CLOUD FOR DATA-DRIVEN POLICY MANAGEMENT

Project Number: 870675 Start Date of Project: 01/01/2020 Duration: 36 months

D4.2 REUSABLE MODEL & ANALYTICAL TOOLS: SOFTWARE PROTOTYPE 1

Dissemination Level	PU
Due Date of Deliverable	31/10/2020, M10
Actual Submission Date	02/12/2020
Work Package	WP4 Reusable Models & Analytical Tools
Task	All
Type	Demonstrator
Approval Status	Final
Version	V0.8
Number of Pages	p.1 – p.47

Abstract: A description of the software demonstration for the components of the Integrated Data Acquisition and Analytics (DAA) Layer, which provides the analytical capabilities of the PolicyCloud platform, including the API Gateway / orchestrator and the built-in analytical tools.

`

PolicyCloud has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 870675.

Versioning and Contribution History

Version	Date	Reason	Author
0.1	17/11/2020	Initial draft skeleton	Ofer Biran
0.2	22/11/2020	DAA API Gateway and operational Data Repository	Ofer Biran, Oshrit Feder, Pavlos Kranas
0.3	25/11/2020	Enhanced Interoperability & Data Cleaning, Situational Knowledge Acquisition & Analysis, Opinion Mining & Sentiment Analysis	Ofer Biran, Oshrit Feder, María Ángeles Sanguino, Jorge Montero, Argyro Mavrogiorgou, Thanos Kiourtis, George Manias, Nikitas Sgouros
0.4	27/11/2020	Few updates / corrections, Conclusions added.	Ofer Biran, Oshrit Feder
0.5	30/11/2020	Updating per the internal reviews. Reviewers: Rafael del Hoyo, Nikos Achilleopoulos	Ofer Biran
0.6	01/12/2020	Format/References corrections	Ofer Biran
0.7	01/12/2020	Quality Check	Argyro Mavrogiorgou
0.8	01/12/2020	Final version	Ofer Biran

Author List

Organisation	Name
IBM	Ofer Biran, Oshrit Feder, Pavel Kravchenko
ATOS	María Ángeles Sanguino, Jorge Montero
UPRC	Argyro Mavrogiorgou, Thanos Kiourtis, George Manias, Nikos Palamaris, Nikitas Sgouros, Ilias Maglogiannis
LXS	Pavlos Kranas

Abbreviations and Acronyms

Abbreviation/Acronym	Definition
API	Application Programming Interface
DAA	Data Acquisition and Analytics
EC	European Commission
EOSC	European Open Science Cloud
GTD	Global Terrorism Database
ML	Machine Learning
NER	Named Entity Recognition
NLP	Natural Language Processing
PDT	Policy Development Toolkit
POS	Part-of-Speech
REST	Representational State Transfer

Abbreviation/Acronym	Definition
SKA	Situational Knowledge Acquisition

Contents

List of Tables

List of Figures

Executive Summary

This doscument is provided in the context of the first software demonstartor deliverable of PolicyCLOUD, at M11 of the project, and is intended for the reviwers of the software deliverables. It provides a description of the software demonstration for the components of the Integrated Data Acquisition and Analytics (DAA) Layer, which provides the analytical capabilities of the PolicyCloud platform. The components include the DAA API Gateway (responsible for the overall orchestration and the layer API), the built-in analytical tolls - for Data cleaning and interoperability, Situational Knowledge, Opinion Mining & Sentiment Analysis and Social Dynamics & Behavioural Data analysis, and the Operational Data Repository. At this phase of the project the demonstrators are at component level and not integrated in end-to-end scenario, so each component provides a description of its API, specification of functionality, and description of the code and how to test the demonstration.

1 Introduction

1.1 Data Acquisition and Analytics Layer

The Data Acquisition and Analytics (DAA) layer is designed for extensibility and reusability of analytic functions. New analytics functions can be registered into PolicyCLOUD and reused for applying analytics on new and existing registered data sources. The detailed architecture and functionality are provided in D4.1 "Reusable Model & Analytical Tools: Design and Open Specification 1". Figure 1 provides the high-level architecture the DAA layer with the main components and interactions with the exploiters.

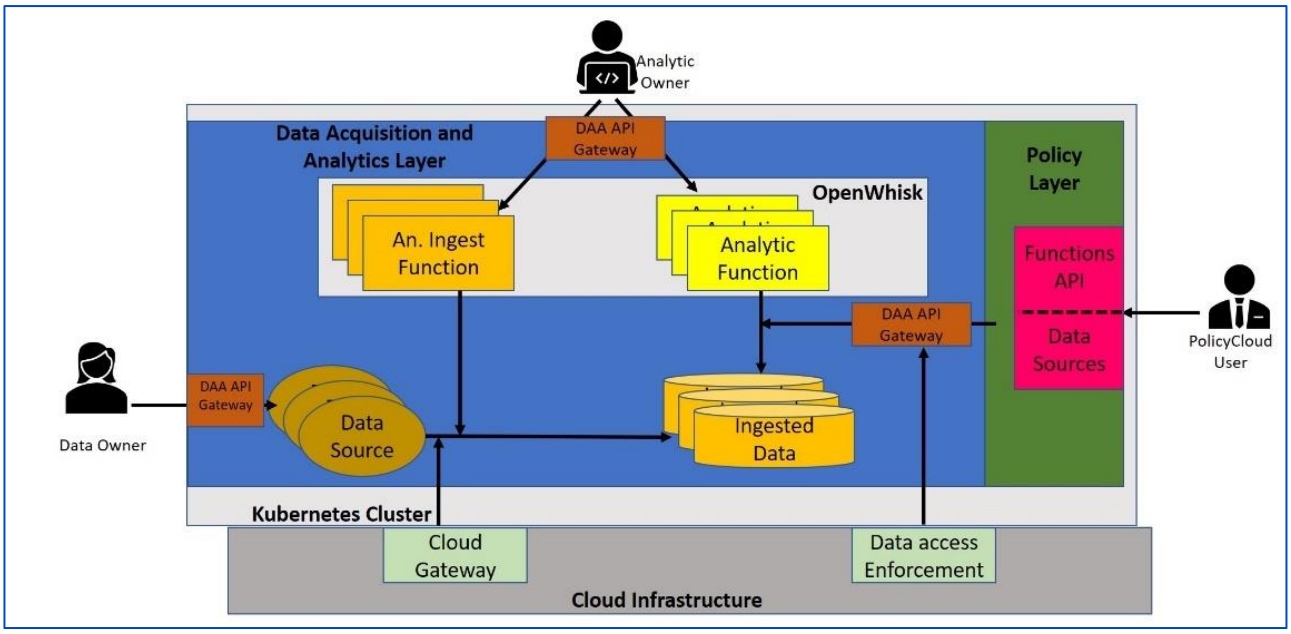


FIGURE 1 - DATA AQUIZITION AND ANALYTICS LAYER

The decided design approach was the serverless model based on the open source OpenWhisk, a distributed serverless platform, where the functions there are activated on demand, either by a direct PolicyCLOUD user request or by event/rule. There are two types of functions:

1. Analytic-ingest function, which are used to apply initial analytic and/or transformation on the data fusion path of data sources.
2. Analytic function, which is activated upon PolicyCLOUD user action invoked through the Policy Layer, to perform an analysis on a specified data source (which was already ingested) to provide analytic results for policy decisions.

The DAA API Gateway component provides the external frontend of the DAA layer, as well as all required orchestration for the function and data source registrations and activation of the API. The other components are the built-in analytics functions for Data Cleaning and Interoperability, Situational Knowledge, Opinion Mining & Sentiment Analysis, and Social Dynamics & Behavioural Data analysis, and the Operational Data Repository component which is provided by the LeanXscale database. All the components (as well as the OpenWhisk cluster and the LeanXscale database) are running above a Kubernetes cluster.

1.2 Summary of Changes

This document is consistent with D4.1 "Reusable Model & Analytical Tools: Design and Open Specification 1" which provided the overall architecture of the Data Acquisition and Analytics layer and the built-in analytical tools. It provides lower details of the API and implementation of the software components that were developed in accordance to the design.

2 Prototype overview

At this phase of the project, the developed software components were not yet integrated to provide end-to-end scenario demonstration – this will be delivered in the second software demonstrator, i.e. D4.4. Thus, each of the components can be run in a stand-alone mode, were stub components are used to demonstrate the component's functionality (e.g. a stub for the Cloud Gateway component for accessing and reading the external data source and populate the Kafka entry topic of the data ingesting path).

As at this point the prototype is presented as stand-alone components, and only sunset of the eventually intended functionality is implemented, the missing (or undefined) parts at this point are marked with "TBD" (e.g. interface parameters that are not exploited now and their format is not decided yet).

2.1 Main components of the prototype

This section provides the description of the components included in the prototype,

2.1.1 DAA API Gateway

The DAA API Gateway developed in Task 4.1 is responsible for the orchestration and integration of all the components of the DAA layer (analytic functions, data sources, data repository) as well as providing the external interface for the layer's exploiters – analytic owners, data source owners, and the Policy layer (through which the end user apply desired analytics). Its roles include forming the proper setup of the registered analytic functions and data sources to ensure the correct data ingest process (applying the required ingest-time analytics/transformation – in both Ingest-now and Streaming modes), and applying the requested analytics on the requested data source. The DAA API Gateway is implemented as a set of serverless functions (in the OpenWhisk cluster), where the state is managed both on the OpenWhisk itself (e.g. analytic functions' metadata) and the database.

2.1.2 Enhanced Interoperability & Data Cleaning

2.1.2.1 Data Cleaning Component

The scope of the Data Cleaning component is to undertake all the processes regarding the data validation and data cleaning of all the incoming heterogeneous data, to the extent possible. The Data Cleaning component provides the interface that implements the data cleaning workflow as documented in deliverable D4.1[1] of the project, in order to be utilized by the rest of the PolicyCLOUD components ensuring data accuracy and consistency of the incoming datasets.

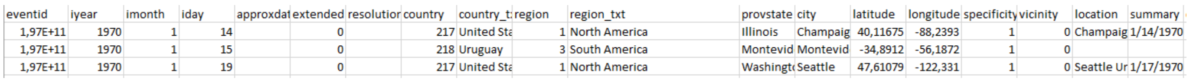
The architecture and design of the Data Cleaning component was documented in D4.1 with the purpose of addressing the volatility of the incoming data information towards the aim of providing data accuracy, consistency and completeness to the PolicyCLOUD platform. Thus, the Data Cleaning component implements all the processes that identify inaccurate or corrupted datasets that may contain inaccurate, incorrect, incomplete or irrelevant data elements and consequently replace, modify or delete these data elements safeguarding the reliability and appropriateness of the incoming data information. The software prototype of the Data Cleaning component was driven by this specification.

To this context, the Data Cleaning component is composed by one (1) main service, namely the DataCleaningService, which in turn consists of four (4) internal services: the ValidationService, the CleaningService, the VerificationService, and the LoggingService. The main service, DataCleaningService, handles all the incoming and outgoing traffic of the DataCleaner component and is the only service exposed to the rest of the platform components. Contrary to the main service, the four (4) internal services are not exposed to the rest of the platform components, whereas the DataCleaningService is interacting with these services internally so as to realize the data cleaning workflow. The provided functionalities of the services are furtherly depicted below.

- **DataCleaningService**: It is the main service of the Data Cleaning component, which is in charge of executing the data cleaning workflow of PolicyCLOUD platform. Since the data cleaning workflow comprises of several sequential steps, each one implemented by one of the internal services of the component, the DataCleaningService is responsible for the orchestration of these internal services as well as for monitoring their execution, and thus providing the execution results to the requestor. In addition to the data cleaning workflow execution, the DataCleaningService is responsible for the implementation of the single interface for data cleaning requests. In deeper detail, the DataCleaningService implements one (1) main function:
    - *initiateCleaning (dataset: File, dataschema: Pandas_schema): Void*: This is the main function that initiates the DataCleaningService. It receives as an input the provided dataset (in csv format), as well as its corresponding data schema (in pandas schema format), being responsible for orchestrating the rest of the internal services towards the execution of the data cleaning workflow.
- **ValidationService**: It is the internal service responsible for the data validation processing of the incoming data. The ValidationService performs a series of validation checks in order to evaluate the conformance to a set of constraints currently integrated in the business logic of the service. The current list of validation rules includes the following:
    - Conformance to specific data type.
    - Conformance to mandatory fields.
    - Conformance to specific value length.
    - Conformance to specific value representation.
    - Conformance to specific value range.
    - Identification of duplicate values for the data elements.
    - Identification of duplicate data elements.

  To this end it should be noted that the list of the validation rules will be furtherly enriched as the project evolves.
  As for its provided functions, the ValidationService implements the following functions:
    - *validateData(dataset: File, dataschema: Pandas_schema): List*: This is the main function that initiates the ValidationService. It receives as an input the provided dataset (in csv format), as well as its corresponding data schema (in pandas schema format), being responsible for identifying and returning the list of errors based on the evaluation of the set of the constraints that have been set. In order to properly work, validateData exploits the following internal functions:
        - *checkString(i: String): Boolean*: This is an internal function validating the conformance to the specific data type.
        - *checkDecimal(d: Decimal): Boolean*: This is an internal function validating the conformance to the specific data type.
        - *checkInt(i: Integer): Boolean*: This is an internal function validating the conformance to the specific data type.
        - *checkMaxDigits(i: Numerical): Boolean*: This is an internal function validating the conformance to the specific value length.
        - *checkRange(i: Numerical, j: Numerical): Boolean*: This is an internal function validating the conformance to the specific value range.

Based on the aforementioned, Figure 2 depicts a snapshot of an uploaded dataset, prior to performing any cleaning action. It should be mentioned that for this dataset, a data schema has been already created including all the validation rules, that each row and column of the dataset should obey to. What is more, it should be noted that this dataset refers to the Scenario that regards the Participatory Policies Against Radicalization (Scenario #1.1), as it is documented in D2.2 [2].



eventid	iyear	imonth	iday	approxdat	extended	resolutior	country	country_t	region	region_txt	provstate	city	latitude	longitude	specificity	vicinity	location	summary
1,97E+11	1970	1	14		0		217	United Sta	1	North America	Illinois	Champaig	40,11675	-88,2393	1	0	Champaig	1/14/1970
1,97E+11	1970	1	15		0		218	Uruguay	3	South America	Montevid	Montevid	-34,8912	-56,1872	1	0		
1,97E+11	1970	1	19		0		217	United Sta	1	North America	Washingt	Seattle	47,61079	-122,331	1	0	Seattle Ur	1/17/1970

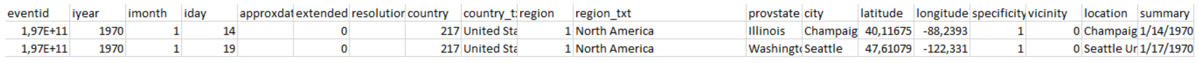
FIGURE 2 - SNAPSHOT OF THE INITIAL UPLOADED DATASET

Following this example, based on the set validation rules, it was described that the "location" column, should not contain any null values. For that reason, the ValidationService highlights this, and reports it to the errors list, which is provided to the CleaningService, for performing the corresponding cleaning actions.

