster of one or more servers that may span multiple data centres or cloud regions. Some of these servers form the storage layer, called brokers. Other servers run Kafka Connect to continuously import and export data as event streams to integrate Kafka with their existing systems, such as relational databases, as well as with other Kafka clusters. It is highly scalable and faults tolerant: if any of your servers fail, the other servers will take over their work to ensure continuous operations without any data loss.
- **Clients:** Enable you to write distributed applications and microservices that read, write and process event streams in parallel, at scale and in a fault-tolerant manner, even in the event of network problems or machine failures. Kafka comes with some of these clients included, which are augmented by dozens of clients provided by the Kafka community.

An event records the fact that "something happened". It is also called a record or message in documentation. When data is read or written in Kafka, it is done in the form of events. Conceptually, an event has a key, a value, a timestamp and optional metadata headers. Producers are those client applications that publish (write) events to Kafka, and consumers are those that subscribe to (read and process) these events. In Kafka, producers and consumers are completely decoupled from each other, which is key to achieving the high scalability for which it is known.

Events are durably organized and stored in topics. Very simplistically, a topic is similar to a folder in a file system, and events are the files in that folder. Topics in Kafka are always multi-producer and multi-subscriber: a topic can have zero, one, or many producers writing events to it, as well as zero, one, or many consumers subscribing to these events. Events in a topic can be read as many times as necessary; unlike traditional messaging systems, events are not deleted after consumption.

Topics are partitioned, meaning that a topic is spread across a number of "buckets" located in different Kafka brokers. This distributed placement of data is very important for scalability, as it allows client applications to read and write data to/from many brokers at the same time. When a new event is published to a topic, it is actually added to one of the topic partitions. Events with the same event key are written to the same partition, and Kafka guarantees that any consumer of a given topic partition will always read events from that partition in exactly the same order in which they were written.

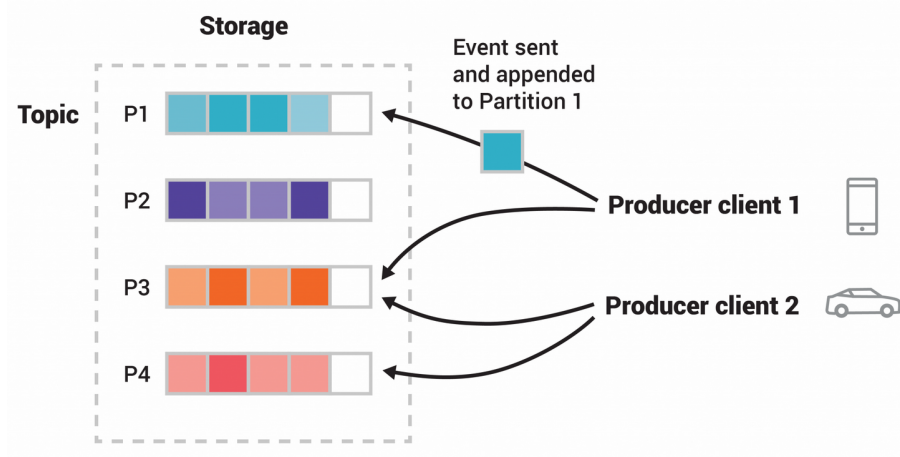


Figure 2.6: Example Apache Kafka event diagram

For the project it will be of vital importance since it will be the tool in charge of receiving OSLC Events from the adapter, as well as producing OSLC Actions to all those DevOps tools that are subscribed to a certain topic.

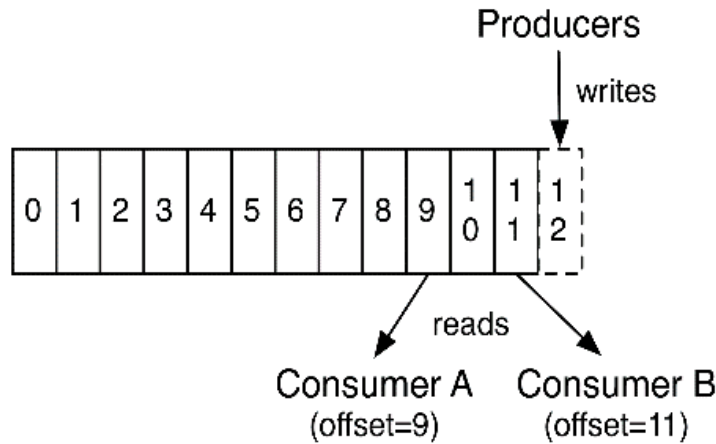


Figure 2.7: Partitions in Kafka topics

2.1.5 The Elastic Stack

Elastic Stack is an Open Source solution commonly used for data analysis. Elastic Stack is an Open Source solution commonly used for data analysis. It consists of a series of software solutions that reliably and securely takes data in any format and, after searching and analyzing them, allows visualization of the data in real time. It is also known as ELK for its main solutions: Elasticsearch, Logstash and Kibana[16]. These solutions will be of vital importance in the data ingestion and processing pipeline that is part of the project.

2.1.5.1 Elasticsearch

It is the core of the Elastic Stack. Elasticsearch is a distributed RESTful search and analytics engine that stores data centrally. The way Elasticsearch works starts with ingesting data from a variety of sources, including logs, system metrics, and web applications.[17] Data ingestion is the process by which these data are analyzed, normalized, and enriched before being indexed in Elasticsearch. Once indexed in Elasticsearch, users can perform complex queries on their data using aggregation to retrieve complex summaries of their data. Elasticsearch aggregations allow you to get a more general view of exploring trends and patterns in your data, using custom HTTP queries to navigate between indexes.

Elasticsearch allows you to perform and combine structured, unstructured, geographic, metric, and other searches. Elasticsearch works through the use of clusters and nodes. Elasticsearch is deployed in a cluster that must have at least 1 node. A node is a physical

or virtual server on which an instance of the Elasticsearch service is running and that is part of the cluster[12] The data stored in Elasticsearch are divided among the nodes of a cluster, thus distributing the load. As the data are divided among the nodes, when we query Elasticsearch for some data, the query is executed in parallel on all nodes.

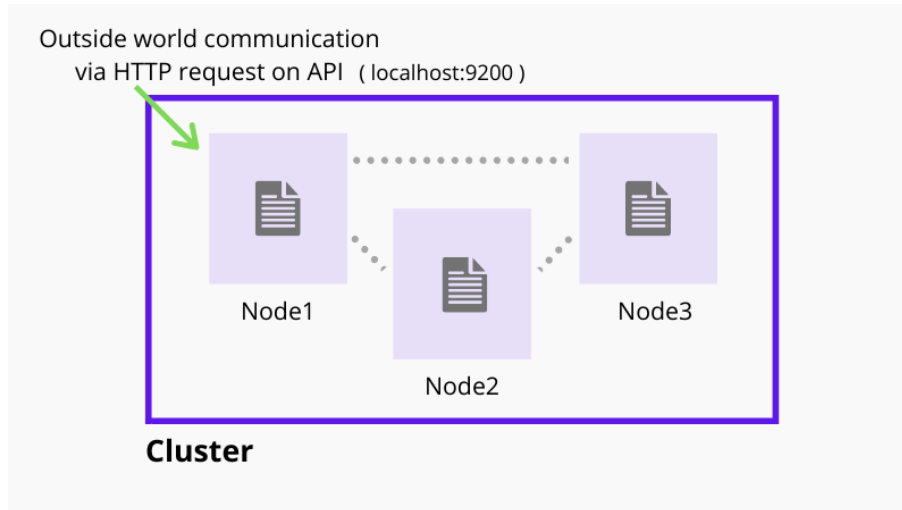


Figure 2.8: Elasticsearch Architecture Example

Therefore, a cluster is nothing more than a group of nodes among which all data or the cluster is divided[18]. These nodes, apart from storing data in a distributed manner, will have different roles/tasks. In an Elasticsearch cluster, some of the different nodes can be found:

- **Data nodes:** Used to store data and perform data-related operations, such as search queries, etc.
- **Master nodes:** Responsible for managing the entire cluster and configuration actions, such as adding and deleting nodes.
- **Ingest nodes:** In charge of first receiving incoming data, where we can preprocess the documents before indexing them.

These data, indexed and distributed among the different nodes that form the Elasticsearch cluster, can be obtained and visualized by Kibana (another ELK solution), explained below.

2.1.5.2 Kibana

Kibana is an open user interface that allows you to visualize Elasticsearch data in multiple ways and navigate between the different solutions in the Elastic Stack, such as Logstash.[19] One of the main features of Kibana is the ability to build dashboards in an intuitive and user-friendly way, as well as the ability to represent the data stored in Elasticsearch in multiple ways.

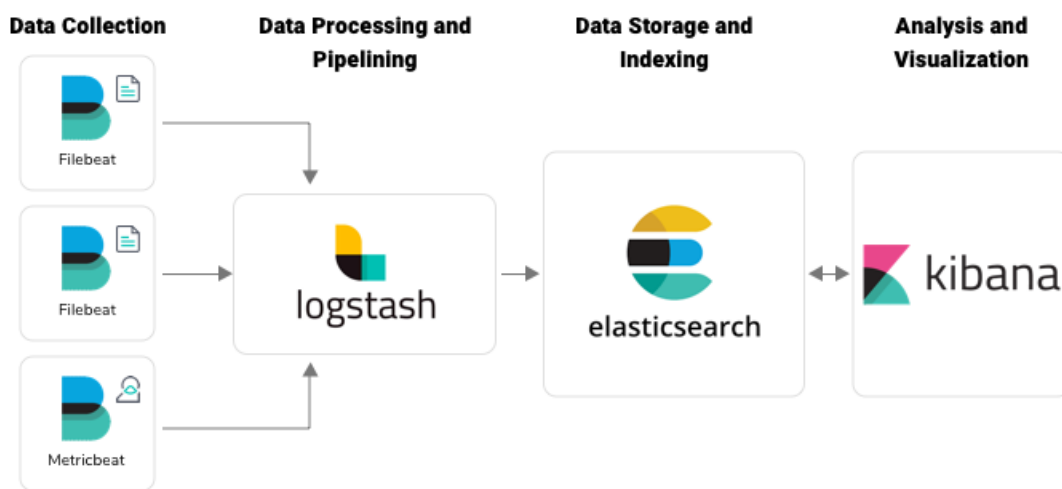


Figure 2.9: Elastic Stack Architecture

In this project we have chosen to develop the social metrics visualization layer with Kibana because of its simplicity of management and its integration with Elasticsearch. Kibana allows the creation of different graphs, which can range from simple data visualizations to complex visualizations that mix different data.

2.1.5.3 Logstash

Logstash is an open server-side data processing pipeline that ingests data from a multitude of sources, transforms them, and sends them out. Generally, data often come in different formats and from different sources. A great advantage of Logstash is that it supports a wide variety of inputs to extract events from different sources at the same time. Once data are received from a given source, Logstash parses each event, identifies the fields, and sends them in a structured form based on the system's interests. This is very useful for managing

logs in a common, and therefore more efficient way. Likewise, it allows sending the data received to different databases or indexing engines[20] as shown in the following diagram:

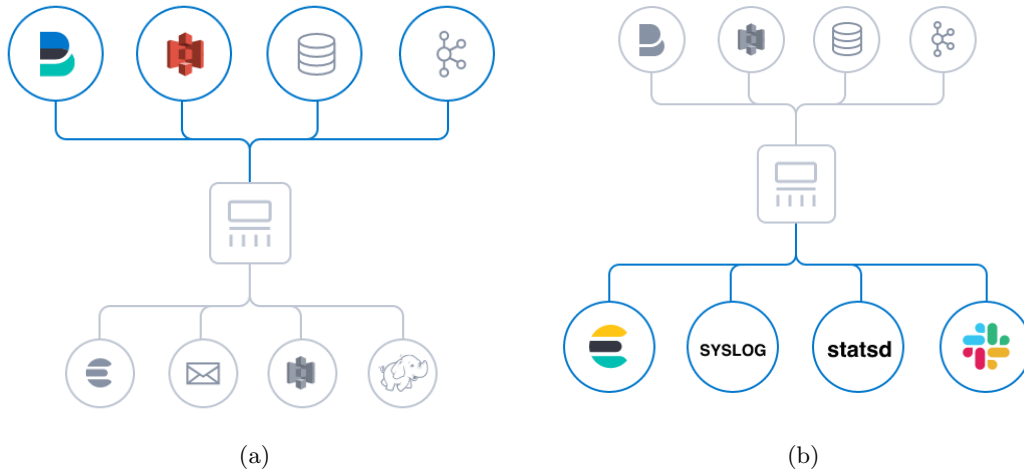


Figure 2.10: Logstash Architecture Example

2.2 Semantic Web

A key part of the project is the Semantic Web and Linked Data. In this case, the Linked Data principles will be used to design and develop a StackStorm adapter to the open standard OSLC (Open Services for Lifecycle Collaboration). Before explaining this standard in detail, it is necessary to explain the fundamentals and main concepts of the Semantic Web, as well as the working principles defined by Linked Data.

The **Semantic Web** is an extension of the Web created by Tim Berners-Lee whose main objective is the creation of technologies that make it possible to publish data readable by computer applications (machines, in the terminology of the semantic Web)[21]. It is also known as Web of Data instead of Web of Document. It consists of adding semantic metadata to the World Wide Web that provides additional information about the content, meaning, and relationship between these data. These metadata in the Semantic Web are data about data that are capable of bringing meaning to data. It could be said that just as the basis of the Web is data, the basis of the Semantic Web is metadata. These metadata must be provided formally so that they are easily readable by machines. It is precisely this advantage that facilitates inter-operability between such machines, since adding this metadata to web data allows one to assign a particular machine-readable meaning, independent of their

configurations. These machines use "intelligent agents" to do this. Intelligent agents are programs that search for information and process it. In this way, computers can interpret the meaning of web data similarly to humans. In short, it is about adding meaning to the data on the Web so that they can be readable and understandable by the machine without the need for human interaction.

The way the Semantic Web defines concepts, to give meaning to data on the Web, is through **Ontologies**. An ontology in the semantic web is a way of describing an entity. This entity can be any entity; a person, a place, a book, etc. These ontologies are developed using **OWL (W3C Web Ontology Language)**. OWL is a Semantic Web language designed to represent complex knowledge about things, groups of things, and relationships between things. It is based on computational logic, so it can be exploited by computer programs[22]. To understand the power of OWL, it is necessary to understand the concept of class, subclass, and property defined by the Resource Definition Framework Schema (RDFS). The most important concepts are explained below:[23]

- **Class:** A class defines a group of individuals who belong to the same class because they share some properties.
- **rdfs:subClassOf:** Class hierarchies must be created by giving one or more indications that one class is a subclass of another.
- **rdfs:Property:** Properties can be used to establish relationships between individuals or from individuals to data values.
- **rdfs:subPropertyOf:** Property hierarchies can be created by giving one or more indications that a property is itself a subproperty of one or more other properties.
- **rdfs:domain:** A property domain narrows the individuals to whom the property can be applied.
- **rdfs:range:** The rank of a property reduces the number of individuals that a property can have as its value.

These developed ontologies can be published on the World Wide Web and can be referenced or referred to other ontologies to interconnect all this knowledge. These ontologies must be previously agreed upon since one of the advantages of the Semantic Web is the interoperability between machines, thanks to the use of a common vocabulary. Both RDF and its main bases will be explained in later sections.

Note that the elements mentioned above are part of a much larger catalog of features that make up OWL. The description of these terms will serve as the basis for the subsequent description of the semantic model developed in the project.

2.2.1 Linked Data

Linked Data is one of the core pillars of the Semantic Web. It presents a method for publishing interlinked structured data. It can also be understood as a set of best practices for publishing and connecting structured data on the Web of Data[24] The following technologies can be defined as fundamental pillars of Linked Data:

- **URIs:** Generic mechanism to identify resources and concepts. Its main advantage is that they are global and unique.
- **HTTP:** Protocol for managing the URIs of these resources. These resources can be managed, through their URIS, with HTTP requests (GET, PUT, CREATE, DELETE).
- **RDF Data Model:** It is a way of describing the relationship between one or more resources using triples.

On the basis of these technologies, a series of principles were defined that establish the requirements to be met. These principles are as follows.

1. **Use URIs as names for things (resources)**
2. **Use HTTP URIs so that people can look up those names (open data)**
3. **When someone looks up a URI, provide useful RDF information about the resource**
4. **Include RDF statements that link to other URIs so that they can discover related things**

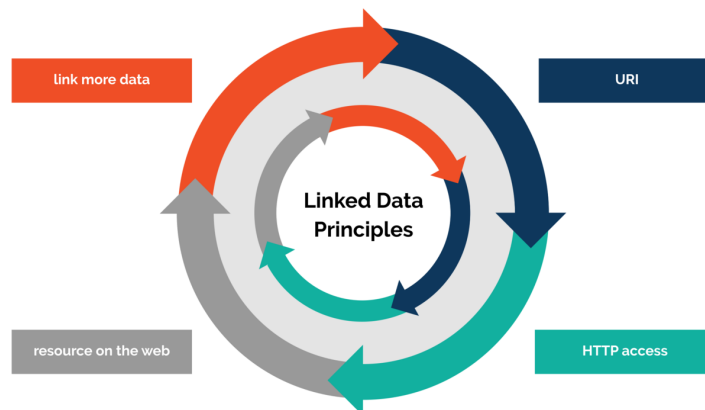


Figure 2.11: Linked Data principles

Following these basic principles, it is possible to obtain a complex network of resources identified with an unequivocal URI, interconnected via HTTP, and accessible through other resources. In this way, the semantic web or web of data is embodied, since each resource or object is represented by RDF/XML and can link or be linked by another resource in an interoperable and open way for machines without the need for human intervention.

2.2.2 RDF (Resource Definition Framework)

The representation of this information is carried out through **RDF (Resource Definition Framework)**. RDF is part of the W3C Semantic Web technology stack, which includes OWL, RDFS, SPARQL, etc. RDF is a data interchange standard that is used to represent interconnected data. How each RDF resource statement is structured is through triples (three positional statements)[25].

These triples consist of a subject, a predicate (or verb), and an object.

- **Subject:** What is described.
- **Predicate:** Property or relationship between resources. It is also called verb.
- **Object:** The value of the property marked by the predicate.

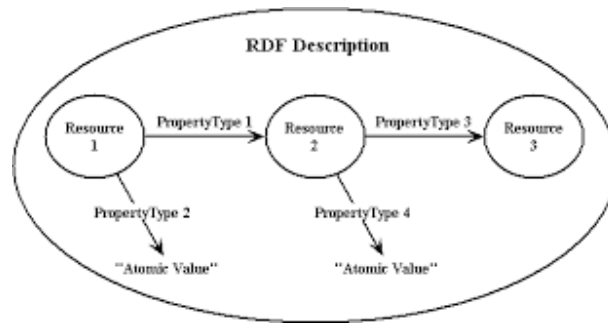


Figure 2.12: RDF Structure Description

In this way, complex and interrelated resource relationships can be built using URIs and ontologies. An example could be:



Figure 2.13: RDF Graph Example

This example shows what the description of a resource would be like through its different properties and values. As can be seen, each resource has its own URI that references the resource for more information about it.

In the example shown, visualization through graphs is used, but because Turtle/N3 is a way of representing information about resources, it allows for representation in an N3-based syntax

Listing 2.1: N3 Representation Example

```
@prefix con: <http://www.w3.org/2000/10/swap/pim/contact#> .

<http://www.w3.org/People/EM/contact#me> a con:Person ;
    con:fullName "Eric Miller" ;
    con:mailbox <mailto:em@w3.org> ;
    con:personalTitle "Dr." .
```

This RDF/XML representation is another way to represent the example in Figure 2.7. Having explained how data are shared on the semantic web and how semantic web resources are described, it is necessary to understand how to interlink resources using Linked Data.

2.2.3 Protégé

For the definition of semantic models, Protégé will be used.

Protégé is a collaborative open source ontology development environment developed by the Center for Biomedical Informatics Research at the Stanford University School of Medicine with the following characteristics[26]:

- Support for OWL ontology editing
- A simple default editing interface, providing access to the most commonly used OWL constructs
- Customizable user interface
- Multiple formats for uploading and downloading ontologies (supported formats: RDF/XML, Turtle, OWL/XML, OBO, etc.).

Allows the creation of a customized ontology model through an interface. In the case of the project, it will be the software used to model the Stackstorm rules, a model that will be explained in detail later in this document. Protégé is based on semantic web concepts like classes, subclasses, etc. In this way, a customized model is developed for your use case, by using other models (with their URIs) using in this way the best practices proposed by the Linked Data principles.

The Protégé interface allows for the visualization of the ontology in the form of a graph, which greatly simplifies the development process, as well as the relationship between classes and subclasses.

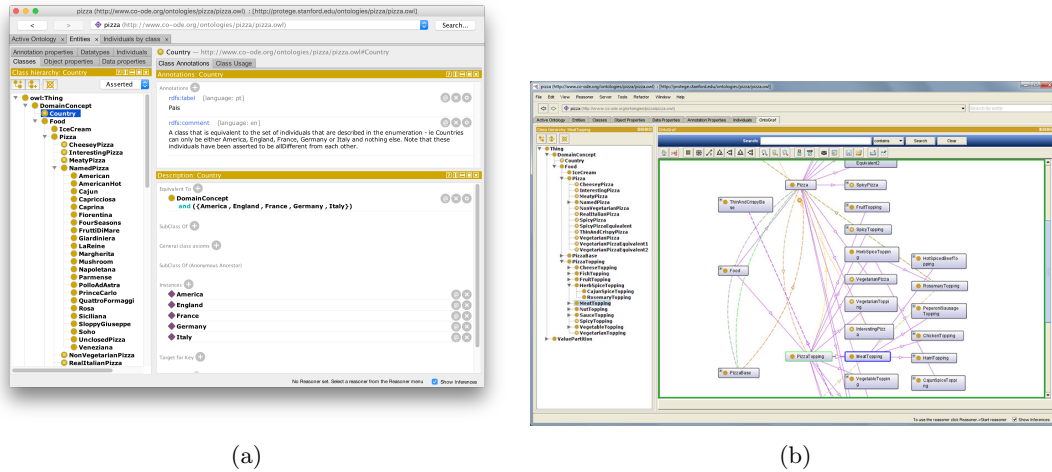


Figure 2.14: Protégé interface example

The idea of using Protégé is to import both the OSLC Core and other domains to implement a semantic model of the Stackstorm rules.

2.2.4 Social metrics and sentiment analysis

As mentioned above, this project will export software metrics (through DevOps tools and their semantic adaptations to OSLC) and social metrics. In the latter case, the following technologies will be included.

Social metrics are understood as those metrics obtained through social sources such as Twitter, Reddit, etc. Once these metrics are obtained, a series of operations will be performed through algorithms and libraries to perform sentiment and emotion analysis of the data obtained. The project will use technologies from the Intelligent Systems Group of the Universidad Politécnica de Madrid. These technologies are included in the Semantic Web section, since they are based on the use of ontologies and semantic models.