- **CleaningService**: It is the internal service responsible the cleaning processing of the incoming data. The CleaningService eliminates the list of errors identified by the ValidationService by applying all the necessary corrective actions on the data elements marked with errors. Cleaning is performed in an automated way based on a set of actions currently integrated in the business logic of the component. The current list of cleaning actions includes the following:
    - Deletion (drop) of the complete record (row).
    - Replacement of data element's value with the mean value.
    - Replacement of data element's value with the maximum value.
    - Replacement of data element's value with the minimum value.
    - Replacement of data element's value with the most frequent value.

  To this end it should be noted that the list of the cleaning actions will be furtherly enriched as the project evolves.
  As for its provided functions, the CleaningService implements the following functions:
    - *cleanData(errors: List, dataset: File): File*: This is the main function that initiates the CleaningService. It receives as an input the provided dataset (in csv format), as well as the identified list of errors produced by the ValidationService, being responsible for constructing and returning the cleaned dataset. In order to properly work, cleanData exploits the following internal functions:
        - *dropRow(row_num: Integer): Void*: This is an internal function rejecting the complete record (row).
        - *replaceWithMean(column: String, row_num: Integer): Void:* This is an internal function replacing the data element's value with the mean value.
        - *replaceWithMax(column: String, row_num: Integer): Void:* This is an internal function replacing the data element's value with the maximum value.
        - *replaceWithMinimum(column: String, row_num: Integer): Void:* This is an internal function replacing the data element's value with the minimum value.
        - *replaceWithMostFrequent(column: String, row_num: Integer): Void:* This is an internal function replacing the data element's value with the most frequent value.

Based on the aforementioned, in the provided example, the row with the identified error has been dropped (Figure 3). Such thing was requested from the cleaning actions of the specific Scenario, where it was described that in the case that a null value would appear, then the overall row containing this null value should be deleted.



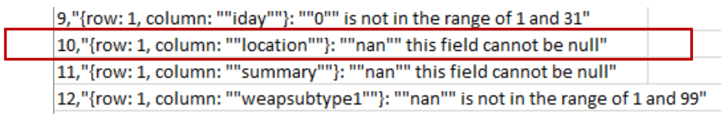
eventid	iyear	imonth	iday	approxdat	extended	resolutior	country	country_t:	region	region_txt	provstate	city	latitude	longitude	specificity	vicinity	location	summary
1,97E+11	1970	1	14		0		217	United Sta	1	North America	Illinois	Champaig	40,11675	-88,2393	1	0	Champaig	1/14/1970
1,97E+11	1970	1	19		0		217	United Sta	1	North America	Washingt(	Seattle	47,61079	-122,331	1	0	Seattle Ur	1/17/1970

FIGURE 3 - SNAPSHOT OF THE CLEANED DATASET

- **VerificationService**: It is the internal service responsible for the verification and evaluation of the corrective actions undertaken by the CleaningService, aiming to ensure the accuracy and the consistency of the updated incoming data according to the PolicyCLOUD platform requirements. In more detail, the VerificationService implements the following main function:
    - *verifyData(): List*: This is the main function that initiates the VerificationService. It receives as an input the provided dataset (in csv format), the cleaned dataset (in csv format), as well as the identified list of errors, being responsible for finally verifying and evaluating the corrective actions in combination with the actual cleaned dataset, and thus providing back the results of the evaluation. In the case that there are not any corrective actions that need to be additionally performed, a null list is returned. On the other hand, the VerificationService returns a list of additional corrective actions that should take place and/ or repeated by the CleaningService.

  Based on the aforementioned, in the provided example, the VerificationService checks and confirms that the CleaningService has successfully performed all the needed cleaning actions, returning a null List since no further corrective actions are needed to take place.
- **LoggingService**: It is the internal service responsible for keeping the records that contain all the identified errors and the corrective actions undertaken, in order to address these errors during the data cleaning workflow execution by the rest of the internal services of the Data Cleaning component. For each execution of the data cleaning workflow a unique record is created and stored in the list of the records kept by the LoggingService. In the current implementation the list of the records is kept in a csv file in the local file system where the LoggingService is running. In order to achieve that, the LoggingService implements the following main function:
    - *createLog(errors: List): Void*: This is the main function that initiates the LoggingService. It receives as an input the identified list of errors, being responsible for creating a new record (i.e. log file) based on the information provided as input. This new record is appended as a new csv file.

  Based on the current example, the LoggingService reports and keeps a record of the errors and cleaning actions that the CleaningService successfully performed, where in that case, it reports that the row containing the null value in the location column was dropped (Figure 4).



```
9,"{row: 1, column: ""iday""}: ""0"" is not in the range of 1 and 31"
10,"{row: 1, column: ""location""}: ""nan"" this field cannot be null"
11,"{row: 1, column: ""summary""}: ""nan"" this field cannot be null"
12,"{row: 1, column: ""weapsubtype1""}: ""nan"" is not in the range of 1 and 99"
```

FIGURE 4 - SNAPSHOT OF THE IDENTIFIED ERRORS IN THE ERROR LIST

2.1.2.2 ENHANCED INTEROPERABILITY COMPONENT

Under the scope of this Deliverable, the 1st Software Prototype of Enhanced Interoperability component consists of several sub-tasks, which are further separated into different layers of the overall proposed pipeline. The initial design of these layers, that integrate Semantic Web technologies with Natural Language Processing (NLP) technologies and tasks, includes complete workflows and pipelines for annotating cleaned data, which are being sent into the Enhanced Interoperability Component from the Data Cleaning Component. Moreover, as, also, introduced in the D4.1 Reusable Model & Analytical Tools Design and Open Specification 1 [1] through these coupled technologies and methods, the Interoperability Component seeks to provide a state-of-the-art approach to achieve interoperability in data-driven policy making domain. Semantic AI entitles this proposed hybrid approach and is a combination of commonly used semantic techniques coupled with the utilization of NLP tasks and methods. The latter is being implemented and utilized from the early beginning steps of the Interoperability Component. To this end, the main functionality of the first sub-component of the Enhanced Interoperability component, that is introduced under the scope of first Software Prototype, is to annotate cleaned data with URIs pointing into corresponding Wikidata and DBPedia [4] entities & proper Ontologies, by utilizing Semantic AI techniques. On top of this the final extraction of this sub-component are URI annotated data in JSON format. To this end, the system will be able to automatically integrate data from different sources by replacing the context-depended keys in the JSON output with URIs pointing to semantic vocabularies, that will be used to represent and link the data [5].

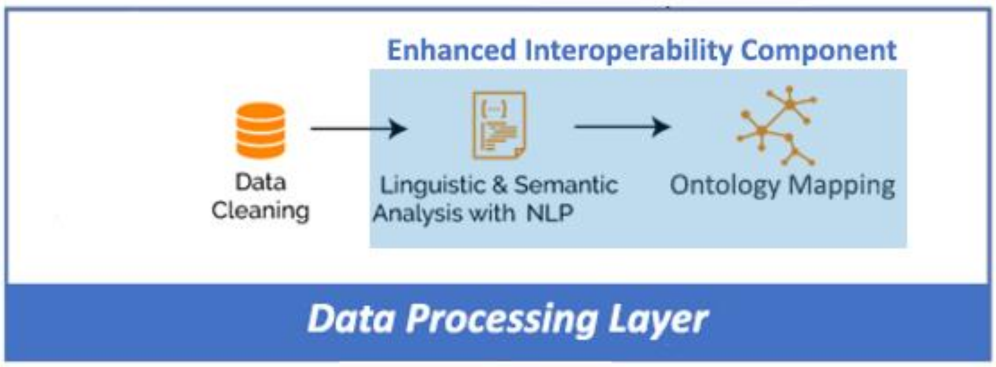

Enhanced Interoperability Component
Data Cleaning
Linguistic & Semantic Analysis with NLP
Ontology Mapping
Data Processing Layer

FIGURE 5 - ENHANCED INTEROPERABILITY SUB-COMPONENTS

Furthermore, as analysed in D4.1 and is shown in the above image (Figure 5) after fetching cleaned data from the Data Cleaning Component, the Enhanced Interoperability Component seeks to implement the hybrid approach of Semantic AI through two core subtasks/sub-components, the Linguistic & Semantic Analysis with NLP (1st sub-component) and the Ontology Mapping (2nd sub-component). To exploit what the Semantic AI offers, data first needs to be structured and annotated. To this end, in the first sub-component, which is presented and introduced in this Deliverable, cleaned data will be analyzed, transformed and annotated with appropriate URI metadata through the utilization of Semantic Web technologies and NLP tasks, such as Named Entity Recognition (NER), Part-of-Speech Tagging (POS Tagging), Tokenization etc. These coupled functionalities are being utilized into three different layers/steps of the overall sub-component as shown in the next figure (Figure 6). In next phases and Deliverables, semantic and syntactic URI annotated data will be interlinked through the task of Ontology Mapping. The overall Interoperability component will be further enhanced and completed in next steps and deliverables by the mechanism of Ontology Mapping, where a topic detection sub-component will detect the topic of the given data and afterwards an Ontology Mapping and structuring mechanism will utilized in order to interlink not only URI annotated data with proper ontologies, but also to interlink and correlate datasets among them.

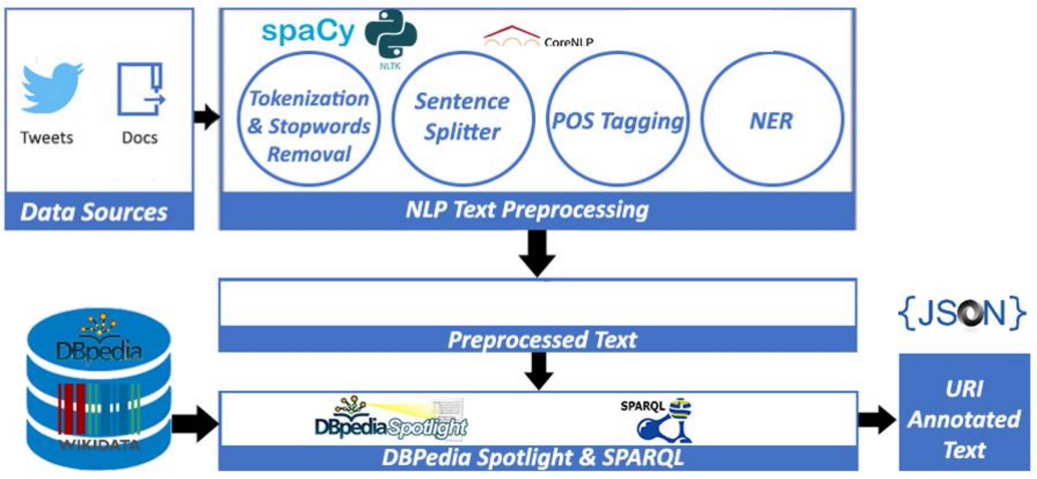


FIGURE 6 - LINGUISTIC & SEMANTIC ANALYSIS WITH NLP

On top of this, based on the above image more concrete examples and details are given below for some of the core functionalities/services that are being implemented under the scope of the first sub-component. To this end, below are listed the outcomes of some core sub-tasks of the first sub-component, which is presented in this Deliverable. Under the scope of the below demonstrations dummy content and text data from a public news page URL[1] were fetched, utilized, and processed.

- **POS Tagging:** POS tagging task is utilized after Tokenization and sentence splitting tasks. The part-of-speech tagger is responsibly to assign to each token an extended POS tag and to identify its role and usage in the text. Below follows an example of the outcome of the POS tagging task that was implemented on a raw sentence.

```
[('Mr', 'PROPN', 'Mr'),
 ('Biden', 'PROPN', 'Biden'),
 ('snatch', 'VERB', 'snatch'),
 ('Arizona', 'PROPN', 'Arizona'),
 ('reliably', 'ADV', 'reliably'),
 ('conservative', 'ADJ', 'conservative'),
('state', 'NOUN', 'state'),
 ('ahead', 'ADV', 'ahead'),
 ('Nevada', 'PROPN', 'Nevada'),
 ('counting', 'NOUN', 'counting'),
 ('expected', 'VERB', 'expect'),
 ('resume', 'VERB', 'resume'),
 ('Thursday', 'PROPN', 'Thursday')]
```

- **NER Task:** Named Entity Recognition (NER) task is the process of classifying named entities found in a text into pre-defined categories, such as persons, places, organizations, dates, etc. To this end, spaCy and its inbuild functions are being utilized. On top of this, spaCy utilizes a statistical model to classify a broad

[1] https://www.bbc.com/news/election-us-2020-54791113

range of entities, including persons, events, dates, cardinals, nationalities / religions among other entities. Below follows an example of the NER task that was implemented on the raw text.

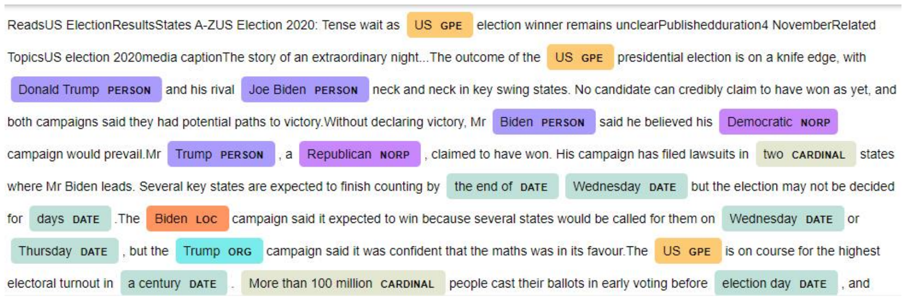

ReadsUS ElectionResultsStates A-ZUS Election 2020: Tense wait as US GPE election winner remains unclearPublishedduration4 NovemberRelated TopicsUS election 2020media captionThe story of an extraordinary night...The outcome of the US GPE presidential election is on a knife edge, with Donald Trump PERSON and his rival Joe Biden PERSON neck and neck in key swing states. No candidate can credibly claim to have won as yet, and both campaigns said they had potential paths to victory.Without declaring victory, Mr Biden PERSON said he believed his Democratic NORP campaign would prevail.Mr Trump PERSON , a Republican NORP , claimed to have won. His campaign has filed lawsuits in two CARDINAL states where Mr Biden leads. Several key states are expected to finish counting by the end of DATE Wednesday DATE but the election may not be decided for days DATE .The Biden LOC campaign said it expected to win because several states would be called for them on Wednesday DATE or Thursday DATE , but the Trump ORG campaign said it was confident that the maths was in its favour.The US GPE is on course for the highest electoral turnout in a century DATE . More than 100 million CARDINAL people cast their ballots in early voting before election day DATE , and

FIGURE 7 - NER TASK

- **JSON URI annotated sentence:** The final step of the provided sub-component is the extraction of the URI annotated text in JSON format. To this end, SPARQL query language was utilized in order to interlink identified entities from the raw text with Wikidata entities. On top of this, DBpedia Spotlight tool is utilized for interlinking and URI annotating text entities with DBPedia entities. A typical example of the code that is being utilized in order to perform the SPARQL query to the Wikidata knowledge base is presented below and the final outcome of a URI annotated sentence follows.

```
for sent in doc.sents:
    labels = list()
    for e in sent.ents:
        sparql = """SELECT ?item WHERE { ?item rdfs:label '%s'@en }""" % (e.text)
        res = return_sparql_query_results(sparql)
        labels.append({e.text, e.start_char, e.end_char, e.label_, res["results"]["bindings"][0]["item"]["value"]})
    djson.append({'text': sent.text, "labels": labels})
```

FIGURE 8 - URI ANNOTATION OF RAW TEXT

```
 [{'text': 'Mr Biden could also snatch Arizona, a once reliably conservative
 state, and is ahead in Nevada, although counting is not expected to resume
there until Thursday.', 'labels': [{'Biden', 'http://www.wikidata.org/entity
/Q421211', 3, 'PERSON', 8}, {'Arizona', 'http://www.wikidata.org/entity/Q816
', 27, 'GPE', 34 }, {'Nevada', 'http://www.wikidata.org/entity/Q1227', 'GPE'
, 88, 94}, {'Thursday', 'http://www.wikidata.org/entity/Q129', 'DATE', 152,
160}]}]
```

2.1.3 Situational Knowledge Acquisition & Analysis

For this first prototype the situational knowledge acquisition and analysis (SKA) task has been focused on providing exploratory analysis. More specifically, the goal of this first prototype is to address the Scenario A from UC1 (Use Case 1: Participatory policies against Radicalization) where the policy maker will be assisted with data aggregations of terrorisms attacks. The information will be extracted from GTD database and it will be showed to the policy maker in the form of heat maps exhibiting the incidence of attacks grouped by cities and by groups.

To fulfil the policy maker needs, the SKA-HeatMap v1 will be an analytical function in charge of connecting with the PolicyCLOUD storage component, retrieving data according a set of input parameters, and return a set of aggregated data in the form of json file. This result json file, for this prototype, will be used directly by the visualization component but in the next versions these results will be sent to the PolicyCLOUD data storage relegating to the PDT backend the task of serving results to the PDT.

This analytical function is provided as a Java application ready to be converted into an OpenWhisk function.

The data categorization functionality planned for this first prototype has been postpone due to the prioritization of having a first SKA-HeatMap prototype integrated with other PolicyCLOUD components, such as the PolicyCLOUD storage, in the way of an end-to-end scenario than having multiple standalone functionalities.

2.1.4 Opinion Mining & Sentiment Analysis

For this first software prototype, the functionality provided from this component is the sentiment analysis. It will cover the analytical part of the Scenario B from UC#2 ("Intelligent policies for the development of denomination of origin") to get the positive and negative opinions of different wine products based on social media data (more details in D6.3 [3]).

To fulfil these requirements, this component will generate positive, negative and neutral tags to each tweet, with a score to have a better understanding of those tags. A global description of the component was provided in D4.1 [1].

The generation of positive/negative/neutral tags is only the first part taken as input and output Kafka topics; this is because it is intended to run in streaming mode, as an ingest function. The second part is related to get some aggregated data, as an analytical function, based on the results of the previous part. Those aggregates, for this prototype, are based on local files (next versions will work with the PolicyCLOUD Datastore) and contains a summary (amount of positive/negative/neutral tweets and an average score) split with a concrete periodicity (second/minute/hour/and more) for a given period of time. The input and output schemas, parameters and connections between both parts are explained in Section 2.2.4.

Both functions are now provided as Apache NiFi workflows and executed with parametrized python code to be connected to the Apache NiFi instance and initialized with concrete variables. This tool is described in detail in Section 2.3.4.

2.1.5 Social Dynamics & Behavioural Data Analytics

We describe the first prototype for the Social Dynamics component of the PolicyCLOUD project. We refer to this component as Politika. Currently Politika runs as a stand-alone web application on the cloud.

Poltika takes as input a graph corresponding to a social network and simulates its dynamics using the agent-based modelling paradigm. Politika allows the user to:

- specify, edit or delete a simulation
- browse the specifications of simulations stored in the system
- execute simulations
- upload/download data to/from a simulation
- examine raw simulation results
- compute and visualize simulation analytics

Politika's output is a graph corresponding to the social network in its input after the simulation of its dynamics.

The Social Dynamics component uses a special-purpose language for specifying modelling individual, connection and policy attributes and dynamics. In particular, the user specifies modelling attributes as keyword/value strings. For example:

```
incubation_period: 5, infection_period: 10, immunity_period: 10, cur_icu_units: 0
```

represent the policy parameters for a containment policy in an epidemiological context. Strings ending in `:' are keyword names for parameters, while the rest are the parameter values.

Individual, connection and policy dynamics as specified as IF..DO rules. For example:

```
IF this.cur_icu_units > 0 and
this.cur_infection_death_prob >
this.baseline_infection_death_prob

DO

this.cur_infection_death_prob:

max(this.cur_infection_death_prob –
```

represents a policy dynamic rule that decreases the current infection death probability if it is above the baseline probability provided there are icu units available.

In particular. the IF part describes a sequence of logical conditions (and, or, not) that need to be satisfied for the rule to fire, while the DO part describes new assignments for the attributes of the entity (individual, connection or policy) executing the rule.. Multiple statements in the DO part are separated with the `;' marker, while different rules are separated with the `~' marker. In the case of individual attributes, the keyword this refers to the current individual executing the rule. The current value of such an attribute is accessed using the period (.) followed by the attribute name. For example, this.name refers to the name attribute of the individual executing this rule. The assignment operator is `:', therefore the statement:

this.name: "Nick"

assigns a new value for the attribute `name' of the current individual.

In the case of connections, the keyword edge refers to the connection executing the current rule, the keyword this refers to the individual from which the connection originates while the keyword distal refers to the individual that is the target of the connection.

The special-purpose language has a set of built-in functions that can help the modeler in developing her social simulation. The specifications for these functions along with the rest of the Politika API are described in the reference manual for Politika at: http://epinoetic.org/Politika/doc/api-reference.html

2.1.6 Operational Data Repository

An integral part of the central data repository of PolicyCLOUD is the operational datastore provided by LXS. It is a relational database that provides support both for SQL and NoSQL executions, ensuring transactional semantics and ACID properties under very high rated data ingestion workload, due to its scalable transactional processing mechanism and the provision of an additional interface that can store data directly to its storage, bypassing the introduced footprint added by the relational query engine of the data store. As a result, it can provide support both for static data ingestion via a batch process and for ingestion of streaming data. Its functionality and features have been documented in the D4.1 deliverable under section 5.3. In the scope of WP4, it has been integrated on one hand with the mechanisms developed in the scope of the DAA API Gateway, as the final chain in the data ingestion pipeline deployed by the latter will be the persistent storage of the data items into the operational datastore. Additionally, it has been integrated with the rest of the components developed under the scope of WP4, as all analytical tools need to consume data from the persistent medium which is, at first phase, the operational datastore. Finally, it is part of the component which will be the output of task T4.6 ("Optimization & Reusability of Analytical Tools"), however the latter starts on M13 and therefore it will be reported in the second version of this deliverable.

2.2 Interfaces

This section provides the description and details of the component interfaces.

2.2.1 DAA API Gateway

Following are the DAA API Gateway API. The API can be invoked with a command line or Rest interface.

2.2.1.1 Analytic function registration

Register the analytic function as action in OpenWhisk.

Parameters:

- name: function name
- filename: jar filename containing the function code
- main: specify the function main class
- function_parameters: function parameters (Json)
- type: analytic / analytic-ingest
- category: Data-Cleaning / Data-Interoperability / Situational-Knowledge / Opinion-Mining / Social-Dynamic
- description: function description and usage
- input data schema/format: TBD
- output data schema/format: TBD
- purpose: for data access enforcement, TBD

Return: success / failure

Spec: Register the function with all associated meta data (provided in the parameters) in OpenWhisk.

Return indicates whether analytic function successfully created as an action in OpenWhisk

Implementation details:

Function name should be unique.

Filename parameter specifies the jar filename containing the function code written under OpenWhisk conventions[2] placed in functions directory under the PolicyCloud project gitlab repository.

In SW1 java packaged as jar is supported for functions. All dependencies should be packaged in the jar, and the jar total size should be under 48MB.

Use cases implementing functions in DAA should follow the guidelines to generate the jar:

- java version 8 is supported
- maven has to be installed
- If the analytic function has external dependencies, they should be specified in pom.xml

[2] https://github.com/apache/openwhisk/blob/master/docs/actions-java.md

- Sample functions jar called ReferenceFunctions.jar can be found in the project gitlab as a reference implementation for jar containing external dependencies
- Run maven clean and package to create the jar file (mvn package -Dmaven.test.skip=true)
- Upload the result jar from target folder to the policyCloud github repository packaged-functions folder

Purpose and access policies are out of scope for SW1

Rest interface: PUT /analytic-function

```
curl -k -X PUT https://$URL:31001/api/v1/web/policycloud/daa/analytic-function -H 'Content-Type: application/json' -d '{"name": "copyAnalytic", "filename": "copyAnalytic.jar", "main": "org.apache.OpenWhisk.analyticFunctions.copyAnalytic", "category": "buildin", "description": "copy dataset into database", "type": " analytic-ingest", “function_parameters”: [{“parameter”:{“name”:”dataset_url”, “type”: “string”}}, {“parameter”: {“name”:”dataset_schema”, “type”:”json”}}]}'
```

2.2.1.2 Data source registration

Import a data source into the database by applying ingest functions on it.

Parameters:

- Name: data source name
- Type: Ingest-now / Streaming / External
- Description: data source description
- url: data source location
- Access: whatever is needed to access and read the data
- Schema: for the read operation, might be used to read e.g. just subset of columns
- Ingest-functions: ordered list of analytic-ingest functions (each with its parameters) [just 1 in SW 1]
- Permissions: for data access enforcement (for what purpose, which users), TBD

Return: success / fail

Spec:

1. Data source name should be unique.
2. Create the functions chain (in this prototype we demonstrate only a single function) involving Kafka topics and OpenWhisk triggers and rules, from the Kafka input topic (written into by the Cloud Gateway) to the resulted table in the DB. Create DB table.
   In SW1: Creates input (T1) and output (R1) topics in Kafka, creates an action sequence with the ingest function and kafka produce
3. Invokes Cloud Gateway task which retrieves the data source content and place it in T1 topic
4. Register the data source with its metadata

Implementation details:

1. Supported types in SW1: Ingest-now and streaming
2. T1 and R1 topic names are derived from the data source name.
3. The Cloud Gateway task is supplied with the url, Access parameters, Schema, T1 topic, and Type. It is responsible on reading the data accordingly and placing it in T1 topic
   - On Ingest-now – a simple chunks loop to place the content as messages in T1 topic
   - On Streaming – a periodic thread to place incoming streams as messages in T1 topic

OpenWhisk triggers and rules created subscribe to T1 topic. Once a message was placed in the topic, the ingest-function given is invoked and the resulted new dataset is placed in R1, which in turn is written to the database.

4. The name of the data source in the DAA and the name of the data source backed table is the data source given name (Name parameter).
5. Cloud Gateway component is called to pull the dataset from the given url (mocked in SW1 by placing messages on T1).

Rest interface: PUT /datasource

curl -k -X PUT https://$URL/api/v1/web/policycloud/daa/datasource -H 'Content-Type: application/json' -d '{"ingest-functions": "CopyAnalytic", "name": "gtd", "description": "demo-datasource", "url": "http://path/to/url/GTD_sample.csv", "type": "ingest-now", "schema": {}, "permissions": ""}"}'

2.2.1.3 List Registered Functions

Display a detailed list of registered functions
Parameters:

- Type: analytic / analytic-ingest

Return: List of all registered functions (analytic or analytic-ingest), each with its metadata

Rest interface: GET /analytic-function?type=

Get detailed list of all the resisted functions

curl -k **-**X GET https://$URL/api/v1/web/policycloud/daa/analytic-function

result example:

```
[{
  "name": "copyAnalytic",
 "category": "buildin",
   "description": "copy dataset into database”,
   "function_parameters": "",
…
}]
```

2.2.1.4 List Registered Data sources

Parameters: None

Return: List of all registered data sources, each with its metadata (including the actual DBTable)

Rest interface: GET /datasource

Get detailed list of all the resisted functions

curl -k -X GET https://$URL:31001/api/v1/web/policycloud/daa/datasource

result example:

```
]{
  "name": " gtd ",
    "description": " demo datasource”,
  "url": "http://path/to/url/GTD_sample_fixed.csv",
  "schema":{},
    …
}]
```

2.2.1.5 Apply Function

Invoke the requested analytics on the requested data source

Parameters:

- UserCredentials: (TBD where is the enforcement point)
- Function: Analytic function name
- Parameters: for the function (Json)
- Data-source: data source name

Return: jobID/json result

Spec:

1. Invoke the OpenWhisk function (with the Parameters and data source (db table))
2. Return the OpenWhisk jobID if asynchronous. The function result in json for synchronous

Implementation details:

Job id is a wrapper for OpenWhisk activation id

Rest interface: POST / analytic-function?result=true

```
curl -k -X POST https://$URL:31001/api/v1/web/policycloud/daa/analytic-function -H 'Content-Type: application/json' -d '{"function": "DisplayFunction", "datasource":"gtd", "parameters": " [] "}'
```

2.2.1.6 Get Job Status

Return the job result

Parameters:

- jobID: id of the job

Return: Working / Completed / Failed, function json result

Spec:

1. Used for invoking asynchronous functions on data source
2. Return the job status, if completed return the function json result in the body

Implementation details:

In SW1 when job is completed – returns the result of the invocation of the function on the data source (json) in the body. error - if job id is not found or job is not completed.