2.2.4.1 GSICrawler

GSI Crawler is an innovative and useful framework that aims to extract information from web pages by enriching them using semantic approaches.[27] Of all possible social sources, for this project, we will be particularly interested in Twitter. The user interacts with the tool through a Web interface, selecting the type of analysis to be performed and the platform to be examined ¹. It is developed in Python, which allows the integration of new crawlers

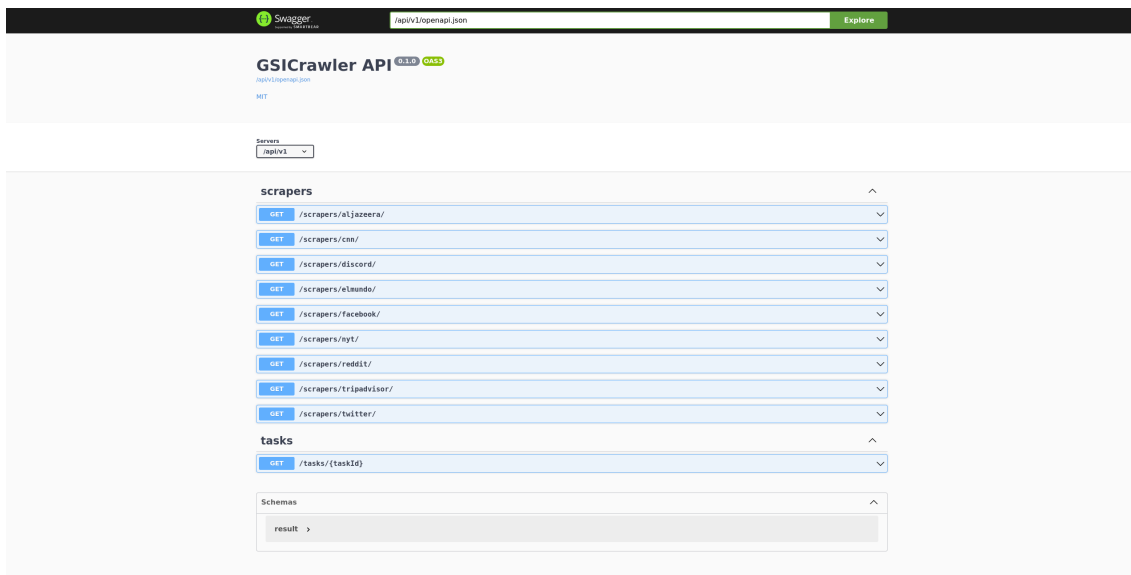


Figure 2.15: GSICrawler Web Interface

or the adaptation of these to a specific use case. The user can interact with them through the endpoint shown in the figure. Web scraping is a technique used by software programs to extract information from websites or to obtain data through the web service API. These software programs are called crawlers.

The GSI Crawler environment can be defined from a high-level point of view as follows:

- **Data ingestion:** This is the main function of GSI Crawler, which consists of extracting data according to the requests sent to it. It works thanks to the use of web crawlers. In this project **snsrape**² will be used as a Twitter crawler.
- **Semantic representation:** Before storage, the data will be enriched according to semantic paradigms to allow for a more powerful analysis later on.

¹<https://crawler.gsi.upm.es/>

²<https://github.com/JustAnotherArchivist/snsrape>

- **Data storage:** After the acquisition and enrichment of the data, the storage process takes place. This storage can be done through Elasticsearch, for later visualization through Kibana or others or Fuseki. The latter will store the RDF triples, which can be consulted through SPARQL.

In this way, different social metrics will be extracted from a given software through Twitter, processed for semantic modeling, sentiment and emotion analysis will be performed with Senpy, and stored in Elasticsearch and Fuseki.

2.2.4.2 Senpy

Senpy is a framework for developing, evaluating and publishing web services for sentiment and emotion analysis in text. The framework is aimed at developers and users. For developers, it is a means of evaluating their classifiers and easily publishing them as web services. For users, it is a way to consume sentiment analysis from different vendors through the same interface. This is achieved through a combination of an API aligned with the use of semantic formats and a set of well-established vocabularies. The framework is open source and has been used extensively in several projects. [28]

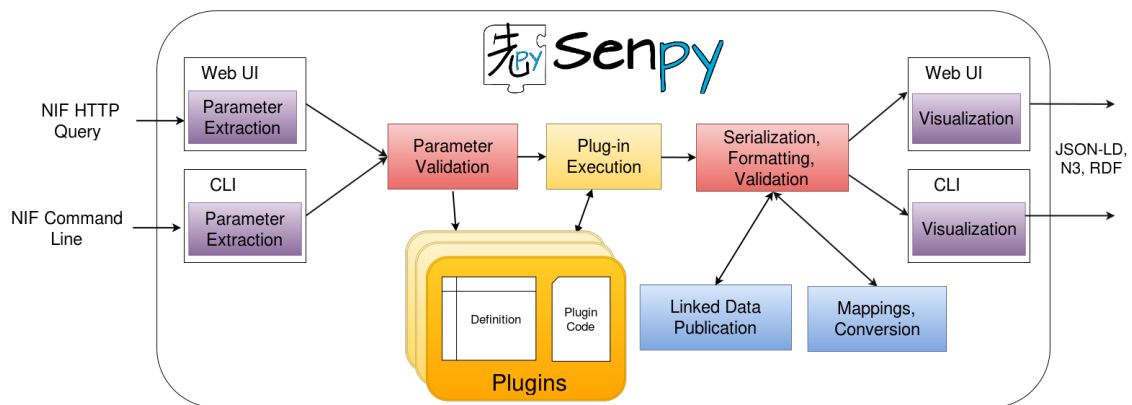
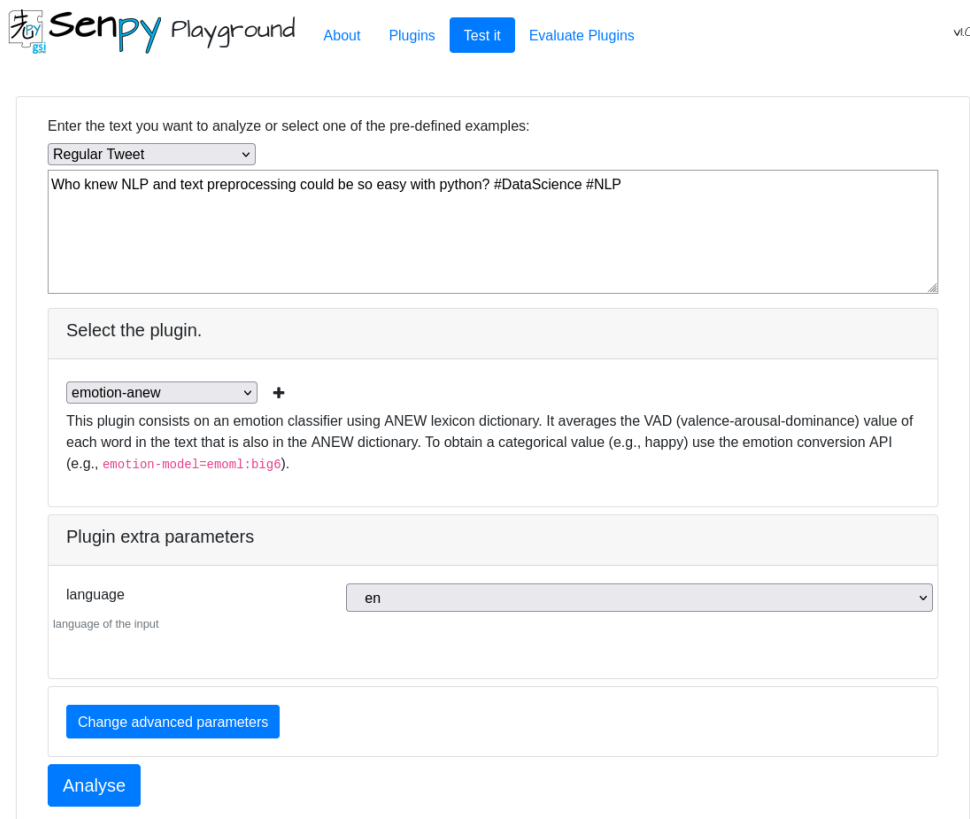


Figure 2.16: Senpy Architecture

All services built with Senpy share a common interface. This allows users to use them interchangeably, with the same API and tools, simply by pointing to a different URL or changing a parameter. The common scheme also makes it easier to evaluate the performance of different algorithms and services. Senpy has a built-in evaluation API that can be used to compare results with different algorithms.[29]

Services can also use the common interface to communicate with each other. In addition, higher-level features, such as automatic result fusion, emotion model conversion, and service discovery, can be built on top of these services.

To achieve this goal, Senpy uses an approach based on Linked Data principles, based on the NIF (NLP Interchange Format) specification, and open vocabularies such as Marl and Onyx. Senpy uses Python-developed plugins that produce one result or another, depending on the dictionaries and algorithms used. Therefore, it allows the user to add the plugins that suit his use case. It has a Web interface that facilitates user operations. The user enters a text, chooses the plugin that he/she wants to apply (depending on the output he/she wants to obtain) and obtains the result of the desired analysis.³



Senpy Playground [About](#) [Plugins](#) [Test it](#) [Evaluate Plugins](#) v1.0.1

Enter the text you want to analyze or select one of the pre-defined examples:

Regular Tweet ▼

Who knew NLP and text preprocessing could be so easy with python? #DataScience #NLP

Select the plugin.

emotion-anew ▼ +

This plugin consists on an emotion classifier using ANEW lexicon dictionary. It averages the VAD (valence-arousal-dominance) value of each word in the text that is also in the ANEW dictionary. To obtain a categorical value (e.g., happy) use the emotion conversion API (e.g., `emotion-model=emoml:big6`).

Plugin extra parameters

language en ▼
language of the input

[Change advanced parameters](#)

[Analyse](#)

Figure 2.17: Senpy Web Interface

It can be observed that in Senpy's semantic modeling, use is made of other ontologies such as Marl, Onyx, or nif. All ontologies used for the project will be explained in detail in Section 3.

³<https://senpy.gsi.upm.es>

2.2.5 Apache Jena Fuseki

Another data storage component of this project will be Apache Jena Fuseki⁴. Fuseki is an SPARQL server[30] that allows the user to perform complex SPARQL searches, through an SPARQL endpoint accessible over HTTP by REST-style interaction with RDF data. Fuseki can be run in the background by an application as an embedded server. The application can safely work with the dataset directly from java while having Fuseki provide SPARQL access over HTTP.

It is part of Apache Jena, which is a free and open source Java framework for building applications of the Semantic Web and Linked Data.[30]

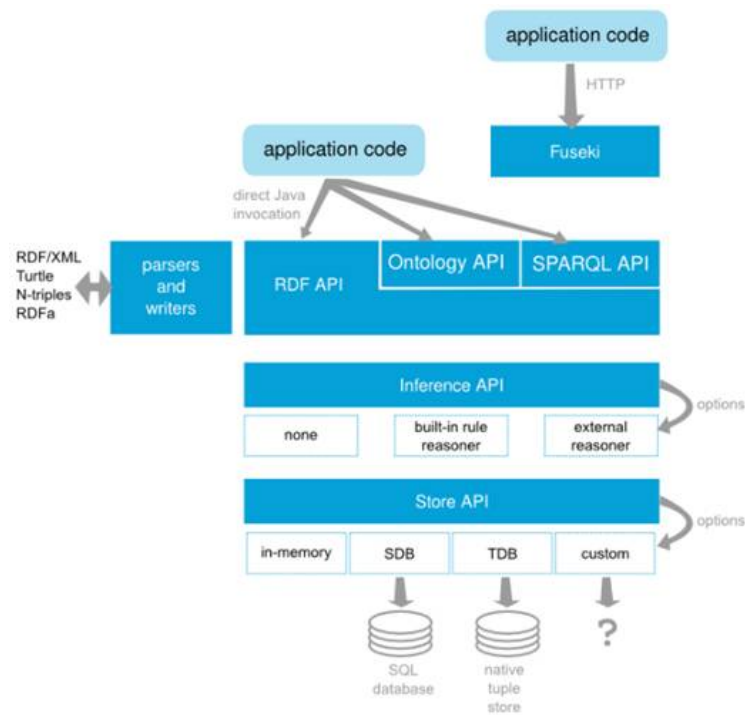


Figure 2.18: Apache Jena Diagram

Its usefulness in the project lies in the fact that the data extracted through GSICrawler and analyzed and semantically enriched through Senpy will also be stored in a Fuseki deployed specifically for this purpose, where a dataset with the extracted data will be stored and will allow the user to perform complex queries through a SPARQL endpoint.

⁴<https://github.com/apache/jena/tree/main/jena-fuseki2>

Semantic models

In this section, the semantic enrichment used for the different metrics obtained will be explained in detail. As mentioned above, the project consists of two types of metrics: software metrics and social metrics. It will be divided into two subsections where the different tools used and the ontologies used by each of them for semantic modeling will be explained. For social metrics, two services of the Intelligent Systems Group will be used, based on the collection and analysis of data from social sources, with semantic enrichment. On the other hand, for software metrics, the OSLC standard will be used, as well as the extension of several of its domains, to adapt it to the use case of the project, which is the semantic enrichment of Stackstorm. Therefore, this block will explain the development of this semantic model through Protégè of the main Stackstorm components.

3.1 Social Models

In this project, social metrics are understood as all data obtained from social sources such as Twitter. In this project, we will mainly obtain data from Twitter, through hashtags from certain software of interest, such as Visual Studio Code, and others. In this way, we

propose an additional way to obtain information about some software of interest and to be able to audit the opinion or feelings of the users about it. For example, it will be possible to see in general terms whether users have received new functionality. For the process of obtaining and analyzing social metrics, the two services mentioned above, GSICrawler and Senpy, will be used. Each service does its semantic enrichment of the data obtained and, when used together, can give very relevant information.

3.1.1 Extracting data with GSICrawler

GSI Crawler is an innovative and useful framework that aims to extract information from web pages by enriching them following semantic approaches[31] As mentioned above, information extraction is performed using snsrape, which has been integrated for the project into GSICrawler. This part will be explained in more detail in later sections. The data extracted by the snsrape scraper are mainly modeled using the Schema ontology.

Schema is a structured data vocabulary founded by Google, Microsoft, Yahoo, and Yandex that defines entities, actions, and relationships on the Internet[32]. These vocabularies cover entities, relationships between entities, and actions and can be easily extended through a well-documented extension model. Schema provides many concepts related to blogging that can be related to the fields obtained from Twitter.

The data model used by Schema is very generic and is derived from the RDF Schema. It is based on a set of types which form a multiple inheritance hierarchy where each type can be a subclass of multiple types.

On the other hand, it is based on a set of properties where each property can have one or more types as its domains. The property can be used for instances of any of these types.

In the development of GSICrawler, the results obtained with the snsrape Python client are mapped to the schema ontology for all tuits, as follows:

Listing 3.1: Code for mapping tuits into schema ontology

```

for tweet in scraper.get_items():

    mytweet = {}
    mytweet["@type"] = ["schema:BlogPosting", ]
    mytweet["@id"] = tweet.id
    mytweet["schema:about"] = query
    mytweet["schema:search"] = query
    mytweet["schema:articleBody"] = tweet.content
    mytweet["schema:headline"] = tweet.content
    mytweet["schema:creator"] = tweet.user.username
    mytweet["schema:author"] = 'twitter'
    mytweet["schema:inLanguage"] = tweet.lang
    mytweet["schema:keywords"] = tweet.hashtags
    mytweet["schema:datePublished"] = tweet.date.strftime('%Y-%m-%d
        T%H:%M:%SZ')

    if tweet.place:
        mytweet["schema:locationCreated"] = tweet.place.fullName

    if tweet.coordinates:
        mytweet['location'] = {
            'lat': tweet.coordinates.latitude,
            'lon': tweet.coordinates.longitude
        }

#...

```

In this way, these semantically enriched data are passed as input to Senpy, which will be the software in charge of performing the sentiment and emotion analysis. The entire relationship between the different services that make up the pipeline will be explained in detail in Section 4.

3.1.2 Sentiment analysis with Senpy

Senpy is a framework for developing, evaluating and publishing web services for sentiment and emotion analysis in text[33].

Provides functionalities for:

- Developing sentiment and emotion classifier and expose them as an HTTP service
- Requesting sentiment and emotion analysis from different providers (ie, Vader, Sentiment140, etc.) using the same interface (API and vocabularies). In this way, applications do not depend on the API offered for these services.
- Combining services that use different sentiment models (e.g., polarity between $[-1, 1]$ or $[0, 1]$ or emotion models (e.g., Ekkman or VAD)
- Evaluating sentiment algorithms with well-known datasets

It is used for the creation of NLP services based on the NIF, Marl, and Onyx vocabularies and the Turtle/n-triples, JSON-LD, and eXtensible Markup Language (XML)-RDF formats. The use of a common semantic model for results and annotations means that other modules of the system, especially the visualization module, need not rely on schemas or formats specific to each service or type of service. This independence is exploited in other modules of the toolkit. For example, more than one type of data store can be used as storage modules, each with its own formats. An ElasticSearch database (JSON-based) can co-exist with a Fuseki data store (RDF-based), provided that the annotations are correct and appropriate conversion mechanisms are in place. Therefore, a proprietary ontology is proposed for Senpy that allows any service to use the same concepts when annotating texts with different sentiment analysis tools. The text analyzes performed in Senpy extract emotional, psychological, and moral information from the text. To extract these annotations, we used the LIWC dictionary and the Moral Fundamentals Dictionary (MFD). This module implements these processes using Senpy plugins, which are modular software additions that can be inserted into the system to enhance its functionality. As mentioned above, a fundamental part of Senpy is its semantic layer. For this purpose, the different ontologies and vocabularies used for sentiment and emotion analysis will be explained.

3.1.2.1 Senpy Annotations

This is the main and indispensable ontology of Senpy, since it semantically defines the annotations. There are many techniques for annotating texts with information from sentiment and emotion analysis. Some of them use scores based on categories (such as the dimensions provided by LIWC). Senpy ontology¹ aims to define the general terms necessary to make

¹<http://gsi.upm.es/ontologies/participation/senpy/ns>

these types of annotations. This ontology has the following classes and properties:

- **Classes:** *Annotation* , *Thing*², *Category*
- **Object properties:** *hasAnnotation*, *hasCategory*
- **Data properties:** *count*, *ratio*

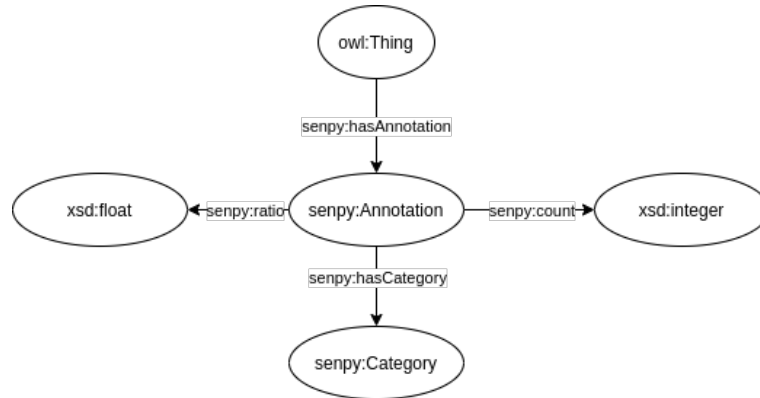


Figure 3.1: Senpy Annotations Ontology Diagram

In conclusion, the purpose of this ontology is to provide a model for the data returned by Senpy, allowing the service to use the same concepts when annotating texts using different sentiment analysis tools. Having explained the Senpy ontology for annotations, it should be said that depending on the plugin used by the framework for the analysis of the received text, one ontology or another will be used.

In this project, priority is given to three, all of them developed by the UPM Intelligent Systems Group. SLIWC, Morality, and Onyx.

3.1.2.2 SLIWC - LIWC dimensions represented as a SKOS taxonomy

Simple Knowledge Organization System (SKOS) provides a model to represent the basic structure and content of concept schemes such as thesaurus, classification schemes, subject heading lists, taxonomies, folksonomies, and similar controlled vocabularies. As an RDF (Resource Description Framework) application, SKOS allows the creation and publication of concepts on the Web, as well as linking them to data on the Web and even integrating them into other concept schemas. This representation of the LIWC dimensions as a SKOS taxonomy is known as SLIWC³.

²<https://www.w3.org/2002/07/owl#Thing>

³<https://gsi.upm.es/ontologies/participation/sliwc/ns/doc/index-en.html>

For the sentiment analysis part, the software used is the Linguistic Inquiry and Word Count (LIWC) standard. Linguistic Inquiry and Word Count (LIWC) is the gold standard in software to analyze word usage. It can be used to study a single individual, groups of people over time, or all social networks. In this case, it will be used to study tweets collected and semantically enriched with GSICrawler[34]

LIWC reads a given text and counts the percentage of words that reflect different emotions, thinking styles, social concerns, and even parts of speech. The importance and popularity of LIWC has led other researchers to adopt its annotation conventions and to use the same format to produce dictionaries compatible with LIWC programs. This project uses LIWC dictionaries for English, Spanish, and Italian.

The LIWC algorithm counts words that reflect emotions, psychological meanings, or social concerns in a text. Thus, the process leverages a dictionary that compiles a wide list of words and their possible inflections, classifying them into a predefined set of categories (e.g., anxiety, anger, family, religion). Consequently, each category present in the text is given a score that corresponds to the number of words found that belong to that category. As an example, if a sentence contains the word 'father', which is classified by the dictionary as 'family' and 'male reference', then both of these categories will have a resulting score of one. Alternatively, if the word 'cousin', which is also a reference to 'family', appears also in the text, then the category 'family' would obtain a score of two.

The categories measured by LIWC are grouped into sets depending on the psychological process they describe. Four of them are taken into account for this analysis: drives, affective, personal concerns, and social. The set 'Drives' gives insight into the motivations of the person behind the text. Measure five parameters, which are affiliation, achievement, power, reward, and risk. Feelings like anxiety, anger, sadness, or positive and negative emotions are part of the set of the 'Affective process.' The 'Personal concerns' set measures how concerned the person behind the text is about different life issues such as work, leisure, home, money, religion, and death. Drives, affective, and personal concerns are considered the three main psychological variables and have been used as a reference to analyze radical discourses (Buckingham & Alali, 2020). The set of 'social processes' is also included to provide more information on references to family, friends, female, and male figures.

Once LIWC has been introduced, it should be noted that the project has made use of the semantic version developed by the Intelligent Systems Group, a version of the LIWC annotation scheme. It consists of two parts. First, we have an ontology representing the general concepts used in LIWC annotation (e.g., dimensions, categories, word-level dimen-

sions, document-level dimensions, etc.). The second part uses these concepts to provide specific elements of LIWC dictionaries, such as specific categories and their hierarchical relationship to each other. These categories have been modeled both in the form of ontology (i.e., classes) and SKOS taxonomy, so that the hierarchical structure can be exploited independently of the ontological relationships.