Job id is a wrapper for OpenWhisk activation id

Rest interface: GET /analytic-function?job_id=

```
curl -k -X GET https://$URL:31001/api/v1/web/policycloud/daa/analytic-function?job_id=123
```

2.2.2 Enhanced Interoperability & Data Cleaning

2.2.2.1 Data Cleaning Component

The incoming HTTP requests are handled by the DataCleanerController while the execution of the workflow is performed by the DataCleaningService. Through the provided interface the following endpoint is offered: Data Cleaning endpoint. This endpoint is responsible for handling the data cleaning workflow execution requests and providing the updated data as a result of the execution. To access the Data Cleaning endpoint, the latter expects the dataset for which the data cleaning workflow will be executed in csv format, accompanied by its corresponding data schema in pandas schema format. The Data Cleaning endpoint is documented below:

Spec:

Initiates the data cleaning workflow and provides the estimated results.

Endpoint URL:

http://hostname[:port]/cleaning

HTTP method:

POST

Parameters:

No parameters needed. All the needed input for starting the execution of the Data Cleaning component is uploaded and given to the component from the end-user through the provided user interface.

2.2.2.2 Enhanced Interoperability Component

Under the scope of this Deliverable the Enhanced Interoperability Component is provided through a IPython Notebook (.ipynb file extension) in order any stakeholder or user to be able to execute the code and identify the outcomes in every step and sub-task of this specific sub-component. Hence, no endpoint or API call is needed to trigger the execute of this specific component. The only parameter that can be changed manually in the code is the given URL from which the raw text will be fetched. Moreover, a docker installation with a corresponding Flask application coupled with the utilization of Swagger UI has already started to being implemented in order to provide a REST application interface following the OpenAPI specification. To this end, it will be easier for the end user to discover the capabilities of the component and to provide well-structured documentation for each of the component's services.

2.2.3 Situational Knowledge Acquisition & Analysis

2.2.3.1 Parameter Model

The analytical tools will be called with a set of input parameters. In order to use the same approach for the description of these input parameters a parameter format has been agreed. This Json format describes (the input) parameters used by the SKA analytical tool. These are the main fields include in the format:

- "Name", the name of the parameter
- "ValueType", the accepted type for the parameter values. There are different possibilities: BOOLEAN, INTEGER, FLOAT, STRING
- "EvaluationTypesAllowed", can be one of the following: EQUAL, LESS, LESS_OR_EQUAL, GREATER, GREATER_OR_EQUAL, IS, NOT, RANGE, IN

An example of a parameter described with this format can be seen in Figure 9:

```
{
  "id": "17c5410f-8788-4d9a-af48-ebcb0e41c4f6",
  "name": "parameter name",
  "unit": "parameter unit",
  "description": "this is the description",
  "valueType": "STRING",
  "valueContraints": [
    {
      "contraintType": "DEFAULT",
      "value": "default value"
    }
  ],
  "evaluationTypesAllowed": [
    "EQUAL",
    "GREATER",
    "GREATER_OR_EQUAL",
    "LESS",
    "LESS_OR_EQUAL"
  ]
}
```

FIGURE 9 - PARAMETER FORMAT

In Figure 10 can be seen an example of a parameter with a concrete value.

```
{
  "name": "iyear",
  "unit": "years",
  "description": "the year belong to a date",
  "valueType": "STRING",
  "valueContraints": [
    {
      "contraintType": "DEFAULT",
      "value": "default value"
    }
  ],
  "evaluationTypesAllowed": [
    "EQUAL",
    "GREATER",
    "GREATER_OR_EQUAL",
    "LESS",
    "LESS_OR_EQUAL"
  ],
  "inputType": "VARIABLE",
  "required": true,
  "value": {
  "type": "EQUAL",
  "value": "2004"
    }
}
```

FIGURE 10 - PARAMETERS WITH VALUES

2.2.3.2 SKA 1ST PROTOTYPE INPUT PARAMETERS

For the case of the SKA-HeatMap, any column belonging to a database imported in the PolicyCLOUD storage system can be subject to be included in the parameters file. The Scenario A of UC1, which works with the GTD database, the two following fields are used to build the heat map:

- "iyear", the year when happen the attacks. It can have a unique value (type:EQUAL): 1975, or a set of values (type:IN): [1975, 1976, 1977]
- "region", the region where happen the attacks. It can have a unique value (type:EQUAL): 1975, or a set of values (type:IN): [1975, 1976, 1977]

Therefore, a parameter.json file, for the SKA-HeatMap 1st prototype, looks like the following example in Figure 11:

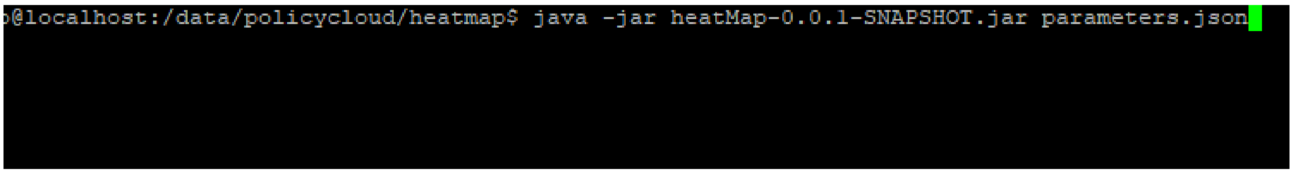
```
{{
  "name": "iyear",
  "unit": "years",
  "description": "the year belong to a date",
  "valueType": "STRING",
  "valueContraints": [
    {
      "contraintType": "DEFAULT",
      "value": "default value"
    }
  ],
  "evaluationTypesAllowed": [
    "EQUAL",
    "GREATER",
    "GREATER_OR_EQUAL",
    "LESS",
    "LESS_OR_EQUAL"
  ],
  "inputType": "VARIABLE",
  "required": true,
  "value": {
  "type": "EQUAL",
  "value": "2004"
    }
},
{
  "name": "region",
  "description": "the region when happened the attack",
  "valueType": "STRING",
  "valueContraints": [
    {
      "contraintType": "DEFAULT",
      "value": "default value"
    }
  ],
  "evaluationTypesAllowed": [
    "EQUAL",
    "GREATER",
    "GREATER_OR_EQUAL",
    "LESS",
    "LESS_OR_EQUAL"
  ],
  "inputType": "VARIABLE",
  "required": true,
  "value": {
  "type": "IN",
  "value": [
            "8",
            "9"
    ]
    }
}
}
```

FIGURE 11 - SKA-HEATMAP INPUT PARAMETERS JSON FILE EXAMPLE

The previous input parameters will give as a result a heat map showing the attacks incidence along the year 2004 in the regions 8 (Western Europe) and 9 (Eastern Europe).

2.2.3.3 SKA-HeatMap 1st prototype Invocation

For this 1st prototype a standalone version of the SKA-HeatMap has been provided. It is a java process and therefore in order to execute it, it will be enough with call the following command:



```
p@localhost:/data/policycloud/heatmap$ java -jar heatMap-0.0.1-SNAPSHOT.jar parameters.json
```

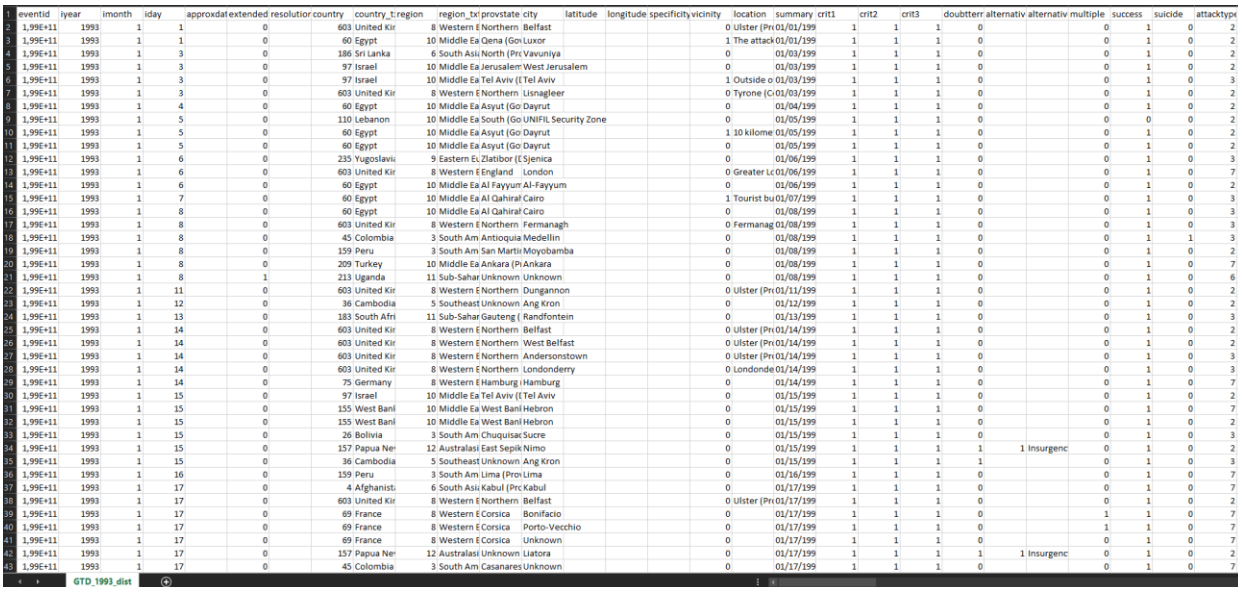
FIGURE 12 - SKA-HEATMAP INVOCATION

The data schema used by the SKA-HeatMap 1st prototype is that from GTD data base. It includes a set of entities, User, Event, Target, but just the Event entity has been used for the heat map:

Attribute*	Mandatory (YES/NO)	Type	Description
eventID	YES	Integer	The unique identifier of the event
iyear	YES	Integer	The year the event occurred
imonth	YES	integer	The month the event occurred
iday	YES	Integer	The date the event occurred
country	YES	Integer	The country code the event occurred
country_txt	YES	String	The country the event occurred
region	YES	Integer	The region code the event occurred
region_txt	YES	String	The region the event occurred
provstate	YES	String	The province the event occurred
city	YES	String	The city the event occurred
location	YES	String	The exact location the event occurred
summary	YES	String	Description of the event occurred
multiple	YES	Integer	Specifies if there were multiple events that took place
success	YES	Integer	Specifies if the event was successful or not
attacktype	YES	Integer	The attack type code that occurred
attacktype_txt	YES	String	Description of the attack occurred
weaptype	YES	Integer	The weapon type code used
weaptype_txt	YES	String	The weapon type used
weapsubtype	YES	Integer	The weapon sub-type code used
weapsubtype_txt	YES	String	The weapon sub-type used
weapdetail	YES	String	Description of the weapon used
nkill	YES	integer	Number of killed persons

TABLE 1 - EVENT ENTITY - GTD DATABASE

An example of the data can be seen in Figure 13.