- **Classes:** *Category*⁴, *Concept*⁵
- **Important named individuals:** *Anger, Anxiety, Core Drives and Needs, Emotional Tones, Family, Morality*, etc.

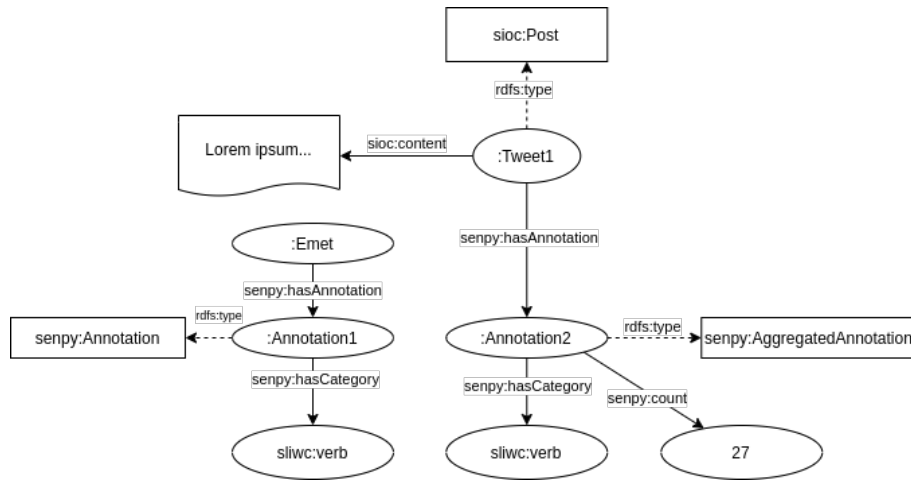


Figure 3.2: SLIWC Ontology Diagram

Attempts have been made to interrelate both ontologies more efficiently, eliminating the classes, objects, and data properties of Senpy in SLIWC. That is, a new Senpy Annotation ontology has been created with some of the previously existing SLIWC properties and classes for this purpose.

- **Deleted classes:** *Agent*⁶, *Annotation*
- **Deleted object properties:** *hasAnnotation*, *hasCategory*
- **Deleted data properties:** *count*, *ratio*

⁴<http://www.gsi.upm.es:9080/ontologies/participation/senpy/ns/doc/index-en.html#Category>

⁵<https://www.w3.org/2009/08/skos-reference/skos.html#Concept>

⁶<http://purl.org/dc/terms/#Agent>

All the information about SLIWC and its different properties is referenced in footnote 3 since the objective of this Section is to explain the developments and changes made for this use case. These are the changes and adaptations required for it. The total properties of SLIWC are very numerous and can be obtained from the link mentioned above.

3.1.2.3 Morality - MFT concepts represented as a SKOS taxonomy

Moral foundation theory[35] was created by a group of social and cultural psychologists to understand why morality varies so much between cultures, yet still shows so many similarities and recurring themes. In summary, the theory proposes that several innate and universally available psychological systems are the foundations of 'intuitive ethics.' Each culture then constructs virtues, narratives, and institutions on top of these foundations, thereby creating the unique moralities we see around the world and conflicting within nations, too.

This ontology⁷ is a SKOS taxonomy that aims to represent the main concepts of MFT and has the following classes and properties.

- **Classes:** *Foundation*
- **Object Properties:** *hasMorality*

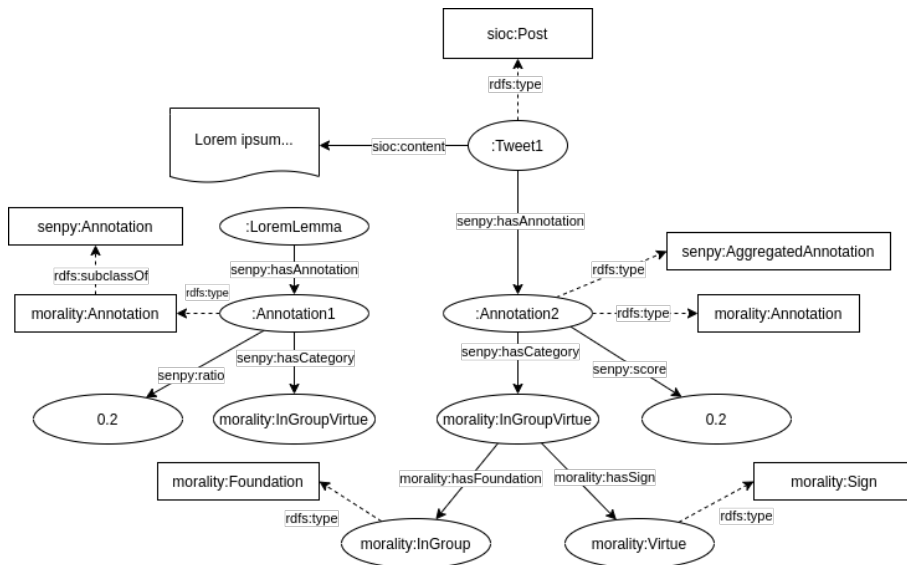


Figure 3.3: Morality Ontology Diagram

⁷<http://gsi.upm.es:9080/ontologies/participation/morality/ns/doc/index-en.html>

3.1.2.4 Onyx Ontology

Onyx⁸ is a standardized ontology designed to annotate and describe emotions expressed by user-generated content on the Web or in certain information systems. The goals of the Onyx ontology to achieve as an ontology are the following:

- Enable the publication of raw emotion data in user-generated content.
- Provide a schema to compare emotions from different systems (polarity, themes, characteristics)
- Interconnect emotions by linking them to contextual information expressed with concepts from other popular ontologies or specialized domains.

It is made up of the following classes and the most important properties:

- **Classes:** *AggregatedEmotion*, *AggregatedEmotionSet*, *Emotion*, *EmotionCategory*, etc.
- **Object Properties:** *ActionTendency*, *source*, *hasEmotionSet*, *hasEmotionCategory*, etc

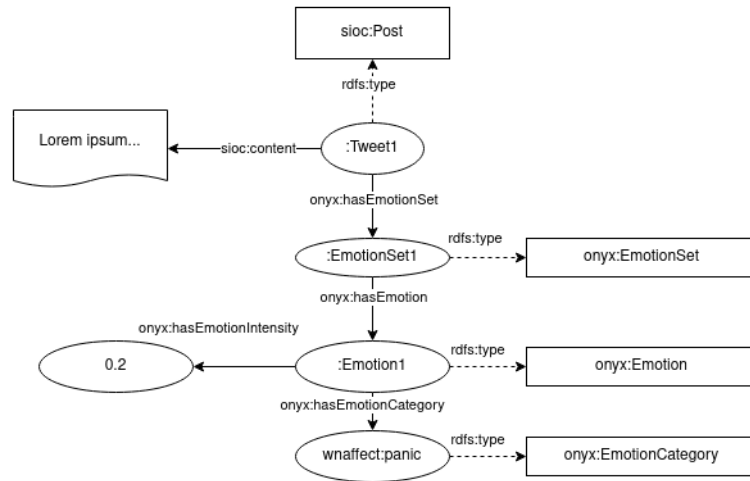


Figure 3.4: Onyx Ontology Diagram

With these ontologies, the semantic layer used for the collection and analysis of social metrics is explained. As mentioned above, depending on the plugin used by Senpy, the result of the analysis will be one or another, but always using Senpy's annotation vocabulary.

⁸<https://www.gsi.upm.es/ontologies/onyx/>

The results of the analysis, as well as the output data in Turtle/N3 representation, will be discussed in detail in later sections.

3.2 Software Models

Another fundamental part of the project, since it includes all the software monitoring parts and the advantages of using the OSLC standard in DevOps tools such as Stackstorm, is the semantic layer developed in the project for this tool.

First, the OSLC CORE specification will be explained, which will be the main basis, since the requirements set there will have to be respected by all OSLC domains. Within this, several domains and concepts will be explained, such as OSLC Actions, Events, and Automation, which will be necessary to explain the semantic model developed for Stackstorm.

Finally, the semantic model proposed to adapt Stackstorm and OSLC will be explained, which will be the backbone of the adapter, which will be explained in later sections.

3.2.1 OSLC Core

Open Services for Lifecycle Collaboration (OSLC) is a community whose goal is to develop specifications for the integration of tools. These specifications allow independent software and product lifecycle tools to integrate their workflows to support end-to-end lifecycle processes. This is because, using the benefits of Linked Data, the tools use a common, open language to integrate their data. This will be the basis for this project. Within the main lifecycle tools, we can highlight defect tracking tools, requirements management tools, and test management tools. There are many more examples in software and product development.[36]

The OSLC community is organized into working groups that address integration scenarios for individual topics such as change management, test management, requirements management, and configuration management. These topics that have OSLC working groups and specifications are called "domains" in OSLC. Each working group explores integration scenarios for a given lifecycle topic and specifies a common vocabulary for the lifecycle artifacts needed to support the scenarios. To ensure consistency and integration between these domains, each working group relies on the concepts and rules defined in the OSLC

Core specification, which is produced by the Core working group. These specifications will be the basis for the rest of the OSLC domains. In this way, the domains can be integrated with each other, facilitating the integration of different tools.

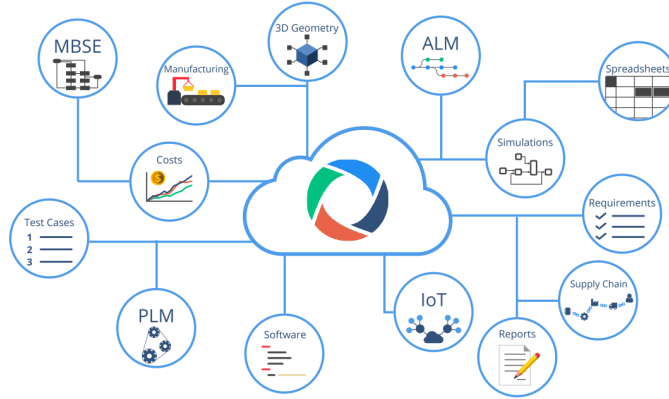


Figure 3.5: OSLC integration diagram

The OSLC core specifies the main integration techniques for integrating lifecycle tools. It consists mainly of standard rules and patterns for the use of HTTP and RDF that all working groups in the domain must adopt in their specifications. The core specification is not intended to be used by itself. Since there is no generic lifecycle tool, each tool is specialized in one or more domains such as requirements, defect tracking, testing, etc. The core specification, along with one or more of the OSLC domain specifications, describes the OSLC protocols offered by a domain tool. OSLC does not attempt to standardize the behavior or capability of any tool or class of tool in a generic way. OSLC specifies a minimum number of protocols and a small number of resource types that must be met to allow two of these tools to work seamlessly.

These domains (or working groups in OSLC) are as follows[37]:

- **OSLC Core:** Central concepts and the relationship between them. These concepts must be respected by all domains.
- **OSLC Query:** Defines a simple, implementation-independent selection and projection query capability
- **Quality Management:** Define the OSLC services and vocabulary for the Quality Management domain.
- **Requirement Management:** Define the OSLC services and vocabulary for the Requirements Management domain

- **Change Management:** Define the OSLC services and vocabulary for the Change Management domain
- **Tracked Resource Set:** Allows servers to expose a set of resources whose state can be tracked by clients.
- **Architecture Management:** Define the OSLC services and vocabulary for the Architecture Management domain.
- **Automation:** Define the OSLC services and vocabulary for the domain that supports the automation of sequences of actions on OSLC resources. It will be important for the project and will be explained in depth in later sections.
- **Actions and Events:** In this case the Actions domain extension and the creation of the Events domain, developed by the Intelligent Systems Group, will be used and, as of today, pending review by the OSLC community. The idea of these domains is to provide more concepts to the Automation domain.

Domains are called phases of the lifecycle, and special properties are defined for tools that belong to these phases. Within a resource specified by an OSLC domain, the goal is to define the minimum properties that are valuable for integration, not all the properties that might be present in a particular tool's resources.

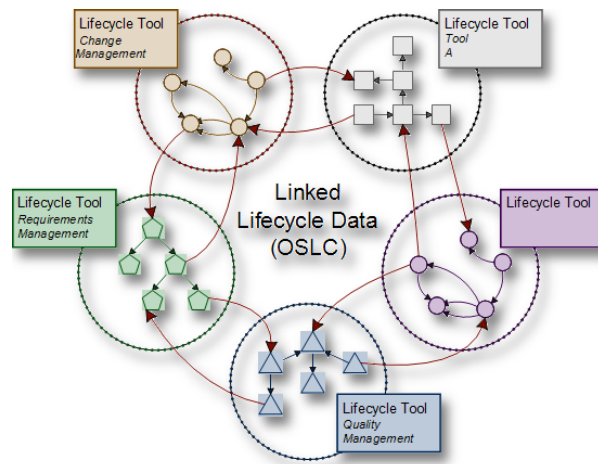


Figure 3.6: OSLC domains diagram example

In this project, an ontological model will be created to define Stackstorm rules, extending and using OSLC domains. OSLC's relevance lies in its proposal to replace the current web of documents with the web of data based on Linked Data principles. It seeks to transition from a world of incompatible systems to a world of connected data through an open standard.

OSLC is based on W3C Linked Data and its principles. Therefore, OSLC offers two main techniques for tool integration[38]:

1. **Data binding via HTTP:** OSLC specifies a common tool protocol for creating, retrieving, updating, and deleting (CRUD) lifecycle data based on Internet standards such as HTTP and RDF using the Linked Data model. Linking is achieved by embedding the HTTP URL of one resource in the representation of another.
2. **Linking Data via HTML User Interface:** OSLC specifies a protocol that allows a tool or other client to have a fragment of another tool's web user interface displayed, allowing a human user to link to a new or existing resource in the other tool or to preview information about a resource in another tool. This allows a tool or other client to exploit existing user interface and business logic in other tools by integrating information and process steps. In some circumstances, this is more efficient and offers more functionality to the user than implementing a new user interface and then integrating it through an HTTP CRUD protocol.

In OSLC, each lifecycle artifact, e.g., a requirement, a defect, a test case, a source file or a development plan, etc., is an HTTP resource that is manipulated using the standard methods of the HTTP specification (GET, PUT, POST, DELETE). Each resource has an RDF/XML representation, but may have representations in other formats, such as JSON or HTML.

The OSLC Core specification⁹ defines a number of simple HTTP and RDF usage patterns and a small number of resource types that help tools integrate and make the lifecycle work. Many specifications have a "closed model", by which we mean that any reference to a resource in the specification will necessarily identify a resource in the same specification or in a referenced specification. UML is an example of a closed specification: Every UML reference refers to another UML object. On the contrary, the HTML anchor tag can point to any HTTP resource, not just other HTML resources. OSLC works more like HTML in this sense. Here are some examples.

- Any HTTP resource can be contained in a ServiceProvider, not just the resources defined in the OSLC specifications.
- A URL reference in an OSLC resource can, in general, point to any HTTP resource,

⁹<https://docs.oasis-open-projects.org/oslc-op/core/v3.0/oslc-core.html>

not just an OSLC resource. OSLC specifications generally do not restrict the value of a property in one OSLC specification to be the URL of a resource in a different OSLC specification (note 3), although it is common for a property defined in an OSLC specification to restrict its value to be the URI reference of a resource in the same specification.

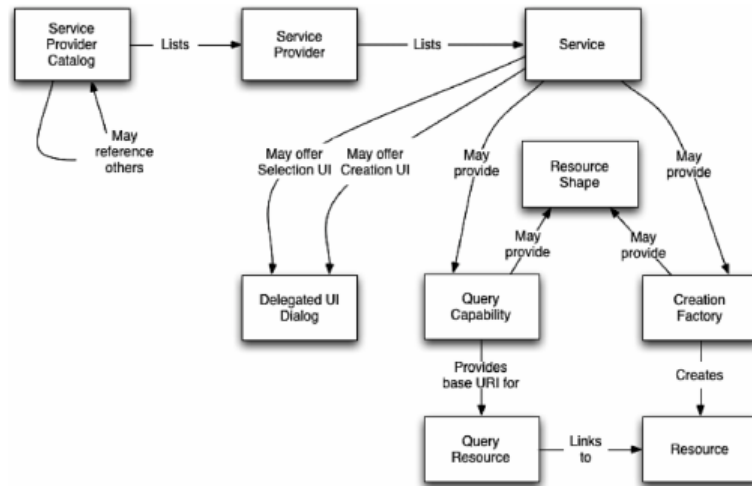


Figure 3.7: OSLC Core Concepts and Relationships

OSLC specifications generally avoid restricting a reference to be a resource in another specification. This independence allows OSLC specifications to evolve independently and new ones to be added without changing existing ones, and also allows tools at different version levels to interoperate.

In this project, it will be useful to understand the concept of an OSLC Service Provider for the future implementation of a semantic model for Stackstorm rules. The diagram reflects the relationship between the different OSLC CORE concepts, which will be those that any integration must comply with in order to meet the requirements of the standard. These OSLC resources are defined in the Primer. The OSLC Primer defines the resources, defined by the Core specifications, necessary for the development of any other domain. These concepts will be fundamental to understanding the structural operation of any semantic adaptation of any product to OSLC.

3.2.1.1 Service Provider

Almost all lifecycle tools have organizational concepts that divide the overall artifact space in the tool into smaller containers. An example of such containers could be projects or modules, databases, etc. . Each artifact created in the tool is created within one of these container-like entities, and users can list existing artifacts within one. These container concepts are very important for the use of tools: The container in which the artifacts are placed and in which the artifacts are located is essential for the way of working and can reflect the project being worked on or the product to which the artifacts belong. OSLC defines the ServiceProvider concept to allow products to expose these containers or partitions for integration scenarios. Service providers answer two basic questions, which are the following:

- To which URLs should I POST to create new resources?
- Where can I get a list of existing resources?

A ServiceProvider is intended to represent a "container" of resources that is hosted by a tool. In this way, OSLC Core generically defines a way to expose a resource container for a particular tool. In the project, the Stackstorm instance itself will be used as the Service Provider, since it will be the container for rules and actions. As discussed previously, OSLC makes use of the benefits of Linked Data and its principles. Therefore, this Service Provider, as well as the rest of the concepts, will have its URI assigned to perform HTTP requests to create, obtain, update, or delete resources. In the case of the Service Provider, unlike what you might think from the definition above, if you make an HTTP GET, you will not get a list of the existing resources in that container. You will get the general properties of the Service Provider as well as metadata about it, including URLs that can be used to find or create resources. Two fundamental properties of a Service Provider are

- *oslc:creation*: URL of a resource that you can POST to create new resources.
- *oslc:queryBase*: URL of a resource that you can GET to get a list of existing resources in the ServiceProvider. This URL is called the "queryBase URL", and the resource identified by this URL is called queryBase.

An example of the simplest response that could be obtained, in Turtle notation, to an HTTP GET request to a Service Provider URL might be:

Listing 3.2: Example of OSLC Service Provider representation in N3

```
@prefix oslc: <http://open-service.net/ns/core#>.
<http://acme.com/toolA/container1>.
to oslc:ServiceProvider;
oslc:creation <http://acme.com/toolA/container1/contents>;
oslc:queryBase <http://acme.com/toolA/container1/contents>.
```

In the simplest case, the creation URI and the queryBase URI will, in fact, be the same URL. By means of an HTTP POST request whose body is the RDF notation definition of a resource, it can be created.

OSLC Core supports more complex options for Service Providers, including the ability to have more than one creation URI and more than one queryBase URI, the ability to attach properties to each creation and each queryBase URI, and the ability to attach properties to each creation and each queryBase URI. The queryBase resource identified by a queryBase URI of a ServiceProvider is an RDF Container resource that lists resources in the ServiceProvider. An example of a queryBase response could be:

Listing 3.3: Example of OSLC Resources List by make a HTTP GET into a queryBase

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
<http://acme.com/oslc/container/1>
<rdfs:member> <http://acme.com/oslc/resource/000000000>;
# 999999998 more triples here
<rdfs:member> <http://acme.com/oslc/resource/999999999>.
```

In this way, all data for a resource can be obtained by HTTP GET from the URIs defined in the example as *rdfs:member*. In the case of the project, as will be explained later in more detail, this OSLC Service Provider concept will be used to define the Stackstorm instance.

3.2.1.2 OSLC Resources

As mentioned above, the concept of Service Provider in OSLC could be understood as a container of OSLC resources. In this project the Stackstorm instance will be modeled as an OSLC Service Provider while Stackstorm rules will be modeled as OSLC resources.

According to the OSLC Primer, any HTTP resource with any representation can be found within a ServiceProvider. An OSLC resource is simply a resource whose type is defined in some OSLC specification, usually one of the domain specifications created by one of the OSLC domain working groups. These resources have a number of main characteristics:

- An RDF/XML representation of the resource can be requested. All OSLC resources have their state defined by a set of RDF properties that may be required or optional.
- OSLC protocols use standard media types. The goal is for any standards-based RDF or Linked Data client to be able to read and write OSLC data, and defining new media types would preclude this in most cases. This will be of vital importance when integrating different DevOps tools by using this standard to define their own OSLC adapters.

OSLC resources use common property names for common concepts. In the current state of lifecycle tools, each tool defines its own properties for common concepts such as tag, description, creator, last modification time, priority, etc. In many cases, an administrator can define these properties locally for an installation, so tool vendors cannot control the vocabulary. This makes it much more difficult for organizations to subsequently integrate tools into an end-to-end lifecycle. OSLC solves this by requiring all tools to expose these common concepts using a common vocabulary for properties, which you will recall are identified by URIs in RDF. Tools may choose to additionally expose the same values under their own private property names on the same resources.

3.2.2 OSLC Automation Specification

Having described the main concepts and specifications of OSLC Core, the Automation domain¹⁰ must be described. Within this domain, we will find OSLC Actions and Events that will be useful to understand the advantage of using an OSLC-Stackstorm adapter for semantic adaptation of DevOps tools.[39]

This domain defines the specifications needed to define the resources and operations supported by an Open Services Automation for Collaboration Lifecycle (OSLC) provider. Automation resources define the automation plans, automation requests, and automation

¹⁰<https://archive.open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.1/>

results of the software development, test, deployment, and operation lifecycle. They represent individual resources as well as their relationships with other automation resources and other linked resources outside the automation domain. The purpose of this specification is to define the set of HTTP-based RESTful interfaces in terms of HTTP methods. GET, POST, PUT and DELETE, HTTP response codes, MIME type handling, and resource formats.

The key terminology to understand the domain can be summarized in the following.

- **Service Provider:** An implementation of the OSLC Automation specification as a server. OSLC Automation clients consume these services
- **Automation Resource:** A resource managed by the Automation service provider. The types of resource defined by this specification are the Automation Plan, the Automation Request, and the Automation Result.
- **Automation Plan:** Defines the unit of automation which is available for execution.
- **Automation Request:** Defines the submission of the information required to execute an Automation Plan and indicates the desired execution state.
- **Automation Result:** Defines the intermediate and final execution status of an Automation Request, along with contributions to the result.

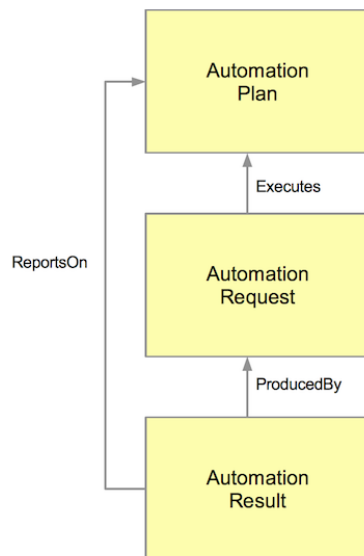


Figure 3.8: Relationships between Automation Resources

An Turtle/N3 representation of an Automation Result could be:

Listing 3.4: Example of OSLC Automation Result representation in N3 notation

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_auto: <http://open-services.net/ns/auto#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://example.org/#link1> a rdf:Statement ;
    dcterms:title ""Build Definition 123: Pet Shop App production build
"";
    rdf:object <http://example.com/plans/123> ;
    rdf:predicate oslc_auto:reportsOnAutomationPlan ;
    rdf:subject <http://example.com/results/4321> .

<http://example.com/results/4321> a oslc_auto:AutomationResult ;
    oslc_auto:reportsOnAutomationPlan <http://example.com/plans/123> .
```

In this way, the automation resources and the relationship between these resources are described with respect to the OSLC core requirements. It will be useful to understand this concept since Stackstorm is an automation tool, and for its semantic layer, part of the concepts mentioned above will be used for this domain.

3.2.3 OSLC Events and Actions

Having explained the Automation domain, it is important to describe other types of OSLC resources that will be important to understand the purpose of the project. On the one hand, the OSLC Automation domain provides the semantic layer necessary to describe an automation server. As discussed in Section 1, this work is part of the SmartDevOps project¹¹ to integrate DevOps tools with semantic technologies and the Big Data approach. In this perspective, it was detected that the OSLC Actions domain was outdated, so two semantic models were developed to model actions and events. The definition of these two models will be very useful since the Stackstorm-OSLC adapter developed in the project will receive a series of actions (HTTP Request, manual activation of a rule, etc.) and generate events (modification of a rule, etc.) that will be listened to by an external server. In the case of SmartDevOps, it will be a Kafka server in charge of listening to events generated by different DevOps tool adapters with OSLC, but this will be left out of this project and will be explained in more detail in Section 4.

¹¹<https://smartdevops.gsi.upm.es>

Therefore, in this context, the proposal of Guillermo García Grao and Alvaro Carrera Barroso[40] from the Intelligent Systems Group will be used to provide a semantic model of OSLC events and actions that extends the OSLC Automation domain.

In the case of the project, the adapter will receive RDF graphs of OSLC Action resources that will imply an action on Stackstorm and will be able to generate OSLC Event type resources to perform what is needed. These two concepts will be key to the monitoring framework, as a listing of events and actions (other than OSLC TRS server logs) performed by/for the adapter to another server or DevOps tool is required. In this case, it will be formed by two servers in parallel; on one side, the StackStorm-OSLC adapter and another server that will be listening for actions and will generate events to the adapter, but this space could be used in the future by another DevOps tool (that has developed its respective adapter) so that both could be connected in a standardized and interoperable way using events and actions.

It is important to understand that the concept of actions and events in OSLC is not equivalent to the concept of actions and events in Stackstorm. On the one hand, we are talking about OSLC resources and, on the other hand, actions performed by the tool itself. The Stackstorm-OSLC adapter will be able to listen for OSLC actions (internal or hypothetically from another DevOps tool) and generate OSLC events to a messaging channel such as Kafka or others.

All of this architecture will be explained in detail in Section 4, but it is worth introducing it in this section to understand the usefulness of the extension of the OSLC Automation domain through OSLC Actions and OSLC Events.

On the one hand, the proposed OSLC Actions domain makes use of several Automation domain classes such as *AutomationPlan*, as well as other Core classes such as *ResourceShape*. This OSLC Actions domain consists of:

- **Prefix:** *oslc_actions*
- **Classes:** *ActionDispatcher*, *Action*
- **Properties:** *executedBy*, *futureAction*, *availableAction*, *executesOn*

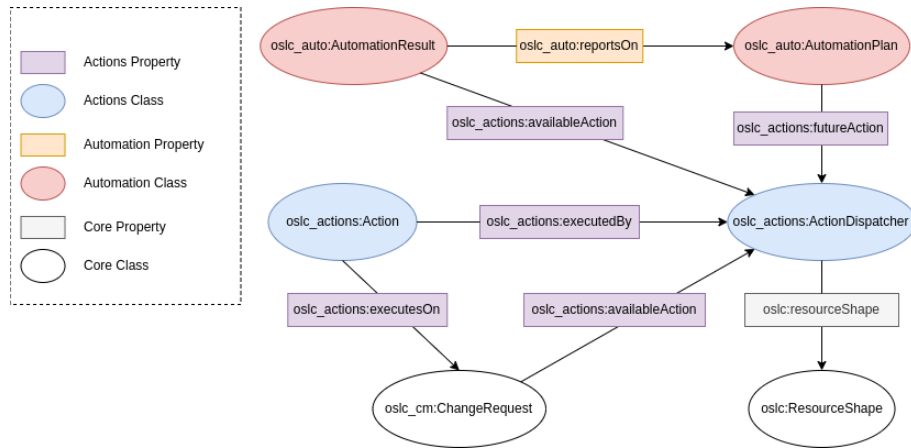


Figure 3.9: OSLC Actions domain

On the other hand, the OSLC Event domain will use classes and properties from the Automation domain as well as from the CORE domain since, as mentioned before, the idea of generating both models is to extend the Automation model by including events and actions, orienting it towards the DevOps approach.

- **Prefix:** *oslc_events*
- **Classes:** *Event*
- **Properties:** *generatedBy*

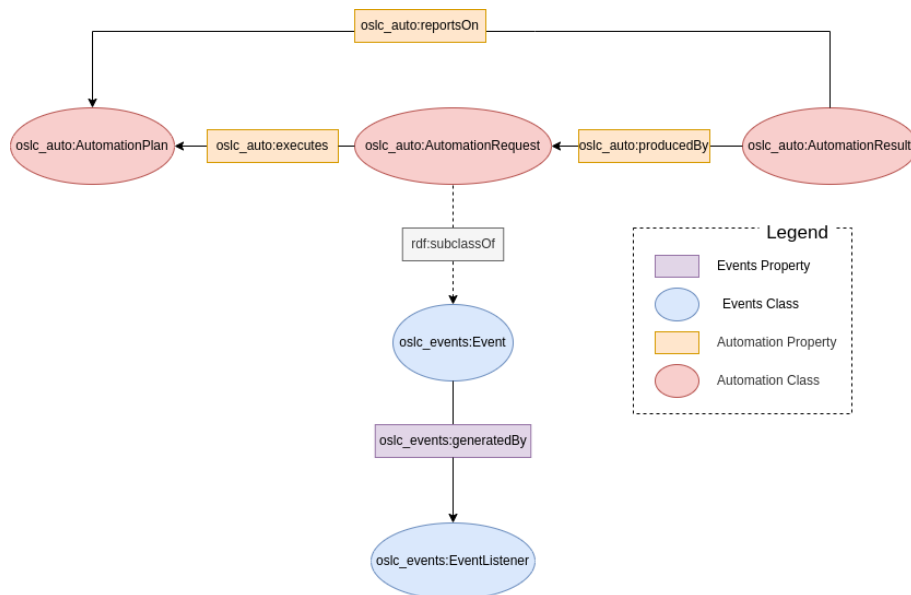


Figure 3.10: OSLC Events domain

3.2.4 Tracked Resource Set Specification

The tracked resource set protocol (TRS) allows a server to expose a set of resources so that clients can discover that set of resources, track additions and deletions from the set, and track changes in the state of resources in the set. It is used to handle sets that contain a large number of resources, as well as very active resource sets that undergo continuous changes. Performs the function of a kind of log server on a set of resources. In this way, if an action is performed that interacts with any of the resources, it is recorded at the TRS endpoint and can be accessed via HTTP GET. It is based on HTTP and follows RESTful principles.[41]

The following key concepts could be distinguished to explain this type of resource:

- **Tracked Resource Set (TRS):** Describes a resource that defines a finite and enumerable collection of Tracked Resources expressed as a Base and a Change Log.
- **Tracked Resource:** A resource identified by a URI that is a member of one or more Tracked Resource Sets.
- **Base:** The portion of a Tracked Resource Set representation that lists tracked resources at some point in time.
- **Change Log:** The portion of a Tracked Resource Set representation detailing a series of Change Events for Tracked Resources, where those changes are relative to the Base.
- **Change Event:** Identifies an addition, removal, or state change of a Tracked Resource in a Tracked Resource Set. These events are represented using the three RDF classes *trs:Creation*, *trs:Modification*, and *trs:Deletion*.
- **TRS Server:** An application or application component that provides one or more Tracked Resource Sets.

In the project, it will be used to have a constant monitoring of all the events that occur in Stackstorm. When a rule is activated/deactivated, deleted, or created, an event of type OSLC Event will be created and stored in the TRS to have a complete view of the events produced in the system.

The following example illustrates the contents of a changeLog in the notation n3.

Listing 3.5: Example of a changeLog in TRS in N3

```
# Resource: http://cm1.example.com/trackedResourceSet
@prefix trs: <http://open-services.net/ns/core/trs#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://cm1.example.com/trackedResourceSet>
  a trs:TrackedResourceSet ;
  trs:base <http://cm1.example.com/baseResources> ;
  trs:changeLog [
    a trs:ChangeLog ;
    trs:change <#2>, <#1>.
  ] .
<#2>
  a trs:Modification ;
  trs:changed <http://cm1.example.com/bugs/22> ;
  trs:order "102"^^xsd:integer .
<#1>
  a trs:Deletion ;
  trs:changed <http://cm1.example.com/bugs/21> ;
  trs:order "101"^^xsd:integer .
```

This concept serves to understand the concept of log server with OSLC, since one of the fundamental parts of the project will be to provide a semantic layer capable of receiving actions and generating events in a fully monitored way.

3.2.5 Stackstorm-OSLC semantic model

As mentioned above, a Stackstorm semantic layer will be proposed that includes its core components, such as actions or rules. It will be based on the OSLC core, as well as other domains such as OSLC automation and OSLC actions. The idea of developing our own vocabulary, using classes and properties of OSLC domains, is to allow the integration of this with other DevOps tools in the future, as long as they have their own OSLC adapter, with the advantages and benefits that give us the use of the semantic web and Linked Data. The development of this adapter will be the backbone of the project and will be explained in detail in later sections. The project will focus on semantically defining the Stackstorm rules, as they will be the ones that receive the actions (HTTP GET, POST, or PUT) to be updated, created, or deleted. These actions can be performed by an external user through

HTTP requests to the adapter or from the Stackstorm interface itself. Protégé, explained in Section 2, has been used for the development of the model.

First, the main OSLC CORE classes have been imported, as they will be the specifications to be met by the adapter. Before starting with the explanation of the proposed model, it should be noted that the project has made a complete adaptation of one of the main components of Stackstorm, but not all of them. That is, the proposed model tries to semantically model the Stackstorm rules, as well as some parameters that give relevant information about them, such as the action to be performed by the rule when the trigger is activated, and the trigger itself. The proposed model does not include the Stackstorm tool in its entirety and is intended to serve as a guide for possible future semantic models.

The semantic model proposed in the project for Stackstorm with OSLC is based on the assumption that, as a resource container, the Stackstorm instance itself will be the OSLC Service Provider. As mentioned above, the OSLC Service Provider acts as a resource container, so it has been decided to define the Stackstorm instance as a Service Provider and the rules as Resources contained by that Service Provider. As for the rules, they have been defined as subclasses of the AutomationPlan resource, explained in previous points, since it has been considered that a rule is not an ordinary resource (as it could be a Github Issue or a Bugzilla Bug), but it is part of an Automation environment, so it will be interesting to adapt it to the OSLC Automation domain.

First, as mentioned above, the StackStorm tool itself has been considered for the project as an OSLC Service Provider so its Data Properties will be inherited from the Service Provider:

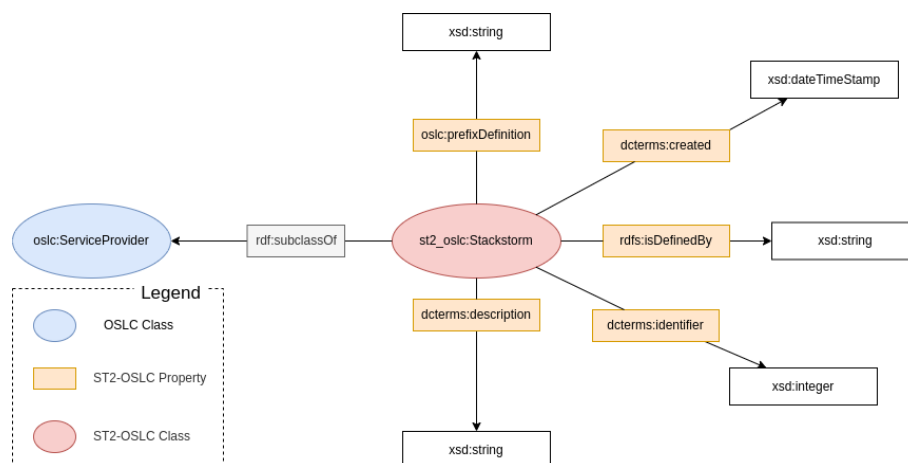


Figure 3.11: Stackstorm-OSLC model

As discussed above, a Stackstorm rule could be considered a subclass of the `ActionPlan` class in the Automation domain. This, in turn, will have the following Data Properties, which define it.

- **Classes:** *StackstormRule*
- **Data Properties:** *ruleAction*, *ruleId*, *rulePack*, *ruleTitle*, *ruleTrigger*, *ruleStatus*, *ruleRef*
- **Data Properties for Stackstorm actions and Triggers:** *actionRef*, *triggerType*, *triggerRef*

The overview of the model adopted by the Stackstorm rule, inherited from Automation-Plan with all its Data Properties, can be seen in the following diagram:

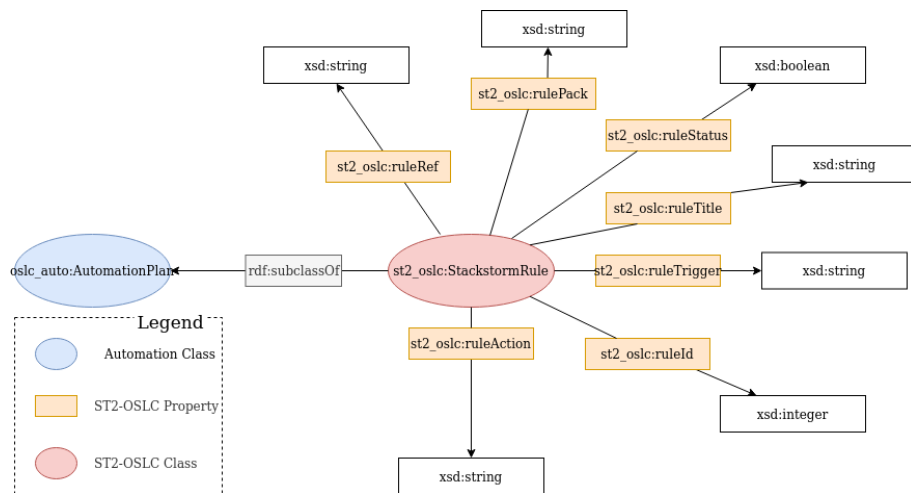


Figure 3.12: Stackstorm Rule Semantic Model

Another important part included in the model is the Stackstorm actions. It should be said that Stackstorm actions do not refer to the same concept as OSLC actions. The former refers to the interaction between the adapter and the Stackstorm API itself to create, delete, or update a rule, while the latter refers to resources of the OSLC Action domain, explained above.

Therefore, it is convenient to relate both concepts semantically, and Stackstorm actions are considered to inherit from the OSLC Action domain mentioned above. Thus, the 3 actions that the adapter developed in the project will receive will be the creation, deletion, and activation or deactivation of a rule.

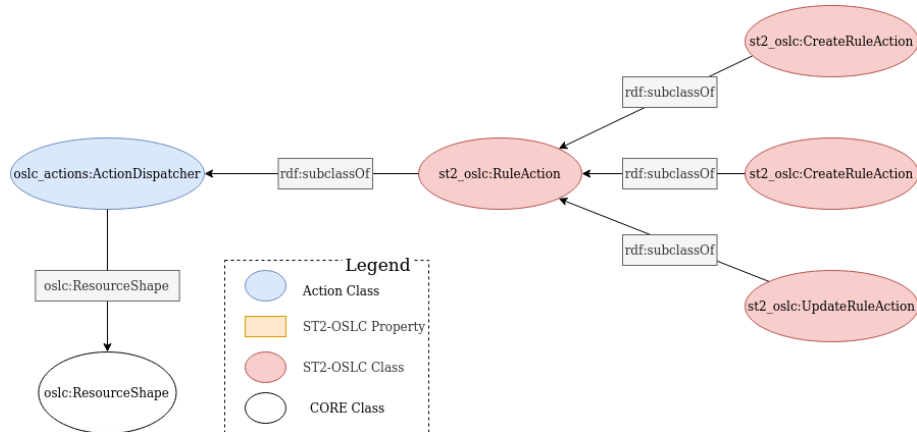


Figure 3.13: Stackstorm Actions - OSLC Semantic Model

This is how the Stackstorm and OSLC semantic model proposal is defined. As mentioned above, this proposal is not intended to cover all Stackstorm components, but rather to serve as a guide for future implementations or extensions that could be made. In the project, the rules, as a fundamental part of Stackstorm, are modeled by means of a Stackstorm-OSLC adapter in such a way that it will act as a receiver of OSLC actions and a generator of OSLC events.

In conclusion, this block has described the different ontologies used for the extraction and analysis of social metrics through Twitter, as well as the OSLC domains used to describe the model that will be used in the Stackstorm-OSLC adapter. The semantic layer occupies an important part of the project, as it provides interoperability, thanks to the benefits of Linked Data and the use of the OSLC standard, as well as thanks to the tools developed by the Intelligent Systems Group to extract data from social sources and perform sentiment and emotion analysis using semantics.

Architecture

This section will describe the architecture of the project. First of all, the complete architecture of the SmartDevOps project will be described, since this work is part of this project and it is necessary to understand the general architecture of the project in order to understand the specific architecture of the work. As for the specific architecture of the project, it will be divided into two.

On the one hand, the proposed architecture for the extraction and analysis of social metrics will be described, by means of a pipeline that links the different software used for this purpose, mentioned in previous points. On the other hand, the architecture proposed for the Stackstorm-OSLC adapter and its monitoring framework, based on actions and events, will be described.

It will also be explained how to include in the monitoring framework other DevOps tools based on semantic technologies and OSLC, as proposed in the SmartDevOps project.

4.1 Overview of the Architecture

As mentioned above, this project is part of a larger project called SmartDevOps¹. This project seeks to simplify and automate the process of integrating tools into Big Data projects by combining the flexibility of OSLC semantic standards with the power of an event-driven rule engine. The complete SmartDevOps architecture considers the integration of more DevOps tools, as well as other modules such as the rule-based automation module that will not be considered for this work. Knowing the general architecture of the SmartDevOps project will be important to know in a more detailed way the architecture proposed for this work.

The general architecture of SmartDevOps is shown in the following diagram:

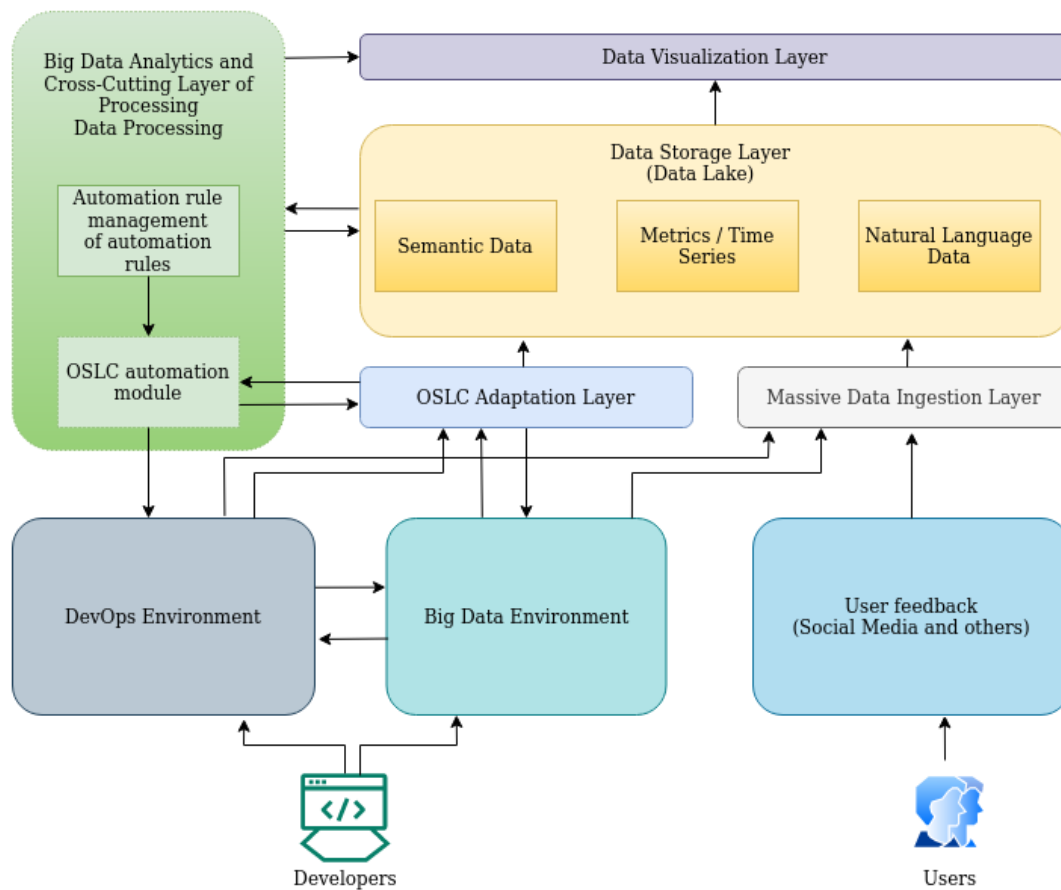


Figure 4.1: SmartDevOps platform general architecture diagram

¹<https://smartdevops.gsi.upm.es/>

In this diagram you can see the different parts involved in this work, which have been mentioned in previous sections, from the OSLC adaptation layer to the social metrics pipeline.

One of the main characteristics of the SmartDevOps architecture is that it seeks the interoperability of different DevOps tools, under the use of semantic technologies and the OSLC standard. Therefore, there are several modules or layers that will not be covered in this work, due to their breadth and complexity. Of all the layers proposed in the general architecture, the OSLC adaptation layer, the OSLC monitoring framework, the DevOps environment and everything related to the extraction, analysis and visualization of social metrics will be of special interest for this work.