1	eventid	iyear	imonth	iday	approxdat	extended	resolutior	country	country_t	region	region_tx	provstate	city	latitude	longitude	specificity	vicinity	location	summary	crit1	crit2	crit3	doubtterr	alternativ	alternativ	multiple	success	suicide	attacktype
2	1,99E+11	1993	1	1		0		603	United Kir	8	Western E	Northern	Belfast				0	Ulster (Pr	01/01/199	1	1	1	0			0	1	0	2
3	1,99E+11	1993	1	1		0		60	Egypt	10	Middle Ea	Qena (Go	Luxor				1	The attack	01/01/199	1	1	1	0			0	1	0	2
4	1,99E+11	1993	1	3		0		186	Sri Lanka	6	South Asi	North (Pr	Vavuniya				0		01/03/199	1	1	1	0			0	1	0	2
5	1,99E+11	1993	1	3		0		97	Israel	10	Middle Ea	Jerusalem	West Jerusalem				0		01/03/199	1	1	1	0			0	1	0	2
6	1,99E+11	1993	1	3		0		97	Israel	10	Middle Ea	Tel Aviv (I	Tel Aviv				1	Outside o	01/03/199	1	1	1	0			0	1	0	3
7	1,99E+11	1993	1	3		0		603	United Kir	8	Western E	Northern	Lisnagleer				0	Tyrone (C	01/03/199	1	1	1	0			0	1	0	2
8	1,99E+11	1993	1	4		0		60	Egypt	10	Middle Ea	Asyut (Go	Dayrut				0		01/04/199	1	1	1	0			0	1	0	2
9	1,99E+11	1993	1	5		0		110	Lebanon	10	Middle Ea	South (Go	UNIFIL Security Zone				0		01/05/199	1	1	1	0			0	0	0	2
10	1,99E+11	1993	1	5		0		60	Egypt	10	Middle Ea	Asyut (Go	Dayrut				1	10 kilome	01/05/199	1	1	1	0			0	1	0	2
11	1,99E+11	1993	1	5		0		60	Egypt	10	Middle Ea	Asyut (Go	Dayrut				0		01/05/199	1	1	1	0			0	1	0	2
12	1,99E+11	1993	1	6		0		235	Yugoslavi	9	Eastern E	Zlatibor (I	Sjenica				0		01/06/199	1	1	1	0			0	1	0	3
13	1,99E+11	1993	1	6		0		603	United Kir	8	Western E	England	London				0	Greater Lc	01/06/199	1	1	1	0			0	1	0	7
14	1,99E+11	1993	1	6		0		60	Egypt	10	Middle Ea	Al Fayyum	Al-Fayyum				0		01/06/199	1	1	1	0			0	1	0	2
15	1,99E+11	1993	1	7		0		60	Egypt	10	Middle Ea	Al Qahira	Cairo				1	Tourist bu	01/07/199	1	1	1	0			0	1	0	3
16	1,99E+11	1993	1	8		0		60	Egypt	10	Middle Ea	Al Qahira	Cairo				0		01/08/199	1	1	1	0			0	1	0	3
17	1,99E+11	1993	1	8		0		603	United Kir	8	Western E	Northern	Fermanagh				0	Fermanag	01/08/199	1	1	1	0			0	1	0	3
18	1,99E+11	1993	1	8		0		45	Colombia	3	South Am	Antioquia	Medellin				0		01/08/199	1	1	1	0			0	1	1	3
19	1,99E+11	1993	1	8		0		159	Peru	3	South Am	San Marti	Moyobamba				0		01/08/199	1	1	1	0			0	1	0	2
20	1,99E+11	1993	1	8		0		209	Turkey	10	Middle Ea	Ankara (P	Ankara				0		01/08/199	1	1	1	0			0	1	0	7
21	1,99E+11	1993	1	8		1		213	Uganda	11	Sub-Sahar	Unknown	Unknown				0		01/08/199	1	1	1	0			0	1	0	6
22	1,99E+11	1993	1	11		0		603	United Kir	8	Western E	Northern	Dungannon				0	Ulster (Pr	01/11/199	1	1	1	0			0	1	0	2
23	1,99E+11	1993	1	12		0		36	Cambodia	5	Southeast	Unknown	Ang Kron				0		01/12/199	1	1	1	0			0	1	0	2
24	1,99E+11	1993	1	13		0		183	South Afri	11	Sub-Sahar	Gauteng (	Randfontein				0		01/13/199	1	1	1	0			0	1	0	3
25	1,99E+11	1993	1	14		0		603	United Kir	8	Western E	Northern	Belfast				0	Ulster (Pr	01/14/199	1	1	1	0			0	1	0	2
26	1,99E+11	1993	1	14		0		603	United Kir	8	Western E	Northern	West Belfast				0	Ulster (Pr	01/14/199	1	1	1	0			0	1	0	2
27	1,99E+11	1993	1	14		0		603	United Kir	8	Western E	Northern	Andersonstown				0	Ulster (Pr	01/14/199	1	1	1	0			0	1	0	3
28	1,99E+11	1993	1	14		0		603	United Kir	8	Western E	Northern	Londonderry				0	Londonde	01/14/199	1	1	1	0			0	1	0	3
29	1,99E+11	1993	1	14		0		75	Germany	8	Western E	Hamburg	Hamburg				0		01/14/199	1	1	1	0			0	1	0	7
30	1,99E+11	1993	1	15		0		97	Israel	10	Middle Ea	Tel Aviv (I	Tel Aviv				0		01/15/199	1	1	1	0			0	1	0	2
31	1,99E+11	1993	1	15		0		155	West Banl	10	Middle Ea	West Banl	Hebron				0		01/15/199	1	1	1	0			0	1	0	7
32	1,99E+11	1993	1	15		0		155	West Banl	10	Middle Ea	West Banl	Hebron				0		01/15/199	1	1	1	0			0	1	0	2
33	1,99E+11	1993	1	15		0		26	Bolivia	3	South Am	Chuquisac	Sucre				0		01/15/199	1	1	1	0			0	1	0	3
34	1,99E+11	1993	1	15		0		157	Papua Ne	12	Australasi	East Sepik	Nimo				0		01/15/199	1	1	1	1	1	Insurgenc	0	1	0	2
35	1,99E+11	1993	1	15		0		36	Cambodia	5	Southeast	Unknown	Ang Kron				0		01/15/199	1	1	1	1			0	1	0	3
36	1,99E+11	1993	1	16		0		159	Peru	3	South Am	Lima (Pro	Lima				0		01/16/199	1	1	1	0			0	1	0	7
37	1,99E+11	1993	1	17		0		4	Afghanist	6	South Asi	Kabul (Pr	Kabul				0		01/17/199	1	1	1	0			0	1	0	7
38	1,99E+11	1993	1	17		0		603	United Kir	8	Western E	Northern	Belfast				0	Ulster (Pr	01/17/199	1	1	1	0			0	1	0	2
39	1,99E+11	1993	1	17		0		69	France	8	Western E	Corsica	Bonifacio				0		01/17/199	1	1	1	0			1	1	0	7
40	1,99E+11	1993	1	17		0		69	France	8	Western E	Corsica	Porto-Vecchio				0		01/17/199	1	1	1	0			1	1	0	7
41	1,99E+11	1993	1	17		0		69	France	8	Western E	Corsica	Unknown				0		01/17/199	1	1	1	0			0	1	0	7
42	1,99E+11	1993	1	17		0		157	Papua Ne	12	Australasi	Unknown	Liatora				0		01/17/199	1	1	1	1	1	Insurgenc	0	1	0	2
43	1,99E+11	1993	1	17		0		45	Colombia	3	South Am	Casanares	Unknown				0		01/17/199	1	1	1	0			0	1	0	7

GTD_1993_dist

FIGURE 13 - GTD SAMPLE

2.2.3.4 SKA-HeatMap 1st prototype output

The SKA-HeatMap analysis gives a result a json file containing the events subject to be depicted in the heat map. It is a list of *regions incidence* objects where each of them has the following format:

- Num_events, number of events found in the region
- Location: a GeoJson object describe the area of research
- Location 2: is GTD-dependant information, such as the region code and the region text
- Start date: the init date of the period to be analysed
- End date: the end date of the period to be analysed. If the end date is null the analysis is performer for a concrete moment denoted by start date.
- Events: a list containing the event found in the analysis. For each event following information is attached:
    - eventid, the event id,
    - iyear, the year when the event take place
    - latitude, the latitude coordinate where the event takes place
    - longitude, the longitude coordinate where the event take place
    - summary, the description of the event

As it can be seen, the result format is enough general to host the result of different types of aggregation related with events. It will be a matter of including or replacing some event's attributes (which are adhered to the GTD schema (iyear, eventid, …) to make this SKA-HeatMap analysis reusable for another different data sources.

An example of the result is shown in Figure 14.

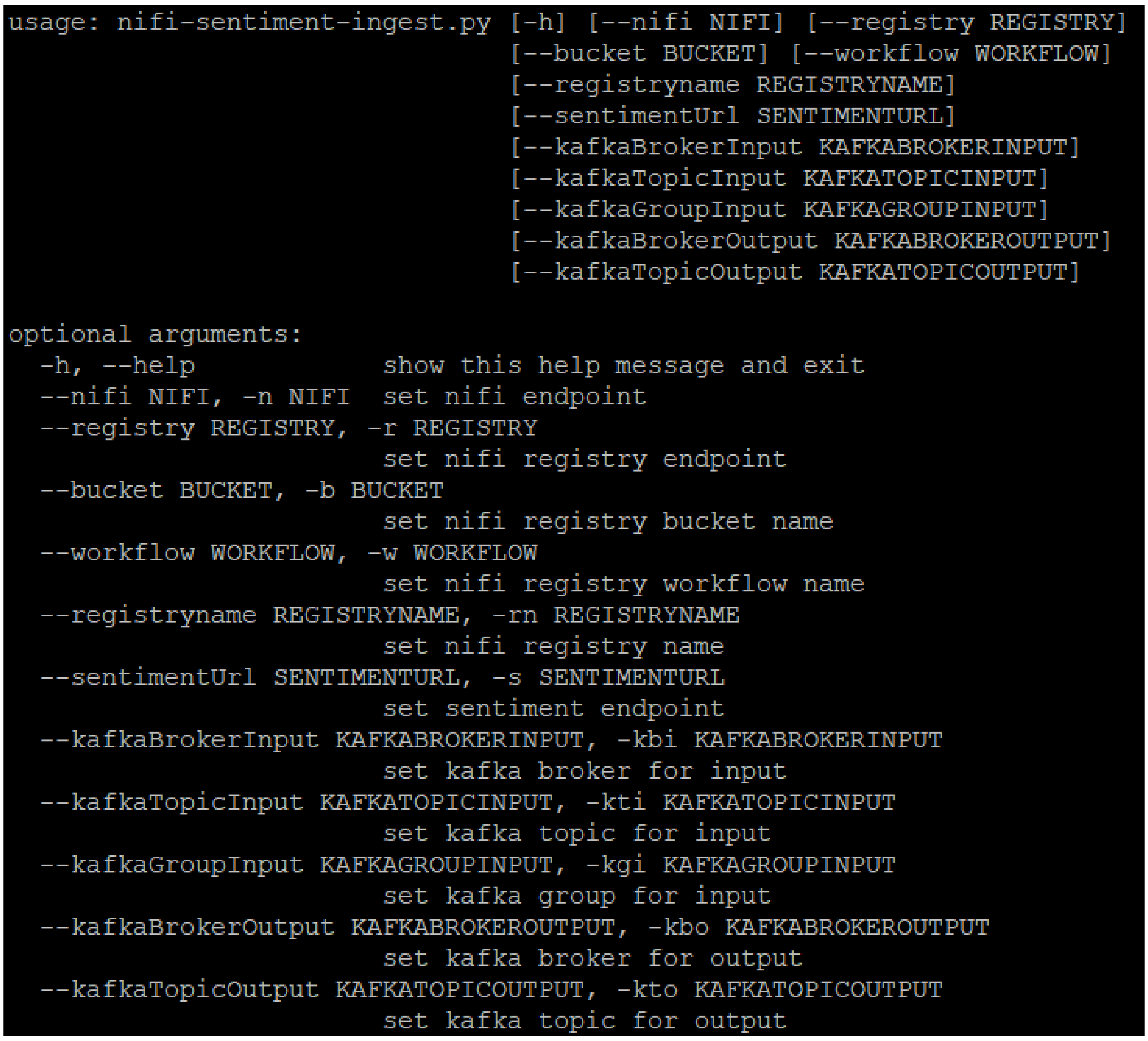
```
[
{"num_events":4,
"location":{"type":"Point","coordinates":[48.1667,100.1667]},
"location2":{"region":1,"region_txt":"North America"},
"startDate":310642173465,
"endDate":973416573465,
"events":[
      {"eventid":341684392,
       "iyear":"1979",
       "latitude":40.697132,
       "longitude":-73.931351,
       "esummary":"10/18/1979: In a series of 2 linked attack…
      }, …
}, …]
```

FIGURE 14 - SKA-HEATMP RESULTS JSON FILE EXAMPLE

Future versions of the SKA will store the results in the PolicyCLOUD storage from where the PDT backend will retrieve the results and put them available to the PDT.

2.2.4 Opinion Mining & Sentiment Analysis

For the ingest function to generate the sentiment of the tweets, there are some parameters that need to be filled: to connect to an instance of Apache NiFi, to communicate with Kafka topics, and to specify the microservice endpoint to apply the sentiment analysis. Figure **15** is the output of the help mode of the function.



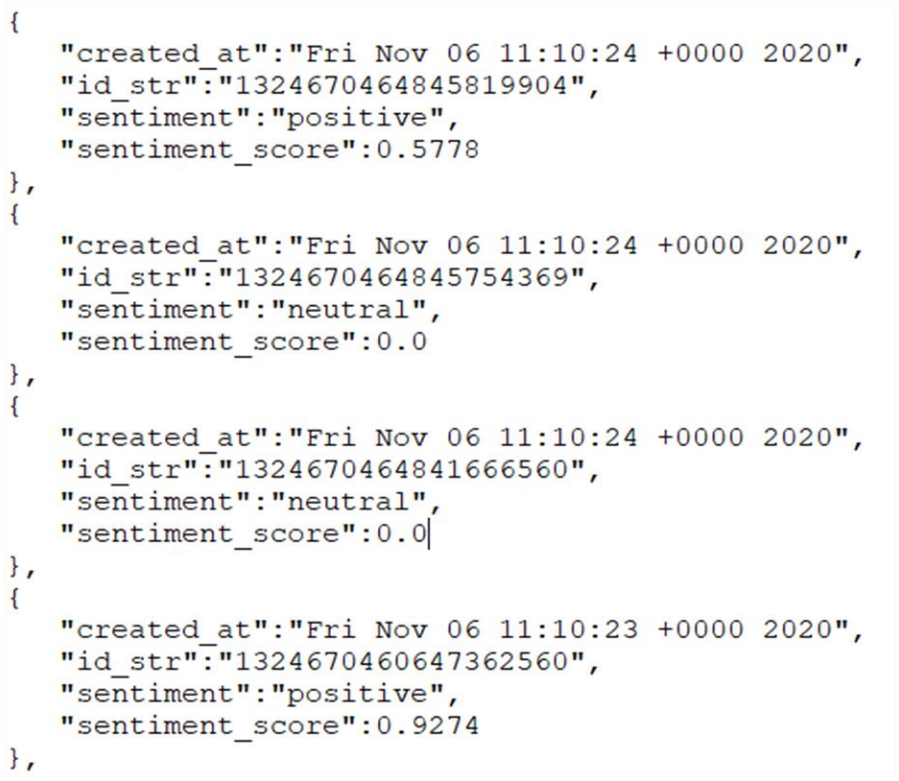
```
usage: nifi-sentiment-ingest.py [-h] [--nifi NIFI] [--registry REGISTRY]
                                [--bucket BUCKET] [--workflow WORKFLOW]
                                [--registryname REGISTRYNAME]
                                [--sentimentUrl SENTIMENTURL]
                                [--kafkaBrokerInput KAFKABROKERINPUT]
                                [--kafkaTopicInput KAFKATOPICINPUT]
                                [--kafkaGroupInput KAFKAGROUPINPUT]
                                [--kafkaBrokerOutput KAFKABROKEROUTPUT]
                                [--kafkaTopicOutput KAFKATOPICOUTPUT]

optional arguments:
  -h, --help            show this help message and exit
  --nifi NIFI, -n NIFI  set nifi endpoint
  --registry REGISTRY, -r REGISTRY
                        set nifi registry endpoint
  --bucket BUCKET, -b BUCKET
                        set nifi registry bucket name
  --workflow WORKFLOW, -w WORKFLOW
                        set nifi registry workflow name
  --registryname REGISTRYNAME, -rn REGISTRYNAME
                        set nifi registry name
  --sentimentUrl SENTIMENTURL, -s SENTIMENTURL
                        set sentiment endpoint
  --kafkaBrokerInput KAFKABROKERINPUT, -kbi KAFKABROKERINPUT
                        set kafka broker for input
  --kafkaTopicInput KAFKATOPICINPUT, -kti KAFKATOPICINPUT
                        set kafka topic for input
  --kafkaGroupInput KAFKAGROUPINPUT, -kgi KAFKAGROUPINPUT
                        set kafka group for input
  --kafkaBrokerOutput KAFKABROKEROUTPUT, -kbo KAFKABROKEROUTPUT
                        set kafka broker for output
  --kafkaTopicOutput KAFKATOPICOUTPUT, -kto KAFKATOPICOUTPUT
                        set kafka topic for output
```

FIGURE 15 **- HELP MODE OF THE SENTIMENT INGEST FUNCTION**

The data schema is represented by the Twitter API[3], in summary it is a JSON object with some fields (for us it is only important the “tweet_id”, the “created_at” field and the “text”) to be analysed for generating a sentiment score. The data that will be sent to another Kafka topic is another JSON object that will contain the id and created_at fields with the sentiment information, as shown in Figure 16. For this prototype, a simple Apache NiFi workflow has been created to fetch data from Twitter, filtered by some keywords related to wine products like “somontano, campo borja, cariñena, calatayud”, using the “GetTwitter” processor and sending the tweets to a Kafka topic. In next phases, this data will be retrieved by the cloud gateways from WP3.

[3] https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/tweet-object



```
{
    "created_at":"Fri Nov 06 11:10:24 +0000 2020",
    "id_str":"1324670464845819904",
    "sentiment":"positive",
    "sentiment_score":0.5778
},
{
    "created_at":"Fri Nov 06 11:10:24 +0000 2020",
    "id_str":"1324670464845754369",
    "sentiment":"neutral",
    "sentiment_score":0.0
},
{
    "created_at":"Fri Nov 06 11:10:24 +0000 2020",
    "id_str":"1324670464841666560",
    "sentiment":"neutral",
    "sentiment_score":0.0
},
{
    "created_at":"Fri Nov 06 11:10:23 +0000 2020",
    "id_str":"1324670460647362560",
    "sentiment":"positive",
    "sentiment_score":0.9274
},
```

FIGURE 16 - OUTPUT OF SENTIMENT INGEST FUNCTION

The Apache NiFi workflow for the ingest function is presented in Figure 17:

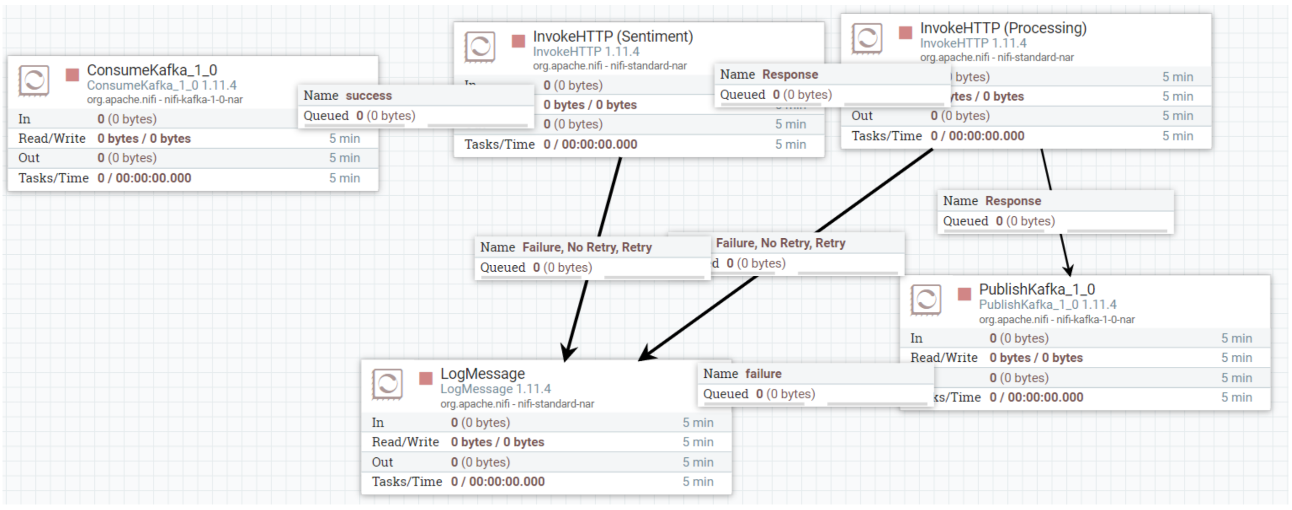
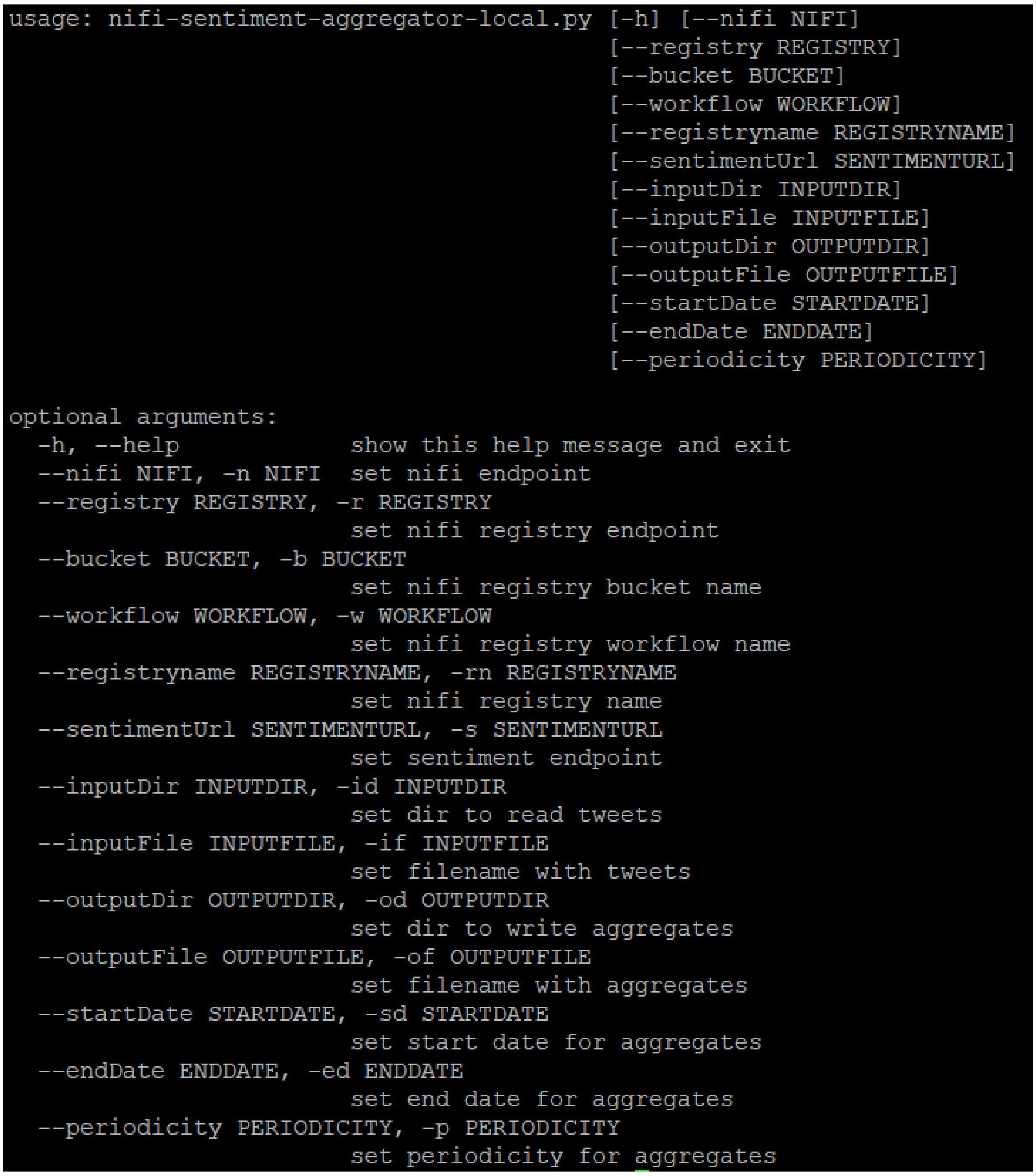


FIGURE 17 - APACHE NIFI WORKFLOW FOR SENTIMENT INGEST FUNCTION

The second function, the analytical one to generate some aggregates, will be connected at the start and at the end with the PolicyCLOUD Datastore, but for this prototype, this functionality is isolated and is working with local files. A very simple workflow has been developed in Apache NiFi to store the tweets sent to Kafka in the ingest function as a list of tweets in JSON. This list of tweets will be the same, or very similar to the data to be retrieve by the datastore.

The parameters to connect to Apache NiFi are the same as before. In this version, there are some parameters to read from and write to local files, that will be changed in next versions to the datastore endpoint. The important parameters that define how to aggregate are: start date, end date and periodicity. All of them are in Figure 18.



```
usage: nifi-sentiment-aggregator-local.py [-h] [--nifi NIFI]
                                          [--registry REGISTRY]
                                          [--bucket BUCKET]
                                          [--workflow WORKFLOW]
                                          [--registryname REGISTRYNAME]
                                          [--sentimentUrl SENTIMENTURL]
                                          [--inputDir INPUTDIR]
                                          [--inputFile INPUTFILE]
                                          [--outputDir OUTPUTDIR]
                                          [--outputFile OUTPUTFILE]
                                          [--startDate STARTDATE]
                                          [--endDate ENDDATE]
                                          [--periodicity PERIODICITY]

optional arguments:
  -h, --help            show this help message and exit
  --nifi NIFI, -n NIFI  set nifi endpoint
  --registry REGISTRY, -r REGISTRY
                        set nifi registry endpoint
  --bucket BUCKET, -b BUCKET
                        set nifi registry bucket name
  --workflow WORKFLOW, -w WORKFLOW
                        set nifi registry workflow name
  --registryname REGISTRYNAME, -rn REGISTRYNAME
                        set nifi registry name
  --sentimentUrl SENTIMENTURL, -s SENTIMENTURL
                        set sentiment endpoint
  --inputDir INPUTDIR, -id INPUTDIR
                        set dir to read tweets
  --inputFile INPUTFILE, -if INPUTFILE
                        set filename with tweets
  --outputDir OUTPUTDIR, -od OUTPUTDIR
                        set dir to write aggregates
  --outputFile OUTPUTFILE, -of OUTPUTFILE
                        set filename with aggregates
  --startDate STARTDATE, -sd STARTDATE
                        set start date for aggregates
  --endDate ENDDATE, -ed ENDDATE
                        set end date for aggregates
  --periodicity PERIODICITY, -p PERIODICITY
                        set periodicity for aggregates
```

FIGURE 18 - HELP MODE OF SENTIMENT AGGREGATION FUNCTION

The list of tweets with the sentiment information for a given period of time and a concrete periodicity will be analysed and is generated an output with the aggregated information (amount of positive/negative/neutral tweets and average score) for each split defined by the periodicity (an example is shown in Figure 19). Those aggregates are now stored in a local file, but it will be sent back to the datastore to be visualized in the PDT.

```
{
   "aggregates":[
      {
         "endDate":"2020-11-06 11:10:23",
         "initDate":"2020-11-06 11:10:22",
         "negative":5,
         "neutral":45,
         "positive":4,
         "score":0.0253462962962963
      },
      {
         "endDate":"2020-11-06 11:10:24",
         "initDate":"2020-11-06 11:10:23",
         "negative":0,
         "neutral":13,
         "positive":1,
         "score":0.07335
      }
   ],
   "endDate":"2020-11-06 11:10:24",
   "initDate":"2020-11-06 11:10:22",
   "periodicity":"second"
}
```

FIGURE 19 - OUTPUT OF SENTIMENT AGGREGATION FUNCTION

This aggregated information will be visualized as gauge chart with the average score provided, and as a time series chart to display the trend of the sentiment along the time.

The aggregate function workflow developed in Apache NiFi is represented in Figure 20:

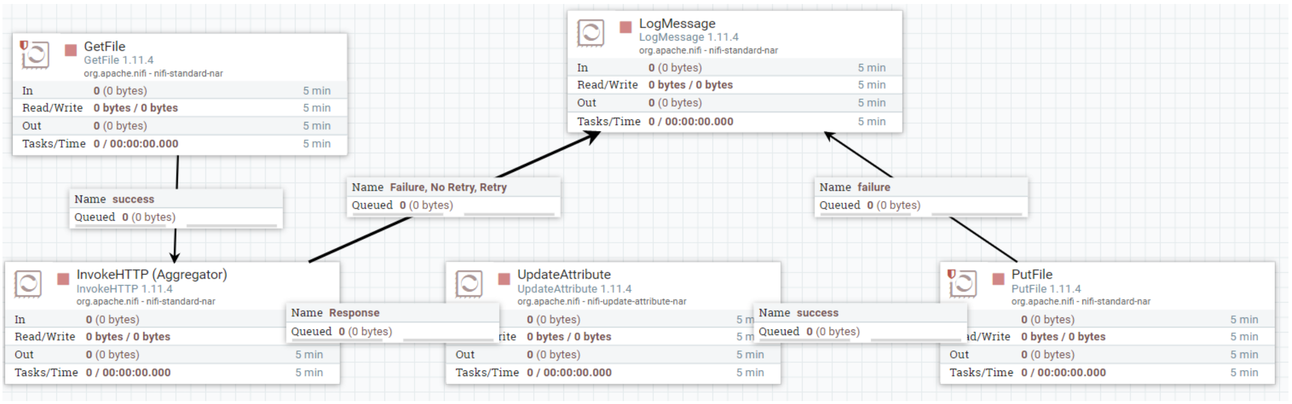

GetFile
GetFile 1.11.4
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
LogMessage
LogMessage 1.11.4
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
Name Failure, No Retry, Retry
Queued 0 (0 bytes)
Name failure
Queued 0 (0 bytes)
InvokeHTTP (Aggregator)
InvokeHTTP 1.11.4
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes)
Read/Write 0 bytes / 0 bytes
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name Response
Queued 0 (0 bytes)
UpdateAttribute
UpdateAttribute 1.11.4
org.apache.nifi - nifi-update-attribute-nar
0 (0 bytes) 5 m
rite 0 bytes / 0 bytes 5 m
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
PutFile
PutFile 1.11.4
org.apache.nifi - nifi-standard-nar
0 (0 bytes) 5 min
d/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

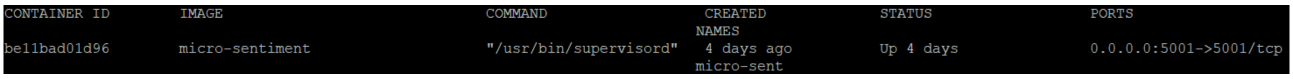
FIGURE 20 - APACHE NIFI WORKFLOW OF SENTIMENT AGGREGATION FUNCTION

Both workflows, sentiment and aggregates, are exported as Apache NiFi templates to be executed in any Apache NiFi instance for production purposes, available following the instructions of Section 3.1.4.

Finally, all the analysis described (sentiment, fields filtering, aggregation) are developed as a microservice, using Python and Flask[4], to be called in the same way that a REST API with different endpoints to each functionality (with an "InvokeHTTP" processor in Apache NiFi). This microservice has been containerized in Docker and can be deployed wherever it is needed. For this software prototype, the list of functions available are:

- /preprocess: for filtering some fields in the tweet JSON object and remove those that are not important for the sentiment function.
- /sentimentByVader: analyse the sentiment of the tweet using Vader library (details in D4.1 [1]).
- /sentimentByTextblob: analyse the sentiment of the tweet using Textblob library (details in D4.1 [1]).
- /aggregator: generate the aggregates for a concrete periodicity in a given period of time.