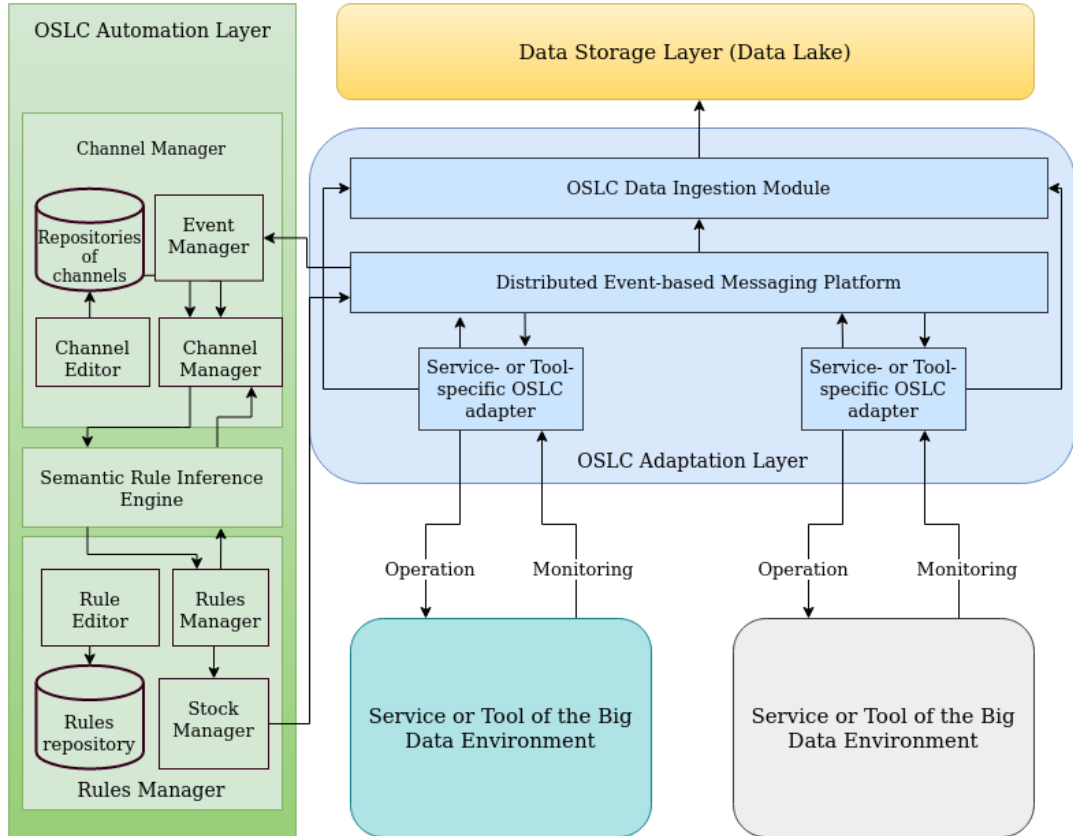


Figure 4.2: SmartDevOps OSLC Layer

This work focuses on the development of an architecture that poses a specific use case for the general SmartDevOps architecture. That is, this work focuses on the development of action and event-based monitoring framework for the proposed Stackstorm-OSLC adapter and a specific architecture for collecting and analyzing social metrics through Twitter, to

measure user feedback for a given software.

Therefore, it can be said that the OSLC adaptation layer will be in charge of providing the DevOps tool (Stackstorm in the case of the project) with the ability to receive actions and generate OSLC events. These events will be sent to Kafka through the use of Topics. These events will be processed by Kafka and sent in the form of OSLC Actions to all those other DevOps tools that are subscribed to this Topic, thus achieving the interconnection between them through semantic technologies and the use of the common and open OSLC standard. Everything related to the incorporation of new DevOps tools is not contemplated in this work except in the future lines, as it is contemplated in the SmartDevOps project.

The general architecture of this OSLC layer can be seen in the following diagram:

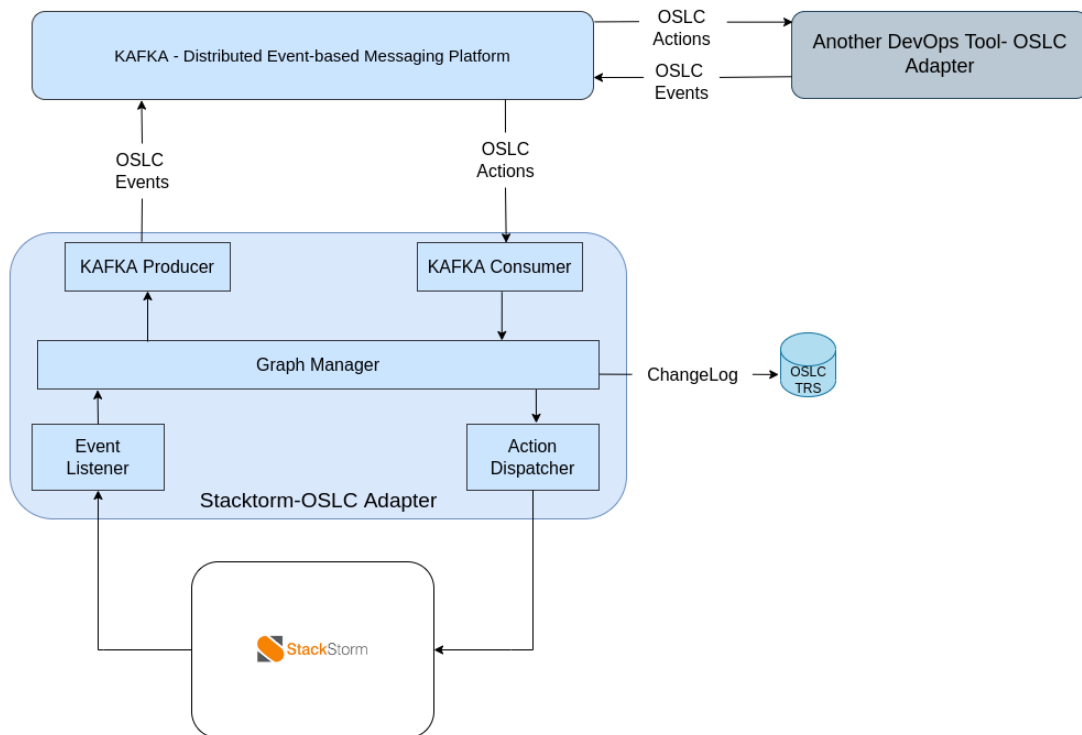


Figure 4.3: Stackstorm-OSLC General Diagram

The following modules should be highlighted:

- **Stackstorm:** An instance that will be deployed and will perform actions such as consulting, activating, deleting, or creating a rule.
- **Graph Manager:** It will be in charge of developing all the semantic parts of OSLC, as well as the one in charge of adding logs of everything that happens in the tool, in the TRS. It will be the module in charge of interacting with the distributed messaging server such as Kafka or, failing that, interacting with another server that will act as an event server. It will convert the actions performed by Stackstorm into RDF/XML graphs, using the basic OSLC concepts explained in Section 3 to convert them into semantic graphs in the form of action or event. This module will send events of type OSLC Events to Kafka and receive graphs of type OSLC Actions from Kafka, alerting of new changes in other DevOps tools. This completes the action and event-based monitoring framework and the OSLC standard. As mentioned above, this work will evaluate the architecture needed for a specific use case (Stackstorm) and therefore the graph manager will be able to send and receive OSLC Actions and OSLC Events without including any other DevOps tool.
- **OSLC TRS:** It can be understood as another server parallel to the adapter where the logs, in OSLC format, of all the events produced by the adapter, will be stored. In the architecture, it is located outside the adapter module itself, since it is not a database as such, but it should be understood as another server that interacts with the Graph Manager through HTTP Requests and that can be consulted at any time both by the adapter and by users who want to have an overall view of what has happened in the DevOps environment.
- **Kafka:** Through a Producer and a Consumer, it can be understood as the intermediary between DevOps tools. On the one hand, it will receive OSLC Events from the adapter and will be able to generate OSLC Actions to all those adapters that are subscribed to a certain Topic. The creation and deployment of these Kafka instances will be explained in detail in the case study section.

Once the general architecture of the work has been described, as well as the general architecture of the project that frames it, the architecture implemented for the work will be explained in detail, both for the monitoring framework and for the extraction and analysis of social metrics.

4.2 Architecture of StackStorm OSLC Adapter

As mentioned above, this work will focus on a specific use case of the general SmartDevOps architecture explained above. In this case, the importance of the architecture will fall on the Stackstorm-OSLC adapter, as well as the different modules that form it.

On the one hand, an instance of Stackstorm will be deployed with its architecture, as explained in section 2. Stackstorm allows storing a log with the events that occur in Stackstorm, which in the case of the project refers to the events that are made to the rules (HTTP CRUD). For the project, the initial architecture of Stackstorm has been slightly modified, so that a MongoDB ReplicaSet cluster has been chosen, consisting of a primary node and two secondary nodes. In this way, active monitoring can be carried out more efficiently.

This monitoring module consists of actively listening to any new record that appears in the ReplicaSet cluster. That is, it has avoided the use of passive monitoring based on periodic HTTP requests to Stackstorm to know the status of its local database, and to check if there has been any change or not. This method has been concluded that could overload the system, so active monitoring has been chosen. Active monitoring is understood as the process by which Stackstorm itself notifies the monitoring module of any new record stored in its local database, thus avoiding system overload.

On the other hand, the Stackstorm-OSLC adaptation layer will have a Graph Manager whose functions will be those described in the previous section. On the one hand, it will be in charge of receiving information from the monitoring module about what type of event and at what time it has occurred. Once it receives this information, it will be in charge of generating a graph of the OSLC Event type, which will be sent to the Kafka instance. On the contrary, it will also be the module in charge of receiving OSLC Action from Kafka. When it receives this OSLC Action, it will interact directly with Stackstorm to update, create, query or delete a rule, depending on the type of action it receives. On the other hand, the Graph Manager will be in charge of recording in the OSLC TRS all the events produced by and in the adapter. This OSLC TRS will have an endpoint that can be consulted to obtain a global view of all the events occurring in the adapter.

In this way, it is possible to have monitoring based on Stackstorm OSLC actions and events.

The architecture of Stackstorm OSLC adapter can be seen in the following diagram:

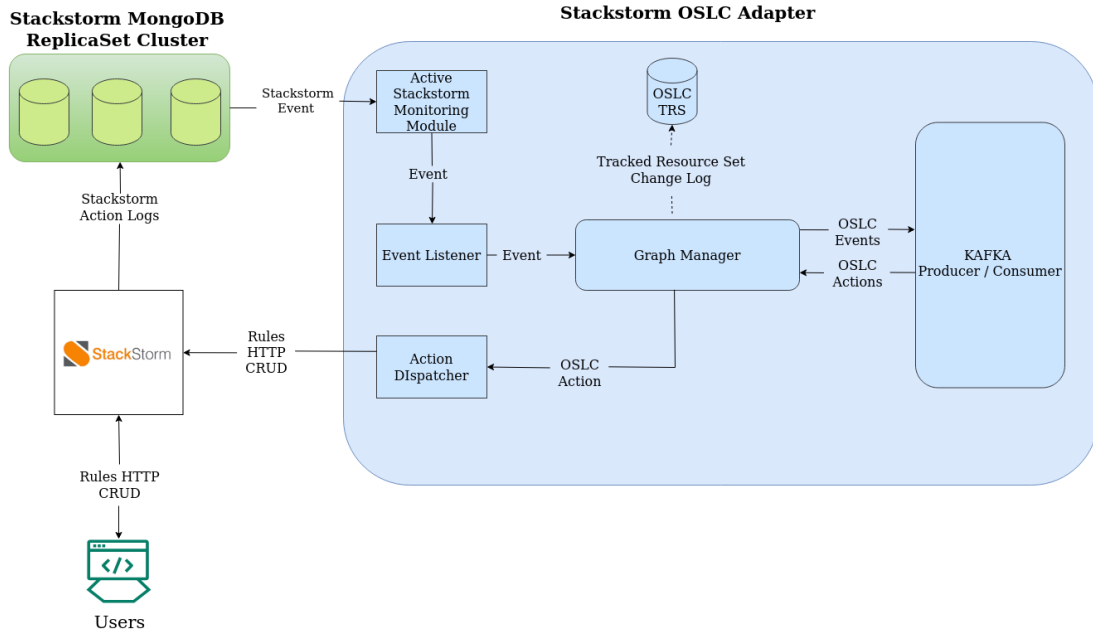


Figure 4.4: Stackstorm OSLC Specific Adapter Diagram

Two scenarios are considered in this diagram:

1. The action is performed directly on the Stackstorm interface. That is, an instance of Stackstorm will be displayed in the scenario and the user will be able to modify, create or delete a rule. This action will be picked up by the monitoring module.
2. An HTTP POST can be sent to the endpoint with the type of action to be performed, which will also be received by the monitoring module.

The idea of the two scenarios is: On the one hand to demonstrate that any manual change in the Stackstorm instance generates an event that will be received by the monitoring module and, on the other hand, to demonstrate that this module will also be able to receive a network of the OSLC Action type and execute this action on Stackstorm. The second scenario can be understood as a simulation of what the full environment (the smartDevOps architecture describes) will look like if a OSLC action is received from another DevOps environment that requires an action on Stackstorm. This allows you to simulate it.

4.3 Architecture of Social Monitoring

In the same way that the Stackstorm-OSLC adapter provides action- and event-based monitoring OSLC, this part of the social monitoring intends to provide new relevant information about the feedback that a given software can have from the users. To this end, a pipeline has been developed that connects the various software used for this entire process, automated by means of a task automator such as Airflow.

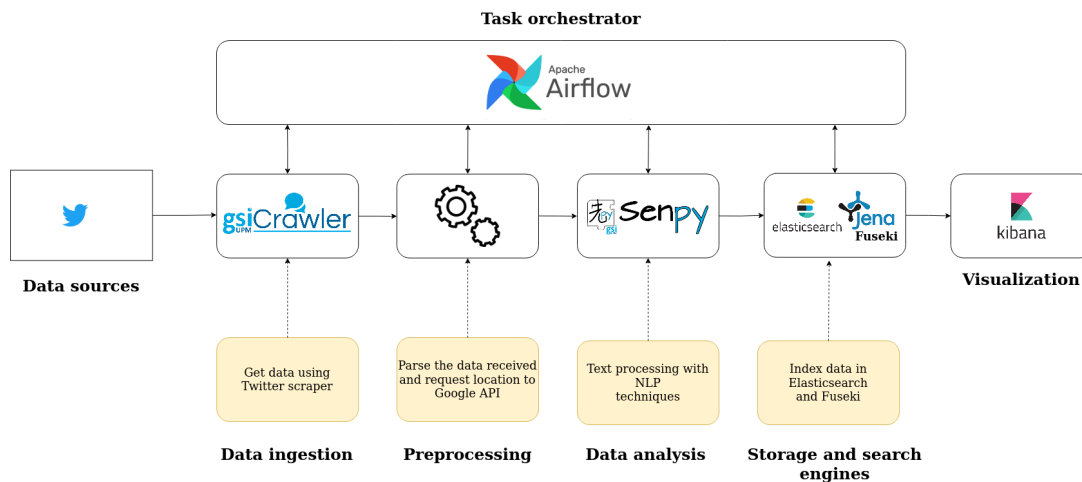


Figure 4.5: Social Metrics Pipeline

Where the software involved in the pipeline and its functions would be:

- **Data ingestion:** The module for extracting social data from the Twitter platform, by using GSICrawler for searching by hashtag. It uses snsrape[42] for this. Once the information is extracted, it enriches it semantically with the Schema ontology.
- **Preprocessing:** This module performs the necessary normalization and adaptation operations on the captured data. Its main objective is to adapt the ingested data to a common format for subsequent analysis. For this purpose, it performs two main processes: NLP preprocessing and geolocation of social network messages. In this case, the GSITK[43] library is used for NLP preprocessing, which, among other things, eliminates stop-words. The location data for each entry is represented in a standardized format so that mapping tools can use it. In the case of Twitter as a data source, to translate the location information to its geographic coordinates, a request is made to the Google API.

- **Data Analysis:** The analysis of the extracted textual data to enrich it, obtain more information and discover patterns is a key feature of the presented intelligent system. The data analysis module performs such analysis, adding value to the data. Senpy is a framework for developing, integrating and evaluating web services for sentiment and emotion analysis and, in general, for text analysis. The text analyses performed in this module extract emotional, psychological and moral information from the text. To extract these annotations, we use the LIWC dictionary and Moral Foundations Dictionary (MFD).
- **Storage and search engines:** The role of the ElasticSearch database is to serve as a central storage for all processed data, allowing its accessibility and visualization. Therefore, once the data is indexed, Kibana is used for the visualization of the results due to its native integration with ElasticSearch. As an additional storage submodule, we use Apache Jena Fuseki. As an endpoint of SPARQL, Fuseki can be understood and accessed by humans and machines using semantic queries. The returned data can be exported in JSON and Comma Separated Value (CSV) formats. In this way, all this data will be stored and analyzed in Fuseki to allow semantic searches and in Elasticsearch to allow its visualization through Kibana.
- **Visualization:** The objective of this module is to provide a control panel for users based on components that can be used for customized and interactive data visualizations. For this project Kibana is used, which will communicate with the data indexed in Elasticsearch. By interacting with these components, the user can modify the filters, and the component communicates the filter change to the rest of the visualization. These Kibana Dashboards can be as complex as you want them to be, depending on the data stored and displayed.
- **Task orchestrator:** This module is responsible for orchestrating all services through the use of pipelines, which are a series of interdependent tasks that are executed in order. Each task is defined by its input (its dependencies), its computation and its output. For this purpose, Airflow is used to orchestrate the different tasks between modules. Each task communicates with the external service that performs actions on the data and transmits them in the pipeline. It ensures that there are no failures when sequencing the tasks.

In this way, the complete collection and processing of metrics from different sources is achieved. On the one hand, an architecture based on actions and events has been developed for Stackstorm monitoring using the OSLC standard. On the other hand, an architecture

based on a task orchestrator such as Airflow has been developed to facilitate, through the use of a pipeline, the extraction and analysis of social metrics through Twitter. In this way, a much broader view of the feedback that a given software may have from users is achieved.

The development of each part involved in this pipeline will be defined in detail in the following section.

Prototype implementation

This section will explain the methodology and development carried out for each of the parties involved in the project. The objective of this section is not to analyze in depth the developed code but to explain the different parts that compose the architecture and the steps followed for the development. The whole adapter is programmed in Python and will be available in Gitlab¹.

On the one hand, as mentioned above, a monitoring framework based on actions and events has been developed using semantic technologies and the OSLC standard. For this purpose, the deployment of the scenario and all the parts involved will be explained, as well as the programming languages or external software used for the development of the Stackstorm-OSLC adapter.

On the other hand, a pipeline based on the Airflow task orchestrator has been developed to implement the whole process of extracting and analyzing social metrics, using tools from the Intelligent Systems Group. For this part, we will also explain everything necessary for its local deployment and all the technical aspects that have been implemented on this software to adapt it to the use case of the project.

¹<https://lab.gsi.upm.es/valvarez/stackstorm-oslc-adapter/>

5.1 Scenario deployment

In this project, there are several tools that are part of the architecture, so several aspects of the deployment have had to be modified to make it centralized and simple for the developer.

On the one hand, as mentioned above, an instance of Stackstorm will be raised with its specific architecture. In this case, the Docker version will be used². For the Stackstorm deployment, several parts of docker-compose.yml have been modified to convert its MongoDB database, formed by a single node, into a ReplicaSet cluster, formed by two secondary nodes and a primary node. In this way, it is intended that the Stackstorm changelog is stored in a distributed and more efficient way, as well as having the cluster correctly monitored by Python functions, which will be explained later.

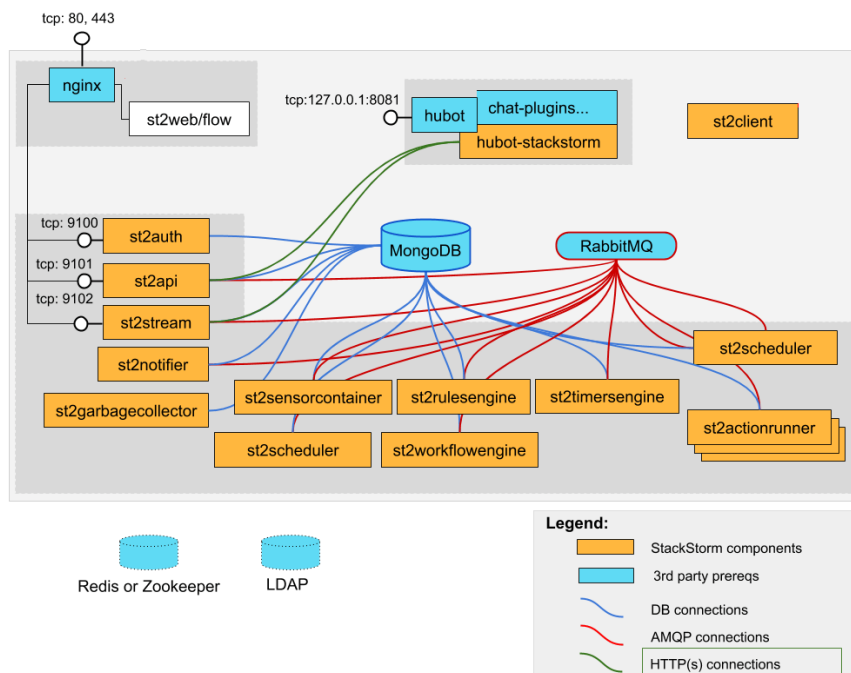


Figure 5.1: Stackstorm MongoDB Architecture

²<https://github.com/StackStorm/st2-docker>

From the previous diagram, we will be interested in modifying the way in which the logs are distributed in MongoDB. Specifically, two new MongoDB containers have been created with a predefined hostname. As a relevant technical aspect, it is worth mentioning that a script has been created that automates the whole process; from the deployment of the containers to the creation of the ReplicaSet cluster, as well as deploying the rest of the containers that form the architecture of the Stackstorm architecture.