As it has been deployed as a microservice using Docker, this list can be easily updated and redeployed again without changing anything in the workflow.



```
CONTAINER ID        IMAGE                             COMMAND                  CREATED             STATUS                   PORTS
                                                                              NAMES
be11bad01d96        micro-sentiment                   "/usr/bin/supervisord"   4 days ago          Up 4 days                0.0.0.0:5001->5001/tcp
                                                                              micro-sent
```

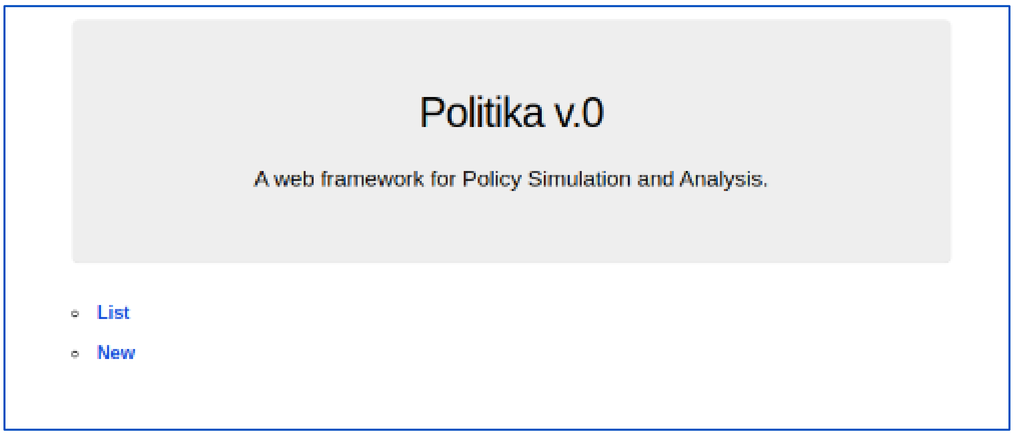
FIGURE 21 - MICROSERVICE DEPLOYED AS DOCKER CONTAINER

2.2.5 Social Dynamics & Behavioural Data Analytics

We describe the options available to the user for interacting with Politika and provide relevant screenshots.

The main screen of the system contains two options:

- List, for listing all existing simulations
- New, for creating a new simulation



Politika v.0

A web framework for Policy Simulation and Analysis.

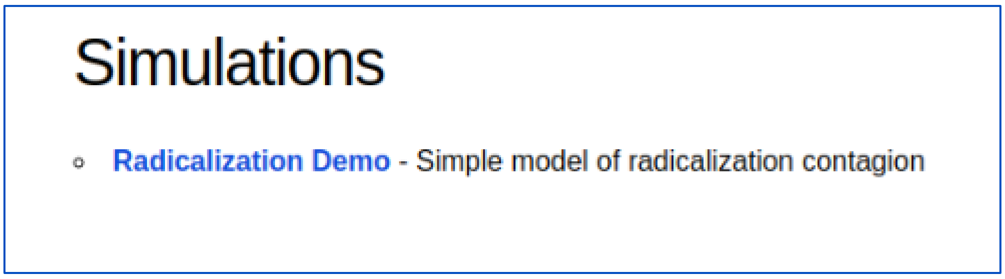
- List
- New

FIGURE 22 - INTRO SCREEN FOR POLITIKA WITH THE OPTIONS TO LIST ALL EXISTING

[4] https://flask.palletsprojects.com/en/1.1.x/

Clicking on New the user is presented with a form for creating a new simulation. The form contains a set of text boxes that the user needs to fill. After submitting the form, it is checked for syntactic correctness of the various fields. If all fields are correct a new simulation is created in the database, otherwise the system displays the same form as filled by the user and containing a list of messages describing the errors messages that were spotted in the user input.

Clicking on List the user is presented with a list of existing simulations. Currently the prototype site contains a simple simulation of radicalization contagion on a social network.



Simulations

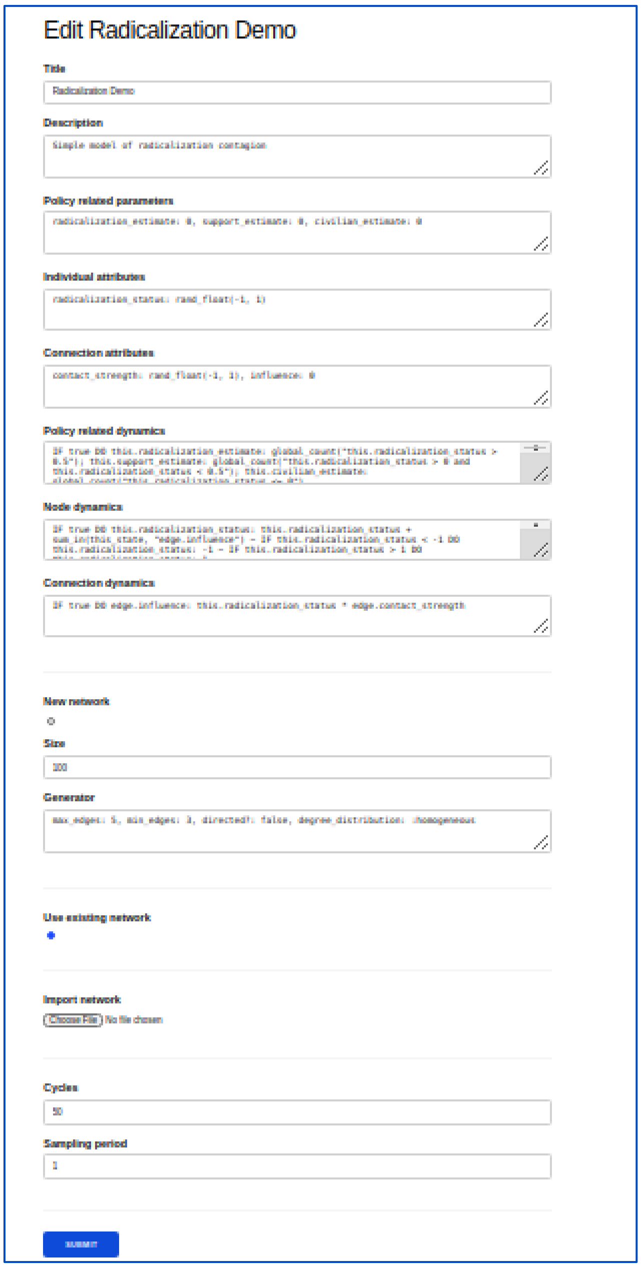
- Radicalization Demo - Simple model of radicalization contagion

FIGURE 23 - EXAMPLE SCREEN WITH LINKS TO ALL SIMULATIONS STORED IN POLITIKA

Selecting any of the listed simulations the user is able to inspect the simulation parameters. A top menu allows him to edit these parameters, run the simulation, examine its results, delete the simulation or perform analysis of these results

The Edit option extends the form described for the New simulation option with the additional options to:

- upload social network data from a user-supplied text file in JSON format using the Input Network field
- create synthetic data for a social network automatically by the system using the New radio button. In this case the user needs to specify the size of the social network (the number of nodes in the graph) and the Generator parameters that will be used for generating the network.



Edit Radicalization Demo

Title

Radicalization Demo

Description

```
Simple model of radicalization contagion
```

Policy related parameters

```
radicalization_estimate: 0, support_estimate: 0, civilian_estimate: 0
```

Individual attributes

```
radicalization_status: rand_float(-1, 1)
```

Connection attributes

```
contact_strength: rand_float(-1, 1), influence: 0
```

Policy related dynamics

```
IF true DO this.radicalization_estimate: global_count("this.radicalization_status >
0.5"); this.support_estimate: global_count("this.radicalization_status > 0 and
this.radicalization_status < 0.5"); this.civilian_estimate:
global_count("this.radicalization_status <= 0")
```

Node dynamics

```
IF true DO this.radicalization_status: this.radicalization_status +
sum_in(this_state, "edge.influence") - IF this.radicalization_status < -1 DO
this.radicalization_status: -1 - IF this.radicalization_status > 1 DO
this.radicalization_status: 1
```

Connection dynamics

```
IF true DO edge.influence: this.radicalization_status * edge.contact_strength
```

New network

○

Size

100

Generator

```
max_edges: 5, min_edges: 3, directed?: false, degree_distribution: :homogeneous
```

Use existing network

●

Import network

Choose File No file chosen

Cycles

50

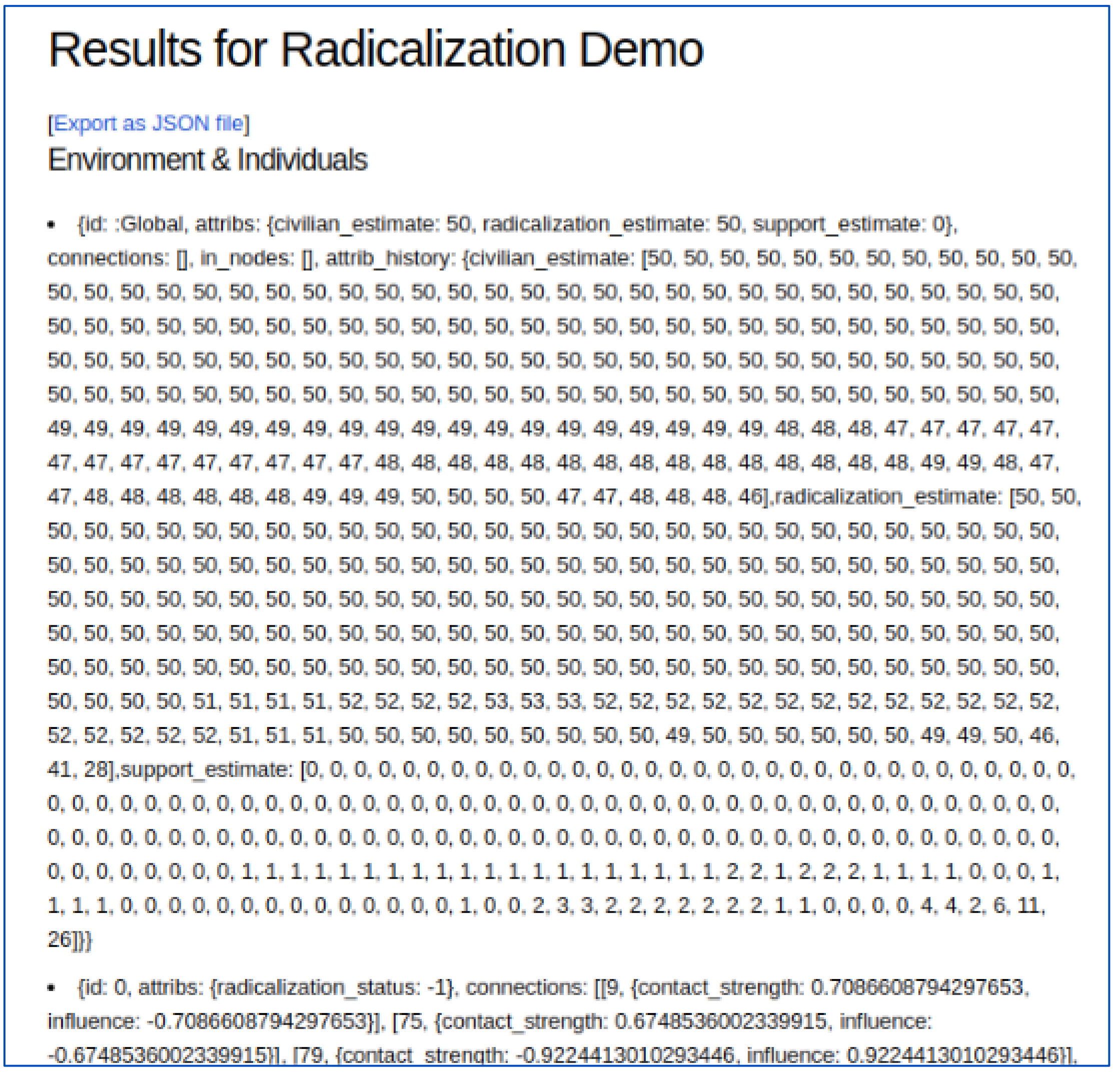
Sampling period

1

FIGURE 24 - EDITING FORM FOR A SIMULATION IN POLITIKA

The Delete option allows the user to delete a specific simulation from Politika after the user confirms this choice through a confirmation window displayed on the screen.

The Results option shows a list of nodes and their resulting attributes for each node in the social network after the simulation has finished. The details of each node are displayed in raw text form.