Listing 5.1: Part of code from the DockerCompose file for MongoDB ReplicaSet deployment.

```
mongo0:
  image: mongo:4.2
  restart: on-failure
  hostname: "fd6bd9c7f88c"
  networks:
    - private
  volumes:
    - stackstorm-mongodb:/data/db
    - ./mongo/rs-init.sh:/scripts/rs-init.sh
  entrypoint: [ "/usr/bin/mongod", "--replSet", "rs", "--bind_ip_all"]

mongo1:
  image: mongo:4.2
  restart: on-failure
  hostname: "9c27972b5baa"
  networks:
    - private
  volumes:
    - stackstorm-mongodb1:/data/db
  entrypoint: [ "/usr/bin/mongod", "--replSet", "rs", "--bind_ip_all"]
  ports:
    - 27018:27018

mongo2:
  image: mongo:4.2
  restart: on-failure
  hostname: "04fe511d95d3"
  networks:
    - private
  volumes:
    - stackstorm-mongodb2:/data/db
  entrypoint: [ "/usr/bin/mongod", "--replSet", "rs", "--bind_ip_all"]
  ports:
    - 27019:27019
```

The following Bash code has been developed for the creation of the cluster (rs-init.sh). This script is executed on the primary node:

Listing 5.2: Script for the creation of a Replica Set of MongoDB.

```
#!/bin/bash

mongo <<EOF
rs.initiate({
  "_id": "rs",
  "version": 1,
  "members": [
    {
      "_id": 0,
      "host": "some_hostname:27017",
      "priority": 3
    },
    {
      "_id": 1,
      "host": "some_hostname:27017",
      "priority": 2
    },
    {
      "_id": 2,
      "host": "some_hostname:27017",
      "priority": 2
    }
  ]
})

rs.status();
EOF
```

On the other hand, to create everything on the same stage, the deployment of Zookeeper and Kafka has been included in the Stackstorm deployment. This Kafka container will connect to the primary node of the MongoDB cluster to communicate and include the topics.

In this way, the entire scenario is intended to be accessed, including the Stackstorm and Kafka instance, using a simple command line.

Listing 5.3: Part of code from the DockerCompose file for KAFKA deployment.

```
st2_zookeeper:
  image: confluentinc/cp-zookeeper:latest
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
  ports:
    - 22181:2181

st2_kafka:
  image: confluentinc/cp-kafka:latest
  depends_on:
    - zookeeper
  ports:
    - 29092:29092
  environment:
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://
                                localhost:29092
```

Stackstorm-OSLC adapter will be explained in detail in the following section, but it is worth mentioning by way of introduction that it is based on the Python Flask library, so different endpoints will be created for the different OSLC resources to be consulted by means of HTTP requests. This adapter is fully developed in Python, and Docker Compose will also be used for its deployment. The Kafka instance has been developed using its Python client³ and communicates with the adapter through HTTP requests

Finally, the deployment of the sentiment extraction and analysis pipeline consists of several virtualized services hosted on different servers of the Intelligent Systems Group (ETSIT-UPM), as well as all the changes required for this use case. That is, the changes that have been made to the software that makes up the pipeline will be explained, as well as where they are deployed in production and how to deploy them locally.

In conclusion, the idea of the project is to be easily deployable using Docker Compose and Python. The monitoring module requires a parallel process, but a script has been created to automate this process. For the rest of the services, such as the analysis of social metrics, different tools already deployed on the servers of the Intelligent Systems Group have been used and will be explained in detail in the following points.

³<https://pypi.org/project/kafka-python/>

5.2 Stackstorm OSLC Adapter Case Study

In this section, we will describe the technical aspects of the development of the Stackstorm-OSLC adapter as well as all its components.

The backbone of the adapter has been developed using Flask and Cookiecutter.

On the one hand, Flask is a Python framework for creating web servers in the MVC pattern[44]. With its use, it is intended to deploy a set of endpoints to which HTTP requests can be made, as part of the Linked Data principles. This is necessary for the adapter since the OSLC standard, as explained in previous sections, requires URIs to identify the different OSLC resources such as the Service Provider or the URIs for creating or querying resources (Stackstorm rules). For communication of the adapter with the Stackstorm instance, use is made of the Stackstorm API itself, as well as its Python client.

On the other hand, Cookiecutter is an open source library for building coding project templates[45]. In this case, it has been used to create the entire structure that will support the project⁴.

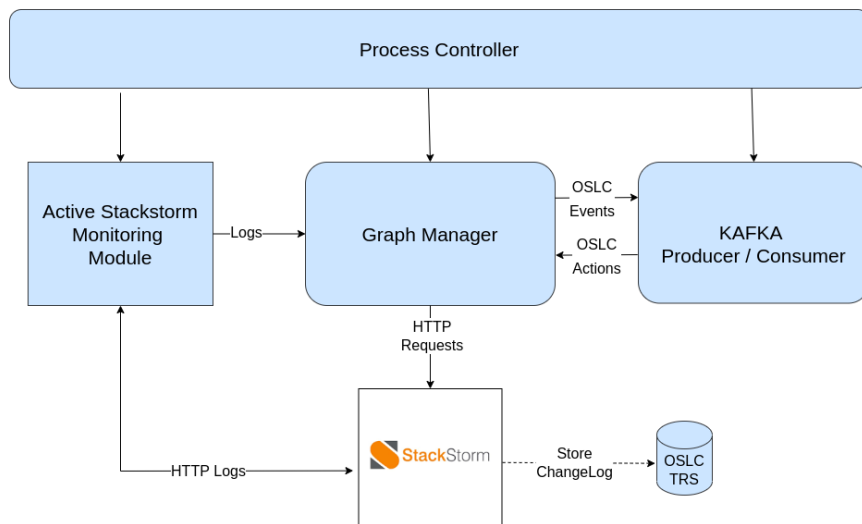


Figure 5.2: Stackstorm OSLC adapter modules

The adapter as a whole will consist of three processes that are started when the adapter is deployed. On the one hand the process controller refers to the docker-compose with which the scenario is raised, as well as all the listening processes of each module.

⁴<https://github.com/cookiecutter-flask/cookiecutter-flask>

This controller will raise three modules: the monitoring module that will be listening for events in Stackstorm, the Graph manager that will be in charge of managing the RDF graphs with RDFLib among others and the Kafka instance that will be used to receive and send actions and events of the OSLC type.

As mentioned above, Flask is used to create the adapter as a web server that has numerous endpoints to handle HTTP requests. These endpoints are necessary to maintain the Linked Data philosophy of exchanging data using URIs and HTTP. Each of them will be associated with a Python function developed to process incoming HTTP requests. The base URI will be `http://localhost:5000/service/..` and it will be necessary to build the rest of the endpoints.

The following diagram reflects the most important endpoints of the project, as well as the relationship with external tools such as Kafka or Stackstorm and the response obtained when a query is made. For example, to obtain a complete list of the list of resources, an HTTP GET must be made to the endpoint `baseURI/serviceProvider/1/changeRequests`.

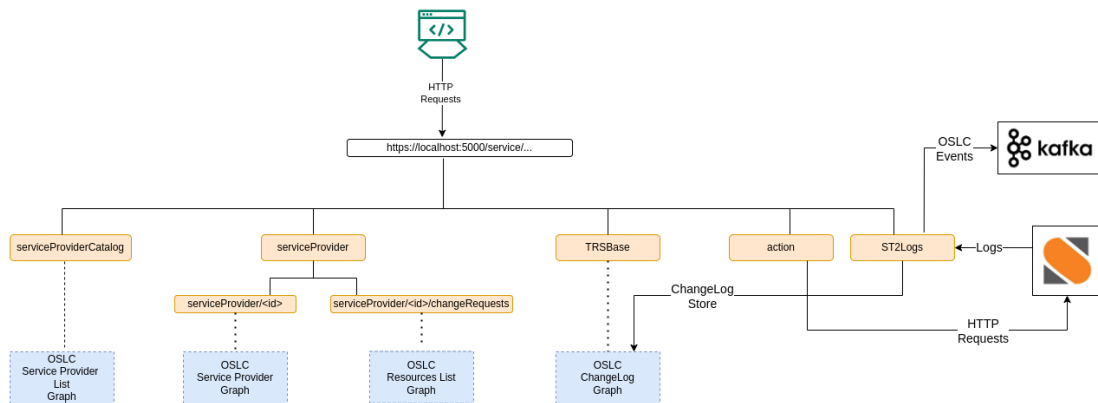


Figure 5.3: Stackstorm-OSLC endpoints

These endpoints are associated with a series of HTTP requests (GET, POST, PUT) via the Python function . When an HTTP request arrives at one of these endpoints, the function is called, and the request processes. These functions will be explained in detail in the Graph Manager.

5.2.1 Monitoring module

The MongoDB change stream was used for the development of the monitoring module. As mentioned above, the changes that occur in Stackstorm are stored in an internal MongoDB database, to which two other secondary nodes have been added for the creation of a ReplicaSet. Change streams allow you to listen for changes to your MongoDB database. This functionality allows you to build applications that can respond immediately to changes in data in real time[46].

The objectives of this module are:

- Listen to Stackstorm events by actively listening by listening for new records in its local database.
- Send these logs to the ST2 endpoint for processing.

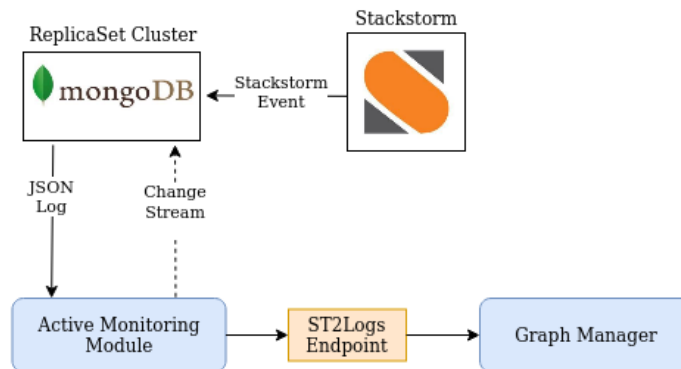


Figure 5.4: Monitoring module diagram

For example, when a rule is updated, created, or deleted, logs of this type will be stored in a local collection from its MongoDB database:

Listing 5.4: Example of an update rule log from the Stackstorm MongoDB

```

{"_id": {"_data": "<some_int>"}, "operationType": "update", "clusterTime":
  {"$timestamp": {"t": 1650549724, "i": 2}}, "ns": {"db": "st2", "coll":
    "rule_d_b"}, "documentKey": {"_id": {"$oid": "<some_rule_oid>"}, "
    updateDescription": {"updatedFields": {"context": {"user": "st2admin"},
      "enabled": false}, "removedFields": ["tags"]}}
  
```


These logs are sent to a new adapter endpoint where these logs will be processed to discover the type of operation that has occurred in Stackstorm.

That is, the **'operationType'** field and **rule id** is obtained and, depending on the type of operation, an OSLC event or another will be created and sent by the adapter to Kafka.

In this way, the changes that occur in the Stackstorm rules are monitored in real time and in a simple and intuitive way.

These changes are processed and send into the Graph Manager for trigger the production of OSLC Events depending on the type of operation produced. This part of the adapter will be explained in detail in the graph manager, since this function called in the adapter when HTTP requests arrive at the ST2Logs endpoint is an important part for the conversion of these logs into OSLC Events to be sent to the Kafka instance.

5.2.2 Graph Manager

It is the central part of the adapter, so in this section we will have to go into the code in more depth to explain how it works. One aspect to note before going into detail on the graph manager implementation is that there is a clear difference for development purposes between the OSLC resources and the Stackstorm rules themselves.

On the one hand, the OSLC resources will be the graphs that define the rules and will be stored in the so-called store. On the other hand, the rules as such will be Stackstorm's answers to a query. In conclusion, the graph manager will be formed by everything related to the semantic world, OSLC resources, RDF/XML graphs, etc. and on the other hand with everything related to its interaction with the Stackstorm API.

The Graph Manager module refers to the module that is in charge of:

- Receiving the Stackstorm logs from the monitoring module.
- Defining Stackstorm as OSLC resources.
- Managing the RDF graph with SPARQL queries.
- Interact with the Stackstorm instance via the HTTP request when a OSLC Action is delivered.
- Sending OSLC Events to Kafka instance when a change in Stackstorm has been made.

- Store the ChangeLog in the OSLC TRS.

The purpose of this section is to explain the whole design and development process, without going into the code in depth. The basis of the Graph Manager resides in the endpoints that have been deployed using Flask, indicated in the previous point. HTTP requests sent to each of these endpoints will be processed in one way or another. Not all endpoints support all types of HTTP requests.

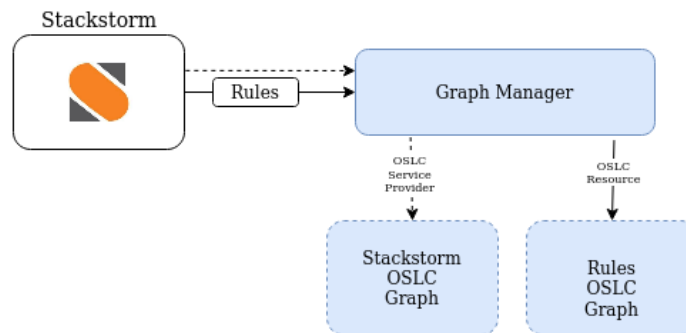


Figure 5.5: Graph Manager Stackstorm-OSLC

For example, the OSLC Service Provider can be queried by means of an HTTP GET to the endpoint defined for this purpose. This request will be processed by a Python function that will perform the semantic adaptation to the OSLC Service Provider, using RDFLib. RDFLib is a pure Python package to work with RDF[47].

With this function it is possible to obtain a representation of the semantic graph in n3 of the OSLC Service Provider, with the resources indicated in the OSLC CORE such as the QueryCapability or the CreationFactory, in the form:

Listing 5.5: Stackstorm representation as a OSLC Service Provider

```

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix ns1: <http://open-services.net/ns/core#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://localhost:5000/service/serviceProviders/1> a ns1:ServiceProvider ;
  ns1:details "http://localhost/api" ;
  ns1:service [ a ns1:Service ;
    ns1:creationFactory [ a ns1:CreationFactory ;
      ns1:creation <http://localhost:5000/service/
        serviceProviders/1/changeRequests> ;
    ] ;
  ] ;

```

```

        ns1:label "Creation Factory" ;
        ns1:resourceType <http://open-services.net/ns/cm#
            ChangeRequest> ] ;
    ns1:domain <http://open-services.net/ns/cm#> ;
    ns1:queryCapability [ a ns1:QueryCapability ;
        ns1:label "Query Capability" ;
        ns1:queryBase <http://localhost:5000/service/
            serviceProviders/1/changeRequests> ;
        ns1:resourceType <http://open-services.net/ns/cm#
            ChangeRequest> ] ] ;
    dcterms:created "2022-05-20T18:47:14.137525"^^xsd:dateTime ;
    dcterms:description "OSLC Stackstorm adapter" ;
    dcterms:identifier 1 ;
    dcterms:title "OSLC Stackstorm adapter" .

```

Similarly, the resource listing can be obtained via an HTTP GET to the endpoint indicated by the Query Capability (as indicated in the OSLC CORE specifications). The following representation shows a rule, semantically modeled with their own ontology, using OSLC.

Listing 5.6: Stackstorm rules representation as a OSLC Resources

```

@prefix ns1: <http://localhost:5001/ns/st2_oslc#> .
@prefix ns2: <http://open-services.net/ns/core#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://localhost:5000/service/serviceProviders/1/changeRequests/1> a <http
://open-services.net/ns/cm#ChangeRequest> ;
    ns1:actionRef "chatops.post_result" ;
    ns1:ruleId "6287c5e09880031168707668"^^rdf:XMLLiteral ;
    ns1:rulePack "chatops" ;
    ns1:ruleRef "chatops.notify" ;
    ns1:ruleStatus true ;
    ns1:ruleTitle "notify"^^rdf:XMLLiteral ;
    ns1:triggerRef "core.st2.generic.notifytrigger" ;
    ns1:triggerType "core.st2.generic.notifytrigger" ;
    ns2:serviceProvider <http://localhost:5000/service/serviceProviders/1>
.

```

In this way, the fundamental concepts of OSLC have been developed in order to associate it with the main concepts of Stackstorm.

Another important part of the Graph Manager will be its ability to interact with Stackstorm, once it receives an action. When Kafka receives certain OSLC Events will be registered under a topic, a network of the type OSLC Action with the action to be performed will be sent to all the tools subscribed to that topic.

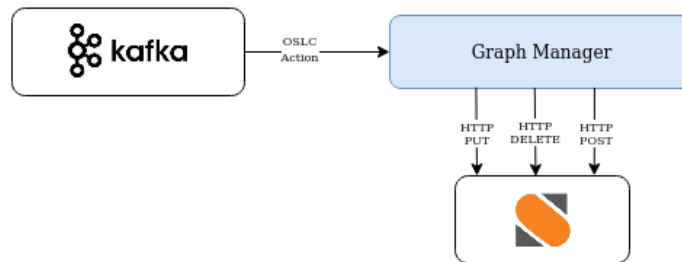


Figure 5.6: Graph Manager interaction with Kafka

This part of the architecture is outside the project frameworks, since no new tools will be included, but an endpoint has been deployed to simulate it. In this case, the Graph Manager will be able to receive a graph of the OSLC Action type similar to the one it would receive from Kafka and will be able to act directly against the Stackstorm API to perform the appropriate action.

For example, let us say the adapter receives an N3 graph to update a rule, of the form:

Listing 5.7: OSLC Action representation in N3 for updating a Stackstorm rule

```

@prefix ns2: <http://localhost:5001/ns/st2_oslc#> .
@prefix ns23: <http://localhost:5002/ns/oslc_actions#> .

[] a <http://localhost:5000/service/UpdateRuleAction>,
    ns23:Action ;
    ns2:ruleId "62761dfae8668ff90c4bec74" .
  
```

This graph will be processed by the Python function associated with the action endpoint. This function will extract the type of action to which it refers, by means of a SPARQL query:

Listing 5.8: SPARQL Query for extracting the OSLC Action type

```

query_action = """
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX oslc_actions: <http://open-services.net/ns/actions#>
    SELECT ?type
    WHERE {
        ?s rdf:type ?type .
    } """

```

In this case, as it is an action of the Update type, which refers to when a rule is activated/deactivated, the function in charge of updating the rules in the deployed Stackstorm instance will be called. This function extracts from the graph the id of the rule with another SPARQL query and interacts with the Stackstorm API to send an HTTP PUT with the id of the rule to be activated/deactivated.

The same procedure will be used for actions of the CreateRule or DeleteRule type also developed in this project. In the general framework, the adapter must be able to receive OSLC Actions and act against any DevOps tool, and at the same time it must be able to update the list of OSLC resources stored in its store.

Finally, the graph manager must be able to receive events from the monitoring module and translate them into OSLC Events to send them to the Kafka instance, which will be explained in the next section. As seen in the description of the monitoring module, with each new change the log is sent to the ST2Logs endpoint where it is processed and everything mentioned above is done.

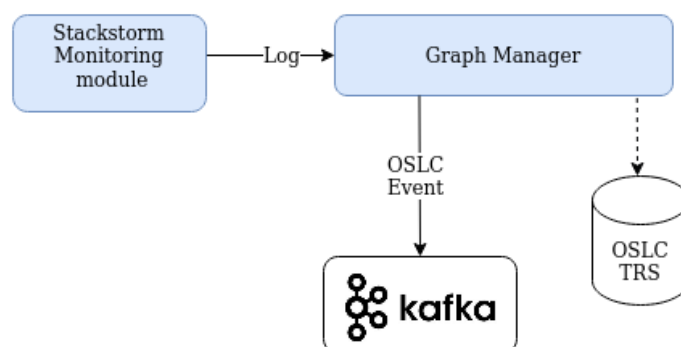


Figure 5.7: Graph Manager ST2Logs endpoint

This log that is sent from the monitoring module, when an event occurs in Stackstorm, to the ST2Logs endpoint is processed to extract what type of action has occurred.

In the same way, this event is stored in the OSLC TRS so that the user or developer can consult via HTTP GET all the log of events that have occurred in the adapter. Whenever a change occurs, the resource listing will have to be updated, so that it reflects the actual Stackstorm situation and not a previous state.

5.2.3 Kafka instance

Another important part of the adapter will be the Kafka instance. As mentioned above, once the Graph Manager processes the logs coming from the monitoring module, the Kafka module comes into play. Kafka consists of a Producer and a Consumer. The Producer is the one that receives from the Graph Manager the semantic data related to OSLC Events and stores them under a certain topic, for example 'st2event'. Once this is done, the topic is stored in different sections called 'partitions' that will be consulted by the Consumer.

The idea of this implementation is that different DevOps tools subscribe to the 'st2event' topic and that, when a new information flow is added, a message is sent to all of them alarming them of new changes in Stackstorm. For this, semantic networks of the OSLC Action type will be generated, which will be received by all the other adapters.

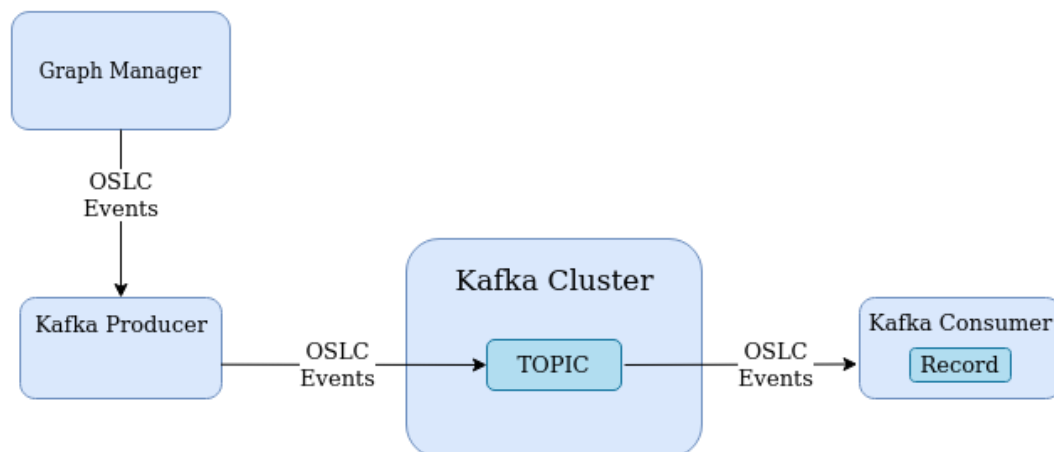


Figure 5.8: Kafka Producer/Consumer Diagram

At the development level, this is done in the function in charge of handling HTTP requests arriving at the ST2Logs endpoint, as shown in the diagrams in this section.

That is, when an event occurs in Stackstorm, an HTTP request is sent to the ST2Logs endpoint with the log, the type of event produced is extracted (activation/deactivation of a rule, deletion or creation of a rule) and an OSLC Event of that type is generated, to be sent by the Kafka Producer under the topic 'st2event'.

5.3 Social metrics Case Study

In this section, we will explain the more technical aspects of the development of the pipeline for the extraction and analysis of social metrics through Twitter. It is not intended to go in-depth into the code of the software itself, since it has been developed by the Intelligent Systems Group, outside this project. However, we will explain all the technical aspects that have been modified or added to this software to adapt it to the specific use case of this project. There will be a subsection for each part of the pipeline.

5.3.1 Airflow

Airflow will be responsible for orchestrating all tasks through the pipeline, so that they are performed in an automated and related manner. The Python airflow library has been used for its development. All of these tasks will be described in detail in the following sections. For this project, we have reused the code of Óscar Araque⁵, from the Intelligent Systems Group, adapting it to this use case.

The execution of the pipeline is handled by the Airflow DAG, which is responsible for setting the main parameters as follows.

Listing 5.9: DAG code example for a daily execution

```
@dag(
    start_date=pendulum.datetime(2022, 2, 15, 5, tz="UTC"),
    tags=["smartdevops"],
    catchup=True,
    schedule_interval="@daily",
    max_active_runs=1,
)
```

⁵<https://gsi.upm.es/es/about-us/people?view=member&task=show&id=137>

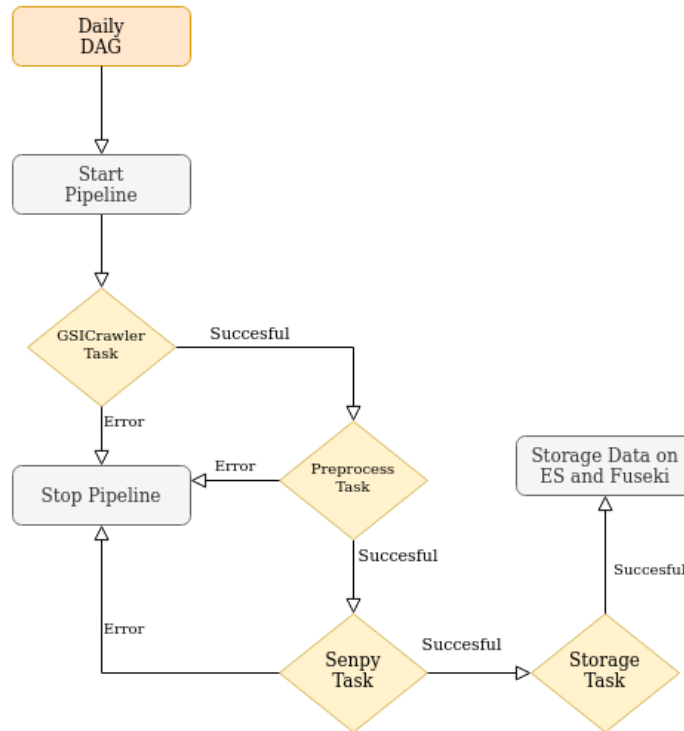


Figure 5.9: Pipeline Flowchart

Therefore, Airflow takes care of the execution of each task in the order indicated in the diagram. If an error occurs in any of them, the pipeline stops, but caches the result of those that have been successfully completed. In this way, if the pipeline is executed again, it will start from the task that failed in the first place. This pipeline and each of its tasks, is entirely developed in Python.

5.3.2 GSICrawler

As mentioned in previous sections, for the extraction of social metrics from Twitter, GSI-Crawler⁶ was used. In the initial development of GSICrawler, the library *twint*⁷ was used, which had certain limitations, so for this project, we have chosen to use *snsrape*⁸. *Snsrape* is an open source that provides scrapers for different social sources such as Facebook or Mastodon. In this case, we were only interested in using it for Twitter.

⁶<https://crawler.gsi.upm.es>

⁷<https://github.com/twintproject/twint>

⁸<https://github.com/JustAnotherArchivist/snsrape>

GSICrawler is deployed on one of the servers of the Intelligent Systems Group, as a Docker container, and is accessible at the previous endpoint. The changes required to implement snsrape in GSICrawler have been to map the results obtained by the library to the schema ontology, used to describe blogs or web pages.

The idea of the project is to obtain all tweets that contain hashtags for a certain software. In this case and for the project demo, tweets containing the hashtag #VisualStudioCode will be extracted.

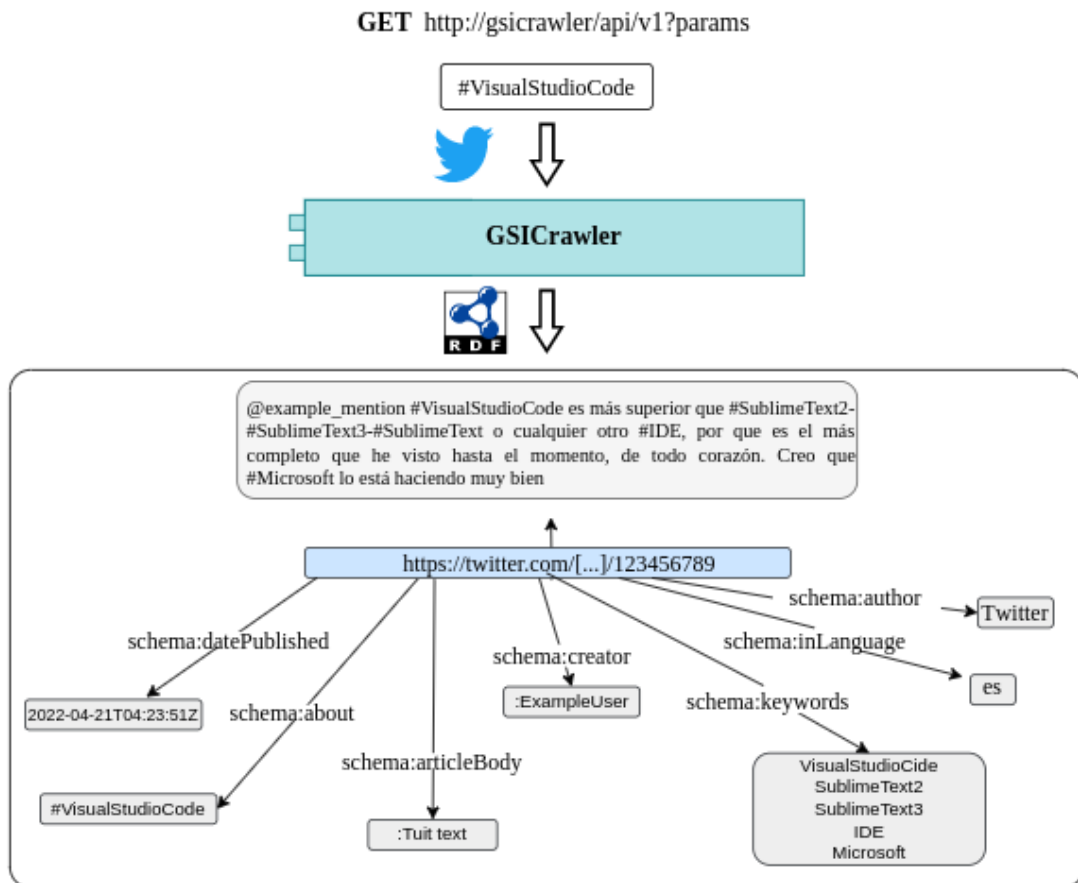


Figure 5.10: GSICrawler input-output example

5.3.3 Preprocessing

These data obtained through GSICrawler are pre-processed using natural language with NLTK and GSITK⁹.

This task is developed in Python like all the others, and this idea consists of the following:

- Remove punctuation
- Tokenize words with the NLTK library
- Removing tweets in languages other than English or Spanish
- Removing stopwords such as "hashtag", "url", "user", etc. with GSITK
- Obtaining the tuit location by using Google Geocode API

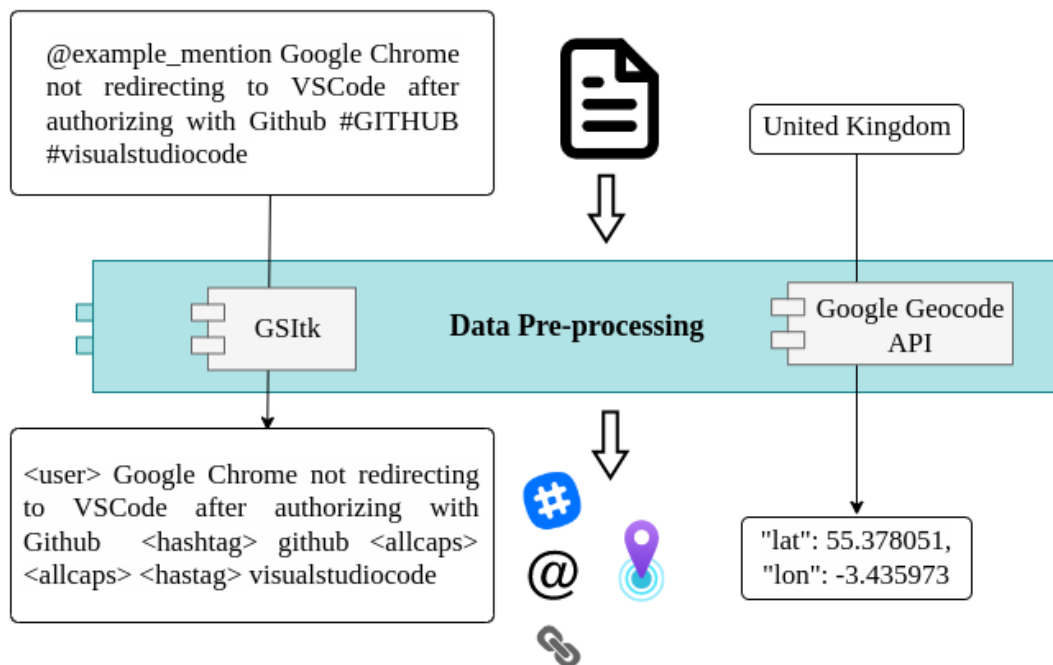


Figure 5.11: Data Preprocessing Example

⁹<https://github.com/gsi-upm/gsitk>

5.3.4 Senpy

As mentioned above, sentiment analysis that provides a semantic layer is performed using Senpy¹⁰. Senpy is a framework developed by Intelligent Systems Group that allows for the creation of sentiment analysis web services easily, quickly, and using a well-known API. Senpy services use semantic vocabularies (e.g., NIF, Marl, Onyx) and formats (turtle, JSON-LD, RDF/XML).

LIWC and MFD dictionaries were used for the project. In this case, only the LIWC dictionary in Spanish and English was needed, since these are the languages chosen for the demonstration of this work. For this purpose, both plugins have been created for Senpy, as indicated in the documentation¹¹.

The LIWC dictionary is made up of 2,290 words and word stems (Tausczik & Pennebaker, 2010). Each word or word stem defines one or more word categories or subdictionaries. LIWC categories are hierarchically ordered. For example, all anger words, by definition, will be classified as negative emotions and general emotion words. The result obtained enriches and contextualizes the textual data input, providing a detailed analysis that includes emotions, thinking styles, social concerns, and personal drivers.

In terms of MFD, it is used with the LIWC program, facilitating its implementation. The MFD was developed to operationalize moral values in text. This dictionary provides information on the proportions of virtue words and vice words for each moral foundation in a text corpus (Graham et al., 2009). It is made up of 336 words and word stems that are classified into the five moral dimensions: care / harm, justice / cheating, loyalty / betrayal, authority / subversion, and sanctity / degradation. All moral dimensions are measured in the text, and their results are aggregated to form a unified view of the data. It is not intended to go in depth into the developed code, so the documentation and Github repository are left as a reference for any reader who wants to dig into the creation and deployment of Senpy plugins. An example of Senpy can be found in the endpoint described in the footer¹².

An example of the output of both plugins for the tweet used in the above example could be:

¹⁰<https://github.com/gsi-upm/senpy>

¹¹<https://senpy.readthedocs.io/en/latest/>

¹²<https://senpy.gsi.upm.es>

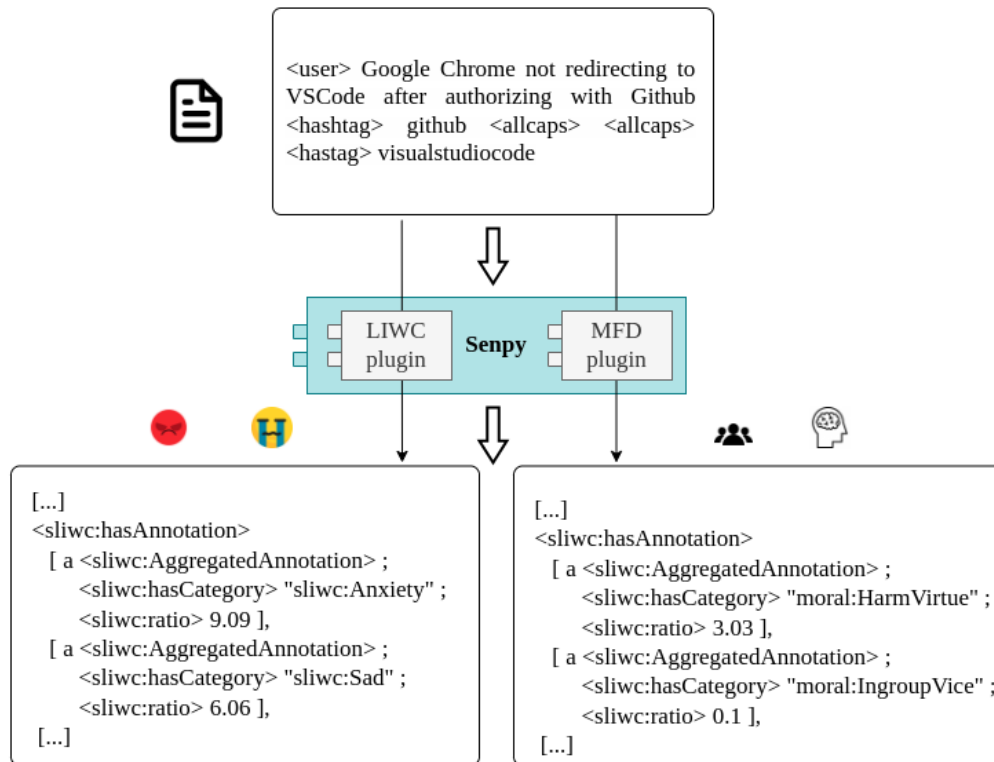


Figure 5.12: Output example with Senpy and LIWC, MFT plugins

5.3.5 Data Storage and Visualization

Once these social metrics are extracted with GSICrawler, pre-processed with NLTK and GSITK and analyzed and semantically enriched with Senpy, they are stored in several endpoints that will have different functions. On the one hand, these data are stored in an instance of Elasticsearch, under the SmartDevOps index, and can be visualized with Kibana.

This Kibana-developed project dashboard is accessible to any user at the endpoint indicated in the footer¹³. Several important aspects of the dashboard are worth noting. On the one hand, the number of tweets obtained, after passing through the filters mentioned above, containing the hashtag #VisualStudioCode, as well as a graph indicating the time of publication of the tweets containing this hashtag, and the most used. The user will also be able to filter these data with the different filters at the top.

¹³<https://dashboard-smartdevops.gsi.upm.es>

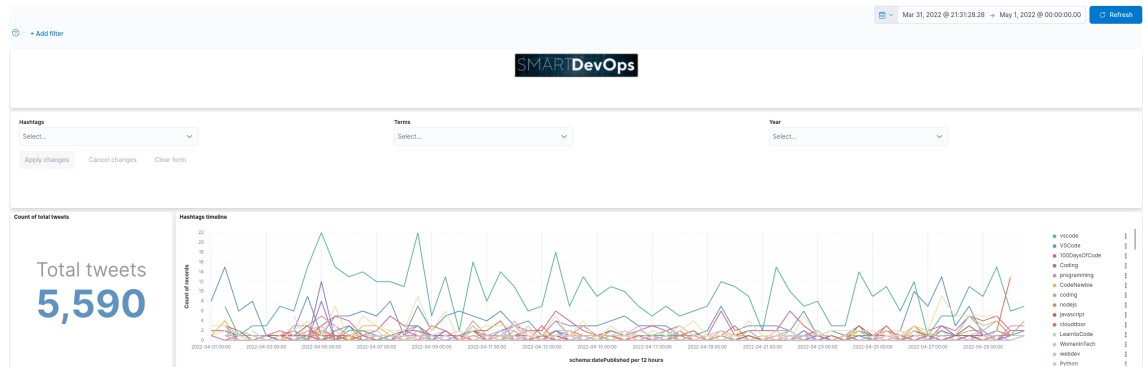


Figure 5.13: Kibana Dashboard with number of tweets and timeline for hashtags

Other interesting data have been added, such as the most used hashtags and the number of times they have been used.



Figure 5.14: Kibana Dashboard with popular hashtags

The next thing displayed will be the results obtained from the Senpy sentiment analysis. This visualization will be done using pie charts whose intention is to divide the results into user-understandable categories.

- **Affect words:** It refers to the identification of the message as positive or negative, which could give an interesting overview of, e.g. a new functionality or release added by a certain software.
- **Social words:** Refers to information content of a social nature, such as references to gender-segregated referents such as friends, relatives, etc.
- **Personal concerns:** Refers to the identified personal use that users give, in this case, to Visual Studio Code. From work use (majority) to leisure use, as well as sectarian (extremist) use. Economic uses could include commercial use or use in an advertising tweet.

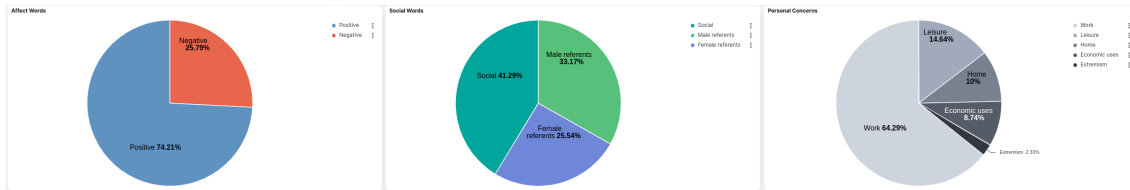


Figure 5.15: Kibana Dashboard with Sentiment Analysis

Finally, as mentioned in previous sections, with the preprocessing of the extracted data, the location of the tweet is extracted through the Google API. Note that not all tweets have location. This visualization is carried out using an interactive heat map.

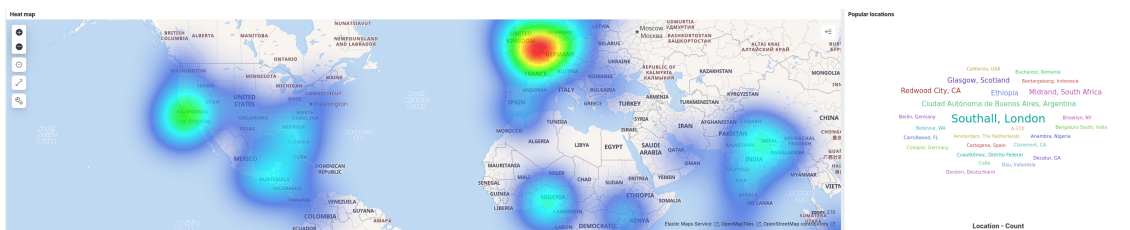


Figure 5.16: Kibana Dashboard with Data Location

However, the data will be stored in a Fuseki deployed by the Intelligent Systems Group¹⁴. The function of this Fuseki is to store all the semantic data, to be queried by the user through an SPARQL endpoint.

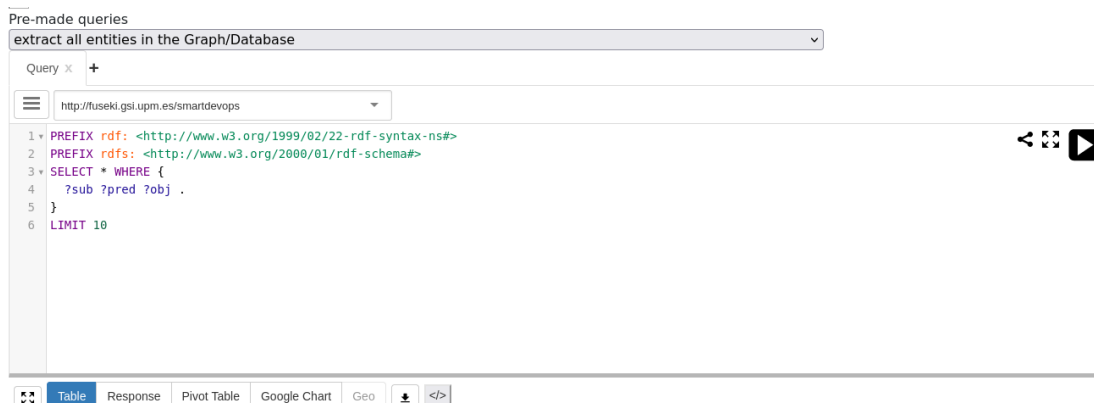


Figure 5.17: SPARQL Endpoint

The idea of storing the data in both data storage is to allow the user to have a basic visualization through Kibana and to make more complex queries to Kafka by using a SPARQL endpoint.

¹⁴<https://fuseki.gsi.upm.es>

Conclusions and Future Work

This chapter will describe the goals achieved by the master thesis, as well as what is left for future work.

6.1 Conclusions

In this project, we have developed a monitoring framework based on actions and events using the OSLC standard and the advantages of Linked Data and the semantic web. For this purpose, we have designed our semantic model to define Stackstorm rules. In this way, a functional prototype has been created that uses the benefits of the semantic web to interrelate different DevOps tools under a common vocabulary and in an interoperable way.

On the one hand, a Stackstorm-OSLC adapter has been developed, capable of generating OSLC Events when a change occurs in any Stackstorm rule, and send it into a Kafka instance. For this purpose, it has been necessary to create a module for monitoring the deployed Stackstorm instance, by actively monitoring its local database, looking for new records that reflect any change, as well as the Graph Manager module where all the graphs and everything related to the processing and exchange of semantic data will be managed.

Finally, a Kafka cluster has been deployed and designed to receive from the Graph Manager OSLC Events, when an event occurs in Stackstorm, and to produce OSLC Actions to the rest of the tools subscribed to this topic. This last part is outside the scope of the project since it is part of a larger project, as indicated in this report.

On the other hand, to complement the monitoring framework based on OSLC Events and Actions, a pipeline has been developed and deployed to extract and analyze social metrics through Twitter, to get an overview of the feedback from users regarding a given software, in this case, Visual Studio Code. In this way, a global view is achieved and added value is added to the framework, by analyzing sentiment data extracted from social sources. This data is also semantically enriched, so the use of ontologies has been a key part of the project, as explained.

6.2 Achieved Goals

The achieved goals for this project are the following:

- **Interaction with Stackstorm via HTTP Requests:** All the necessary functions have been developed to interact with the deployed instance of Stackstorm, using its own API.
- **Development of a semantic model for Stackstorm rules:** A semantic model has been created to define Stackstorm rules, using OSLC Core classes and other domains as well as attributes defined for the rule structure. This allows the management of these rules by the Graph Manager.
- **Development of a monitoring framework for Stackstorm's internal database:** Taking advantage of the fact that any change produced in Stackstorm is stored in a local MongoDB, an active monitoring module has been developed to listen to events registered in this database. On the other hand, to increase its productivity, the number of MongoDB nodes has been increased from a single node to a ReplicaSet cluster consisting of a primary node and two secondary nodes.
- **Development of a Graph Manager for Stackstorm-OSLC adapter:** All the necessary functions have been developed for all the semantic enrichment of Stackstorm rules, following the OSLC standard. On the one hand, the Python OSLC client and RDFLib have been used to work with RDF/XML graphs and allow the adapter to

create its own OSLC Event or OSLC Action graphs depending on the action performed in Stackstorm. This Graph Manager is in charge of interacting with Stackstorm, modifying its rules, depending on the type of operation indicated in the OSLC Action graph it receives, as well as creating OSLC Event type graphs and sending them to Kafka when a manual change occurs in any of these rules.

- **Deployment and development of a Kafka instance:** A Kafka instance has been deployed, consisting of a Kafka Producer and a Kafka Consumer. On the other hand, the necessary functions have been developed, through the kafka-python library, to be able to receive OSLC Events type graphs and generate OSLC Actions type graphs. The part that other DevOps tools can receive these OSLC Actions by subscribing to a certain Kafka topic is outside the scope of this project.
- **Deployment and development of a pipeline for visualization of social metrics:** It has been developed, using software already deployed and created by the Intelligent Systems Group and adapting it to the use case of the project, a social metrics monitoring pipeline capable of extracting data from the Twitter social network on a given hashtag, perform an analysis of feelings and emotions and be displayed on a Kibana dashboard. In turn, a SPARQL endpoint has been deployed to allow the user to perform complex searches taking advantage of semantic data enrichment.

6.3 Future Work

Having explained the conclusions, once the action- and event-based monitoring framework has been designed and developed, it is worth mentioning the future work. Part of the future work will be carried out under the SmartDevOps project, of which this work is a part. These future points could be encompassed in the following:

- **Extend the semantic model for Stackstorm:** For this work, the Stackstorm rules have been modeled semantically. Although rules are an important part of this tool, they are not the only one. Within Stackstorm there are other components such as actions, triggers, packs, which can be semantically modeled in the future in order to realize a complete Stackstorm-OSLC adapter.
- **Incorporation of new DevOps tools into the system:** As described in the SmartDevOps project definition, the future idea is to achieve interoperability between DevOps tools. That is, having deployed for this work a Stackstorm-OSLC adapter

capable of generating OSLC events and receiving OSLC Actions, the idea will be to incorporate into the complete architecture other OSLC adapters from other tools that are capable of the same. Each adapter will have its own TRS, as well as its own Graph Manager. The idea will be to connect a diverse set of tools through Kafka, through subscriptions to topics, using a common vocabulary such as OSLC. Kafka will generate OSLC Actions every time it receives OSLC Events from a certain tool, under a certain topic, and will send it to those subscribed to that topic. The Graph Manager of each adapter will be in charge of interacting with the tool to make the necessary modifications.

- **Implement automation:** The process described in the previous section is intended to achieve the automation of events and actions. For example, let's say there is a certain tool subscribed to the "st2_topic" topic. If a rule is activated in Stackstorm, an OSLC Event will be generated to Kafka under the topic "st2_topic". Kafka will generate an OSLC Action to all the tools subscribed to the topic and will generate their actions directly against their own APIs or Python clients or whatever is needed, in an automated way, thanks to the OSLC standard.
- **Deployment of the system in production environment:** The entire system will be deployed in production environments of the Intelligent Systems Group. This environment will have a virtualized instance of Stackstorm, of Kafka, as well as the OSLC adapter developed for the tools that require it.

Project Impact

This is the description of the appendix.

A.1 Introduction

DevOps methodologies are being used exponentially on a daily basis in most technology companies. These methodologies consist of tools that are not always easily interoperable due to their own configurations, so using a common standard so that they can communicate in a fluid and interoperable way can be a great advantage. For this purpose, this system uses OSLC and the advantages of Linked Data to create a framework for monitoring software metrics or social metrics. This section will explain the impact of the developed system.

A.2 Social Impact

At the social level, the goal of the system is to provide an alternative to the interconnection of DevOps tools through the use of the semantic web and Linked Data. Developing systems

capable of communicating through a common vocabulary would allow systems to respond more fluidly to changes in user demand, so that users can receive a higher quality of service. On the other hand, the use of social networks such as Twitter has become a daily routine for millions of users, so analyzing the information flowing in this network for a particular software or update can bring considerable advantages to the developers of these softwares.

A.3 Economic Impact

In the field of economics, this system proposes changes that would be of great relevance at the economic level. On the one hand, if automation of communication processes between different DevOps tools is achieved for different life cycles, a much greater optimization would be obtained in the complete life cycle of a software product. Companies would be able to manage their infrastructures in a much simpler way, being able to manage periods of high demand and placing themselves at the forefront of the rest of the companies in their sector.

A.4 Environmental Impact

The system itself does not require large computational requirements, but the idea of using Kafka would be to create an instance in several systems to decentralize the sending of messages. Therefore, depending on the volume of DevOps tools incorporated into the system as well as the volume of data extracted from Twitter from the appropriate hashtags, more computational capacity may be required. Storage servers such as NFS must also be added to the system for persistence. Because of this, it is necessary to carry out an energy transition that uses less polluting energy and more renewable energy sources. On the other hand, it is also necessary to use green computing techniques to reduce pollution, not only with this project, but with so many others.

A.5 Ethical Implications

The ethical implications refer in this case to user privacy in the deployment of the social metrics extraction pipeline. In the case of the project, private user information is not public. Certain data are only used for the visualization module, without an external user being able to link sensitive data to identify a specific user. If this platform were to be used

by a private company and they wanted to publish all extracted data, users would have to give their explicit consent.

Project Budget

B.1 Introduction

In this section, all project costs, both human and economical, as well as the resources used during development will be discussed

B.2 Human Resources

This project has been developed within the framework of the SmartDevOps research project, for which the owner of this work is contracted as a lab technician in the Intelligent Systems Group. This master's thesis, which corresponds to the university master's degree in networks and telematic services, corresponds to 12 ECTS, which would be 360 hours of work. Taking into account that at 8 € each hour of work, it can be concluded that the cost per person has been 2.880 €.

B.3 Physical Resources

Servers from the Intelligent Systems Group have been used for this project. First, a desktop computer with the following characteristics:

- **CPU:** Intel Core i5-10300H 2.50GHz
- **RAM:** 16 GB
- **Disks:** 500 GB - HDD
- **Operative System:** Ubuntu 21.04 LTS
- **Price:** 500€

On the other hand, other servers in the group have been used, such as a NAS server as well as the production Kubernetes cluster. This cluster consists of a master node and two secondary nodes. This NAS server has the following features:

- **CPU:** Intel Xeon E5-2603 1.70GHz
- **RAM:** 8 GB
- **Disks:** 27T (only a small part available for the development of this project)
- **Operative System:** Ubuntu 20.04 LTS
- **Price:** 10.000 €

On the other hand, the 3 nodes mentioned above have been used for the Kubernetes cluster owned by GSI, if each computer has the following characteristics:

- **CPU:** Intel Xeon Silver 4210 CPU 2.20GHz, x86_64 architecture.
- **RAM:** 16 GB
- **Disks:** 500GB
- **Operative System:** Ubuntu 21.04 LTS
- **Price:** 400 €

B.4 Total Costs

Based on the above, the total project costs could be 15.080 €.

Stackstorm ontology

For the creation of the Stackstorm model with OSLC, Protégé has been used as explained in the previous sections. The purpose of this annex is to show all the classes and properties that compose the model. The documentation has been done with Widoco¹ and, at the moment it is deployed on a local Apache server but it will be included in a web server to make it accessible.

¹<https://dgarijo.github.io/Widoco/>

APPENDIX C. STACKSTORM ONTOLOGY

This version:
http://localhost:5001/Stackstorm_OSLC/01

Latest version:
http://localhost:5001/Stackstorm_OSLC

Authors:
Victor Álvarez Provencio

Imported Ontologies:
<http://open-services.net/ns/autof/>
<http://open-services.net/ns/core/>

Download serialization:
[Format: JSON-LD](#) [Format: RDF/XML](#) [Format: N-Triples](#) [Format: TTL](#)

License:
[License](#) [License same goes here](#)

Visualization:
[Download with WebVOWL](#)

Cite as:
Retrieved from: http://localhost:5001/Stackstorm_OSLC/01

language [en](#)

Abstract

Stackstorm OSLC Ontology

Table of contents

- 1. Introduction
 - 1.1 Namespace declarations
- 2. Stackstorm OSLC Ontology: Overview
- 3. Stackstorm OSLC Ontology: Description
- 4. Cross-reference for Stackstorm OSLC Ontology classes, object properties and data properties
 - 4.1 Classes
 - 4.2 Data Properties
- 5. References
- 6. Acknowledgments

1. Introduction

This is a place holder text for the introduction. The introduction should briefly describe the ontology, its motivation, state of the art and goals.

back to [ToC](#)

2. Stackstorm OSLC Ontology: Overview

This ontology has the following classes and properties.

back to [ToC](#)

Classes

[create rule action](#) [delete rule action](#) [rule](#) [rule action](#) [stackstorm](#) [update rule action](#)

Data Properties

[action_ref](#) [rule_action](#) [rule_created](#) [rule_deleted](#) [rule_enabled](#) [rule_id](#) [rule_name](#) [rule_pack](#) [rule_title](#) [rule_trigger](#) [rule_updated](#) [trigger_ref](#) [trigger_type](#)

3. Stackstorm OSLC Ontology: Description

This is a placeholder text for the description of your ontology. The description should include an explanation and a diagram explaining how the classes are related, examples of usage, etc.

back to [ToC](#)

4. Cross-reference for Stackstorm OSLC Ontology classes, object properties and data properties

This section provides details for each class and property defined by Stackstorm OSLC Ontology.

back to [ToC](#)

4.1. Classes

[create rule action](#) [delete rule action](#) [rule](#) [rule action](#) [stackstorm](#) [update rule action](#)

[create rule action](#)

IRI: http://localhost:5001/Stackstorm_OSLC#CreateRuleAction

has super-classes

[rule action](#)

back to [ToC](#) or [Class ToC](#)

[delete rule action](#)

IRI: http://localhost:5001/Stackstorm_OSLC#DeleteRuleAction

has super-classes

[rule action](#)

back to [ToC](#) or [Class ToC](#)

[rule](#)

IRI: http://localhost:5001/Stackstorm_OSLC#Rule

has super-classes

[automation plan](#)

is in domain of

[rule action](#) ^{dp}, [rule_created](#) ^{dp}, [rule_enabled](#) ^{dp}, [rule_id](#) ^{dp}, [rule_name](#) ^{dp}, [rule_pack](#) ^{dp}, [rule_title](#) ^{dp}, [rule_trigger](#) ^{dp}, [rule_updated](#) ^{dp}

back to [ToC](#) or [Class ToC](#)

[rule action](#)

IRI: http://localhost:5001/Stackstorm_OSLC#RuleAction

has super-classes

[action](#)

has sub-classes

[create rule action](#) ^c, [delete rule action](#) ^c, [update rule action](#) ^c

back to [ToC](#) or [Class ToC](#)

stackstorm ^c	back to ToC or Class ToC
IRI: http://localhost:5001/Stackstorm_OSLC#Stackstorm	
has super-classes service provider ^c	
update rule action ^c	back to ToC or Class ToC
IRI: http://localhost:5001/Stackstorm_OSLC#UpdateRuleAction	
has super-classes rule action ^c	

4.2. Data Properties

action ref	rule action	rule created	rule deleted	rule enabled	rule id	rule name	rule pack	rule title	rule trigger	rule updated	trigger ref	trigger type
action ref ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#actionRef												
has super-properties rule action ^{dp}												
has range string												
rule action ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#ruleAction												
Action to be performed by a rule												
has sub-properties action ref ^{dp}												
has domain rule ^c												
has range string												
rule created ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#ruleCreated												
has domain rule ^c												
has range boolean												
rule deleted ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#ruleDeleted												
has range boolean												
rule enabled ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#ruleEnabled												
has domain rule ^c												
has range boolean												
rule id ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#ruleId												
Rule ID assigned randomly by Stackstorm												
has domain rule ^c												
has range token												
rule name ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#ruleName												
Name to identify a StackStorm rule												
has domain rule ^c												
has range string												
rule pack ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#rulePack												
has domain rule ^c												
has range string												
rule title ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#ruleTitle												
has domain rule ^c												
has range string												
rule trigger ^{dp} <div>back to ToC or Data Property ToC</div>												
IRI: http://localhost:5001/Stackstorm_OSLC#ruleTrigger												
Definition of the trigger that will lead to the execution of a rule in Stackstorm												
has sub-properties trigger ref ^{dp} , trigger type ^{dp}												
has domain rule ^c												

APPENDIX C. STACKSTORM ONTOLOGY

rule updated ^{dp}	back to ToC or Data Property ToC
IRI: http://localhost:5001/Stackstorm_OSLC#ruleUpdated	
has domain rule	
has range boolean	
trigger ref ^{dp}	back to ToC or Data Property ToC
IRI: http://localhost:5001/Stackstorm_OSLC#triggerRef	
has super-properties rule.trigger ^{dp}	
has range string	
trigger type ^{dp}	back to ToC or Data Property ToC
IRI: http://localhost:5001/Stackstorm_OSLC#triggerType	
has super-properties rule.trigger ^{dp}	
has range string	
Legend	back to ToC
 Classes  Data Properties	

The authors would like to thank Silvio Peroni for developing LODE, a Live OWL Documentation Environment, which is used for representing the Cross Referencing Section of this document and Daniel Garijo for developing Widoco, the program used to create the template used in this documentation.

Bibliography

- [1] “Devops: A complete guide,” Jun. 2021. [Online]. Available in: <https://www.ibm.com/cloud/learn/devops-a-complete-guide>. [Accessed: 2022-03-13]
- [2] OSLC Primer, “Why OSLC?” publication Title: OSLC: Why OSLC? Type: OSLC: Why OSLC? [Online]. Available in: <https://open-services.net/why/>. [Accessed: 2022-04-06]
- [3] Javatpoint, “DevOps Lifecycle,” publication Title: DevOps Lifecycle Type: DevOps Lifecycle. [Online]. Available in: <https://www.javatpoint.com/devops-lifecycle>. [Accessed: 2022-04-06]
- [4] C. Ebert, G. Gallardo, J. Hernantes, y N. Serrano, “DevOps,” *IEEE Software*, vol. 33, no. 3, pp. 94–100, May 2016.
- [5] “DevOps Methodology | Stages of DevOps Methodology & Tools,” Jan. 2020. [Online]. Available in: <https://www.educba.com/devops-methodology/>. [Accessed: 2022-04-12]
- [6] “StackStorm Overview — StackStorm 3.6.0 documentation.” [Online]. Available in: <https://docs.stackstorm.com/overview.html>. [Accessed: 2022-04-06]
- [7] Hewlett Packard Enterprise, “¿Qué es la infraestructura como código? \textbar Glosario.” [Online]. Available in: <https://www.hpe.com/es/es/what-is/infrastructure-as-code.html>. [Accessed: 2022-04-06]
- [8] Airflow Documentation, “Apache Airflow Documentation — Airflow Documentation,” publication Title: Apache Airflow Documentation Type: Apache Airflow Documentation. [Online]. Available in: <https://airflow.apache.org/docs/apache-airflow/stable/index.html>. [Accessed: 2022-04-06]
- [9] “Apache Airflow,” Oct. 2021. [Online]. Available in: <https://datascientest.com/es/todo-sobre-apache-airflow>. [Accessed: 2022-04-12]
- [10] “Architecture Overview — Airflow Documentation.” [Online]. Available in: <https://airflow.apache.org/docs/apache-airflow/stable/concepts/overview.html>. [Accessed: 2022-04-12]
- [11] “¿Qué Es MongoDB?” [Online]. Available in: <https://www.mongodb.com/es/what-is-mongodb>. [Accessed: 2022-04-19]
- [12] “¿Qué es el sharding en MongoDB? ¿Cómo funciona el sharding en MongoDB? | ramoncarrasco.es.” [Online]. Available in: <https://www.ramoncarrasco.es/es/content/es/kb/141/que-es-el-sharding-en-mongodb-como-funciona-el-sharding-en-mongodb>. [Accessed: 2022-04-19]

- [13] “MongoDB Architecture.” [Online]. Available in: <https://www.mongodb.com/mongodb-architecture>. [Accessed: 2022-04-19]
- [14] “Apache Kafka.” [Online]. Available in: <https://kafka.apache.org/>. [Accessed: 2022-05-17]
- [15] “Apache Kafka.” [Online]. Available in: <https://kafka.apache.org/documentation/>. [Accessed: 2022-05-17]
- [16] Elastic, “Búsqueda open source: los creadores de Elasticsearch, el ELK Stack y Kibana \textbar Elastic.” [Online]. Available in: <https://www.elastic.co/es/>. [Accessed: 2022-04-06]
- [17] —, “¿Qué es Elasticsearch?” publication Title: Elastic. [Online]. Available in: <https://www.elastic.co/es/what-is/elasticsearch>. [Accessed: 2022-04-06]
- [18] Studytonight, “Understanding Elasticsearch Architecture - Studytonight.” [Online]. Available in: <https://www.studytonight.com/elasticsearch/understanding-elasticsearch-architecture>. [Accessed: 2022-04-06]
- [19] Elastic, “Kibana: Explora, visualiza y descubre datos,” publication Title: Elastic. [Online]. Available in: <https://www.elastic.co/es/kibana>. [Accessed: 2022-04-06]
- [20] Elastic, “Logstash: Recopila, parsea y transforma logs,” publication Title: Elastic. [Online]. Available in: <https://www.elastic.co/es/logstash>. [Accessed: 2022-04-06]
- [21] DBPedia, “Semantic Web,” publication Title: DBPedia: Semantic Web Type: DBPedia: Semantic Web. [Online]. Available in: https://dbpedia.org/page/Semantic_Web. [Accessed: 2022-04-06]
- [22] W3C, “OWL - Semantic Web Standards,” publication Title: OWL - Semantic Web Standards Type: OWL - Semantic Web Standards. [Online]. Available in: <https://www.w3.org/OWL/>. [Accessed: 2022-04-06]
- [23] “Vista General del Lenguaje de Ontologías Web (OWL).” [Online]. Available in: <https://www.w3.org/2007/09/OWL-Overview-es.html#s3>. [Accessed: 2022-04-12]
- [24] “Linked data como modelo de datos | datos.gob.es.” [Online]. Available in: <https://datos.gob.es/es/noticia/linked-data-como-modelo-de-datos>. [Accessed: 2022-04-12]
- [25] Ontotext, “What is RDF?” publication Title: Ontotext. [Online]. Available in: <https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf/>. [Accessed: 2022-04-06]
- [26] “WebProtege - Protege Wiki.” [Online]. Available in: <https://protegewiki.stanford.edu/wiki/WebProtege>. [Accessed: 2022-04-12]
- [27] “What is GSI Crawler? — GSI Crawler 1.0 documentation.” [Online]. Available in: <https://gsicrawler.readthedocs.io/en/latest/gsicrawler.html>. [Accessed: 2022-04-13]
- [28] J. F. Sánchez-Rada, O. Araque, y C. A. Iglesias, “Senpy: A framework for semantic sentiment and emotion analysis services,” *Knowledge-Based Systems*, vol. 190, p. 105193, Feb. 2020. [Online]. Available in: <https://linkinghub.elsevier.com/retrieve/pii/S0950705119305313>. [Accessed: 2022-04-23]

-
- [29] “What is Senpy? — Senpy documentation.” [Online]. Available in: <https://senpy.readthedocs.io/en/latest/senpy.html>. [Accessed: 2022-04-13]
- [30] “Apache Jena - Apache Jena Fuseki.” [Online]. Available in: <https://jena.apache.org/documentation/fuseki2/index.html>. [Accessed: 2022-05-16]
- [31] “Software.” [Online]. Available in: <https://gsi.upm.es/es/tecnologias/software>. [Accessed: 2022-04-23]
- [32] “Schema.org - Schema.org.” [Online]. Available in: <https://schema.org/>. [Accessed: 2022-04-23]
- [33] J. F. Sánchez-Rada, O. Araque, y C. A. Iglesias, “Senpy: A framework for semantic sentiment and emotion analysis services,” *Knowledge-Based Systems*, vol. 190, p. 105193, Feb. 2020. [Online]. Available in: <https://linkinghub.elsevier.com/retrieve/pii/S0950705119305313>. [Accessed: 2022-04-23]
- [34] “SLIWC: LIWC dimensions represented as a SKOS taxonomy.” [Online]. Available in: <http://gsi.upm.es:9080/ontologies/participation/sliwc/ns/doc/index-en.html#intro>. [Accessed: 2022-04-24]
- [35] “Morality: MFT concepts represented as a SKOS taxonomy.” [Online]. Available in: <http://gsi.upm.es:9080/ontologies/participation/morality/ns/doc/index-en.html>. [Accessed: 2022-04-26]
- [36] “Get started with developing OSLC applications.” [Online]. Available in: <https://oslc.github.io/developing-oslc-applications/>. [Accessed: 2022-04-13]
- [37] “Oslc specifications.” [Online]. Available in: <https://open-services.net/specifications/>. [Accessed: 2022-04-13]
- [38] “OSLC Primer.” [Online]. Available in: <https://open-services.net/resources/oslc-primer/>. [Accessed: 2022-04-13]
- [39] “OSLC Automation Specification Version 2.1 | Automation - Open Services for Lifecycle Collaboration.” [Online]. Available in: <https://archive.open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.1/>. [Accessed: 2022-05-04]
- [40] G. García Grao y A. Carrera, “Towards an Architecture for Automation of Devops Based on the Oslc Standard,” Social Science Research Network, Rochester, NY, SSRN Scholarly Paper 4020632, Jan. 2022. [Online]. Available in: <https://papers.ssrn.com/abstract=4020632>. [Accessed: 2022-05-05]
- [41] “OSLC Tracked Resource Set Version 3.0. Part 1: Specification.” [Online]. Available in: <https://oslc-op.github.io/oslc-specs/specs/trs/tracked-resource-set.html>. [Accessed: 2022-05-03]
- [42] JustAnotherArchivist, “snsrape,” May 2022, original-date: 2018-09-09T20:16:31Z. [Online]. Available in: <https://github.com/JustAnotherArchivist/snsrape>. [Accessed: 2022-05-10]
- [43] “GSITK project,” Jan. 2022, original-date: 2015-11-12T13:51:10Z. [Online]. Available in: <https://github.com/gsi-upm/gsitk>. [Accessed: 2022-05-10]

BIBLIOGRAPHY

- [44] “Flask Documentation (2.1.x).” [Online]. Available in: <https://flask.palletsprojects.com/en/2.1.x/foreword/#what-does-micro-mean>. [Accessed: 2022-05-13]
- [45] “Cookiecutter.” [Online]. Available in: <https://www.cookiecutter.io/>. [Accessed: 2022-05-13]
- [46] “MongoDB Change Streams with Python.” [Online]. Available in: <https://www.mongodb.com/developer/quickstart/python-change-streams/>. [Accessed: 2022-05-14]
- [47] “Rdflib 6.1.1 documentation.” [Online]. Available in: <https://rdflib.readthedocs.io/en/stable/>. [Accessed: 2022-05-17]
- [48] “TrackedResourceSet 2.0 | Core - Open Services for Lifecycle Collaboration.” [Online]. Available in: <https://archive.open-services.net/wiki/core/TrackedResourceSet-2.0/revision/3244/#Tracked-Resource-Set>. [Accessed: 2022-05-03]