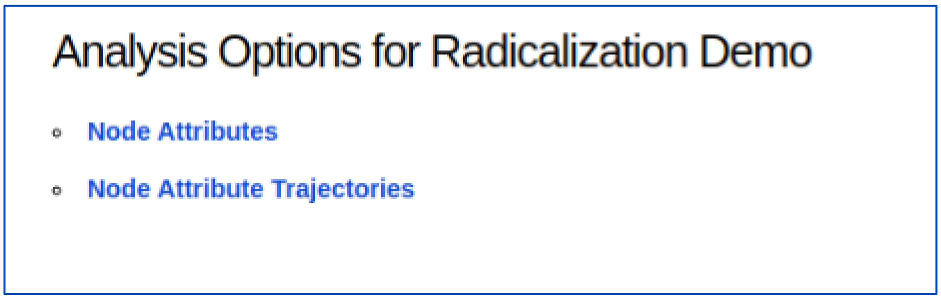
Results for Radicalization Demo

[Export as JSON file]

Environment & Individuals

- {id: :Global, attribs: {civilian_estimate: 50, radicalization_estimate: 50, support_estimate: 0}, connections: [], in_nodes: [], attrib_history: {civilian_estimate: [50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 48, 48, 48, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 49, 49, 48, 47, 47, 48, 48, 48, 48, 48, 48, 49, 49, 49, 50, 50, 50, 50, 47, 47, 48, 48, 48, 46],radicalization_estimate: [50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 51, 51, 51, 51, 52, 52, 52, 52, 53, 53, 53, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 51, 51, 51, 50, 50, 50, 50, 50, 50, 50, 50, 50, 49, 50, 50, 50, 50, 50, 50, 49, 49, 50, 46, 41, 28],support_estimate: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 3, 3, 2, 2, 2, 2, 2, 2, 2, 1, 1, 0, 0, 0, 0, 4, 4, 2, 6, 11, 26]}}
- {id: 0, attribs: {radicalization_status: -1}, connections: [[9, {contact_strength: 0.7086608794297653, influence: -0.7086608794297653}], [75, {contact_strength: 0.6748536002339915, influence: -0.6748536002339915}], [79, {contact_strength: -0.9224413010293446, influence: 0.9224413010293446}],

FIGURE 25 - SCREEN DEPICTING IN RAW FORM THE RESULTS OF RUNNING A SIMULATION

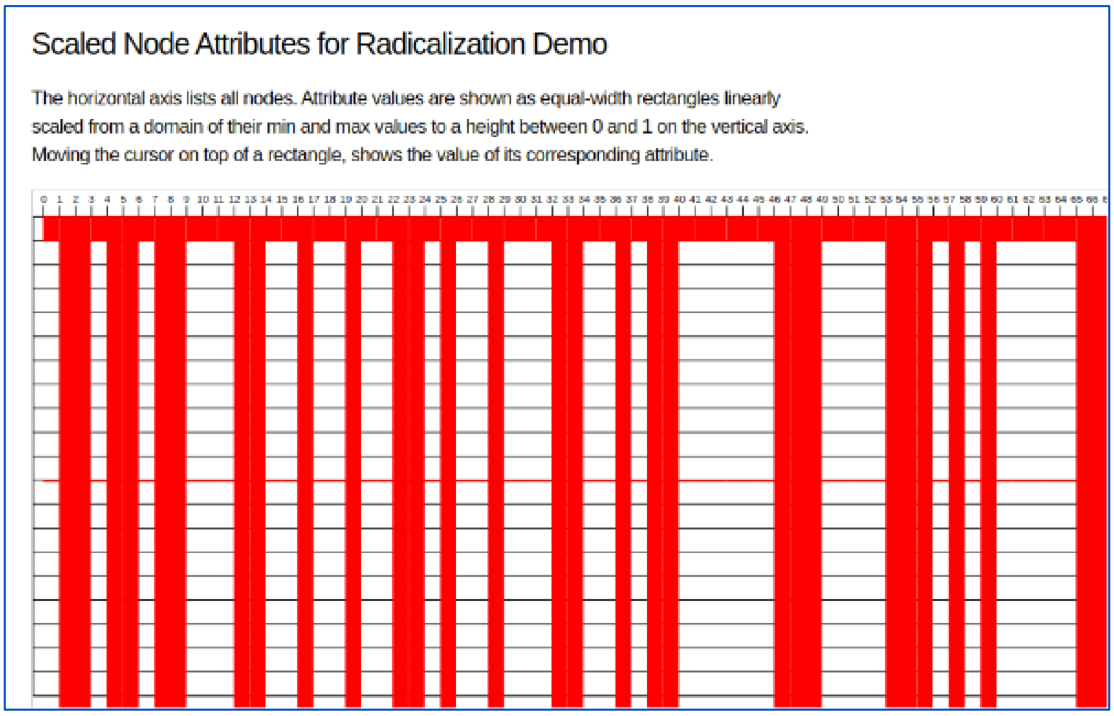


Analysis Options for Radicalization Demo

- Node Attributes
- Node Attribute Trajectories

FIGURE 26 - SCREEN DEPICTING THE ANALYSIS OPTIONS FOR THE RESULTS OF A SIMULATION

Analysis provides a screen with a set of options for a simulation that currently include the ability to chart the attributes of each node in the social network and to chart their change over time.



Scaled Node Attributes for Radicalization Demo

The horizontal axis lists all nodes. Attribute values are shown as equal-width rectangles linearly scaled from a domain of their min and max values to a height between 0 and 1 on the vertical axis. Moving the cursor on top of a rectangle, shows the value of its corresponding attribute.

FIGURE 27 - EXAMPLE SCREEN DEPICTING THE ATTRIBUTE VALUES FOR EACH NODE IN A SOCIAL NETWORK SIMULATED BY THE SYSTEM

2.2.6 Operational Data Repository

The operational data repository provides various interfaces for data connectivity that can be used in different scenarios. Firstly, it implements the JDBC interface, which is a standard specified under the Java JSR-221[5]. JDBC allows for a connection with the SQL query engine of the operational data repository, where the data user can exploit the SQL scripting language in order to perform complex query statements supported by the rich capabilities of the SQL itself. Currently, the JDBC driver of the operational datastore supports the execution of the standard SQL statements, under its current ISO SQL:2016.

Apart from the JDBC, an implementation of the OData[6] specification is provided, which is an OASIS standard that allows the connection to a data store via well-defined REST API. It allows for the integration of components that are not compatible with JDBC and cannot usually open direct connections to a datastore, rather than should communication via HTTP messages.

Finally, the operational data repository provides a direct api that allows to open connections directly to the storage engine of LXS. The latter provide rich query capabilities like SQL, however it cannot support JOIN operations. The benefit for connecting directly to the storage is that data connections and query executions are performed much more efficiently, as they bypass the query engine layer. This is crucial in cases where very low latency is a requirement, such as the support for data ingestion in very high rates. At this phase, this direct-api is written in C and it is still closed and it is available only to the query engine itself that makes use of it via a JNI binding. Currently,

[5] https://jcp.org/en/jsr/detail?id=221
[6] https://www.odata.org/documentation/

there is a work in progress to implement a Java interface similar to what document-based datastores, such as MongoDB provides. However, this is still under implementation and therefore it has not been included in the prototype and will be reported in the next version of this deliverable. In order for the data scientist to exploit the benefits of the direct connection to the storage and the lower latency of the query execution, there has been implemented various connectors to popular data/streaming processing frameworks, such as Apache Spark and Apache Flink accordingly. Due to this, the data scientist can use the API of those frameworks, and those will make use of the direct connections, through the connectors implemented and provided by the operational datastore.

2.3 Baseline technologies and tools

This section describes the baseline technologies and tools under the delivered components.

2.3.1 DAA API Gateway

2.3.1.1 KUBERNETES

Kubernetes [6] is the leading open source container management platform used in production of growing number of enterprises as the base of private on-prem cloud, as well as offered by almost all cloud providers as a managed dedicated cluster, and as described in D4.1 [1] it is the chosen underlying platform for all PolicyCloud components. The underlying software components of DAA, both OpenWhisk and kafka, will be deployed on a Kubernetes cluster once the PolicyCloud testbed is available. In the prototype both components are deployed on internal IBM Kubernetes cluster using helm.

2.3.1.2 KAFKA

Apache Kafka [7] is used as a transformation buffer to store intermediate data between components in reliable and asynchronous way. In SW1 communication between functions is done through OpenWhisk, in the future in case the OpenWhisk limitations of input and output size (1MB) to actions would not be adequate, Kafka will be used as the intermediate buffering between functions as well.

LeanXcale database interface would be implemented in the components integration phase using Kafka connector to insert the PolicyCloud processed datasets into the LeanXcale database backend (https://www.leanxcale.com/new-blog/2020/5/26/kafka-connector-for-leanxcale-part-2).

2.3.1.3 OPENWHISK

Apache OpenWhisk [8] is an open source light-weight serverless platform that provides capability to deploy functions (written in any language) and specify triggers and rules by which the functions are executed. It offers a simple programming model where the function developer can concentrate only on the mere logic of the function, while the deployment and activation details are taken care transparently. It is based on containers as the functions' wrapper, and deploys and integrates perfectly on a Kubernetes environment. All analytic functions in PolicyCLOUD will be deployed as OpenWhisk actions, with appropriate triggers and activation rules, addressing the extendibility and reusability requirements. Additional important capability of OpenWhisk is the composition of functions to be activated per trigger/rule, which enable to specify multiple analytics to be applied in series to a data source.

The DAA APIs are implemented as a set of serverless functions (in the OpenWhisk cluster), web actions were written to expose functions and data sources APIs with REST, implemented in node.js

Packages and package-bindings were implemented to store metadata and pass it as parameters in functions invocations.

Triggers, rules and kafka-feed were used to subscribe to topic and apply data transformation functions as data enters the platform and is available.

Analytic functions are implemented using OpenWhisk actions. Sample reference functions were written in java and packaged in jar according to OpenWhisk actions guidelines and implement connectivity to LeanXcale database as backend.

2.3.2 Enhanced Interoperability & Data Cleaning

2.3.2.1 Data Cleaning Component

The Data Cleaning component is developed with Python 3.7 using the Flask python micro web framework[7]. Flask is a powerful framework written in Python and based on the Werkzeug[8] toolkit and Jinja2[9] template engine that is independent from particular libraries or tools and that supports a large list of extensions for application features. Besides the Flask framework, a list of libraries and tools has been used in the context of the Data Cleaning to support several functionalities of the component. For the data structure handling Pandas[10] has been selected, while NumPy[11] is used for all the numerical computations.

2.3.2.2 Enhanced Interoperability Component

For the implementation and utilization of the first sub-component of the Enhanced Interoperability component different technologies and techniques from the fields of NLP and Semantic Web were utilized. To this end, the overall code was implemented based on Python3 [9] programming language. Moreover, spaCy [10] framework was utilized to perform concrete and high accuracy NLP sub-tasks, such as Tokenization, Sentence splitting, POS tagging, NER etc., which are core tasks in the provided sub-component and their importance and overall functionality have introduced in previous sections. On top of this, NLP tasks integrate with Semantic Web technologies in order to provide a hybrid approach through the utilization of state-of-the-art Semantic AI approach. To this end, SPARQL [11], a widely used RDF query language, is being utilized. The latter is being used to perform queries on Wikidata Knowledge base to identify and interlink appropriate entities based on the ones that have already been recognised from the raw texts.

2.3.3 Situational Knowledge Acquisition & Analysis

As it has been described in the previous section, this component has been built using Java, Maven, JSON and LXC storage system.

2.3.4 Opinion Mining & Sentiment Analysis

As described in previous sections, this component has been built as a data workflow using Apache NiFi, a powerful tool that allow us to create reusable modules and use them in data workflows in an easy way. In collaboration with Apache NiFi Registry to maintain the versions of the workflows developed, and to easy the process of exporting and importing the workflows in an automated way.

[7] Flask, http://flask.pocoo.org/
[8] Werkzeug, http://werkzeug.pocoo.org/
[9] Jinja2, http://jinja.pocoo.org/
[10] Pandas, https://pandas.pydata.org/
[11] NumPy, http://www.numpy.org/

To execute the Apache NiFi workflows in an automated way without the GUI, the Python client SDK NiPyApi[12] has been used. This client permits to connect to different Apache NiFi instances, export and import workflows, execute and configure those workflows, and get the status of them, among others.

For the microservice, it has been developed in Python with the help of the library Flask to be able to call the different functions using a REST API approach. It has been containerized in Docker to easy the deployment in different environments, and to facilitate updates or additions of functions without any change in the workflows.

In the sentiment analysis function, two Python libraries have been investigated to perform the sentiment of the tweets: Vader and Textblob. It is not limited to them, as other libraries and models should be integrated in next phases.

2.3.5 Social Dynamics & Behavioural Data Analytics

Politika has been implemented using the following baseline technologies:

- Elixir/Erlang (https://elixir-lang.org/)
- Phoenix Web Framework (https://www.phoenixframework.org/)
- D3.js (https://d3js.org/)

2.3.6 Operational Data Repository

The operational data repository consists of various components and tools, each one of those is built over different baseline technologies and tools. More precisely, it consists of:

- The adaptable and distributed storage engine: It has been written in C and provides the capabilities for persistent storage of data elements.
- The transactional processing mechanism: It enables support for transactions and ensures ACID properties. It can be scaled out linearly and it has been completely implemented in Java 8.
- The query engine: It provides support for executing queries written in SQL scripting language. It has been implemented in Java 8 and makes use of the Apache Calcite query processing framework, which lies in its core and extends its functionality to provide fully SQL support.
- The monitoring engine: It makes use of the Prometheus monitoring framework for gathering metrics and monitoring information by each of its components. All components expose monitoring information to specific URIs where custom exporters are placing data that can be imported by the Prometheus. Additionally, the openwizard.io framework is being used for exporting custom monitoring information (i.e. query compilation time) via JMX extensions.
- The monitoring dashboard: It makes use of the Grafana Dashboard for visualizing the monitoring information gathered by the Prometheus.
- Ansible: An ansible installer is provided that automatically installs various components in a target infrastructure (either a virtual one via the use of containers, or over bare metal)

[12] https://nipyapi.readthedocs.io/en/latest/index.html

3 Source code

3.1 Availability

This section provides the links and access information to the actual code repositories.

3.1.1 DAA API Gateway

Software prototype code can be found on the PolicyCLOUD's github which is accessible to the members of the consortium and includes code to deploy in OpenWhisk.

http://snf-877903.vm.okeanos.grnet.gr/oshrit/daa

3.1.2 Enhanced Interoperability & Data Cleaning

3.1.2.1 Data Cleaning Component

The software prototype of the Data Cleaning is provided in PolicyCLOUD's GitLab repository and can be found under the URL http://snf-877903.vm.okeanos.grnet.gr/argyro/cleaning-component

3.1.2.2 Enhanced Interoperability Component

Software prototype code can be found on the PolicyCLOUD's Gitlab repository, which is accessible to the members of the consortium. As analyzed before the overall code has been deployed and executed in IPython Notebook (.ipynb file extension), which is accessible under this specific URL

http://snf-877903.vm.okeanos.grnet.gr/george/interoperability-component

3.1.3 Situational Knowledge Acquisition & Analysis

The code is available at: http://snf-877903.vm.okeanos.grnet.gr/nines/heatmap and has the following folder structure:

- src/main/java/atos/ari/data/policyCloud/heatmap → it contains all the source code of the prototype.
- pom.xml → maven artefact for an easy integration/reusability of the source code.

3.1.4 Opinion Mining & Sentiment Analysis

The code is available at http://snf-877903.vm.okeanos.grnet.gr/jorge/sentimentanalysis and has the next folder structure:

- **Root**: description of the git project and details in the requirements and steps needed to deploy it.
- **Functions**: folder that contains the python code to execute the sentiment and the aggregates functions.
  - nifi-sentiment-aggregator-local.py
  - nifi-sentiment-ingest.py
- **Installation**: folder with a docker-compose file to deploy an Apache NiFi instance.
- **Microservice**: folder with the files needed to run a Docker container with the microservice.
- **Scripts**: three python files to export/import/execute an Apache NiFi workflow, and one python file to set the Apache NiFi Registry.
  - nifi-execute-flow.py

- nifi-export-flow.py
- nifi-import-flow.py
- nifi-add-registry.py

- **Templates**: the exported version of the two Apache NiFi workflows that represent the two functions described in the component.
    - nipyapi-sentiment-aggregator-local-flow.json
    - nipyapi-sentiment-ingest-flow.json

3.1.5 Social Dynamics & Behavioural Data Analytics

The code for the current Politika prototype resides in:

https://urldefense.proofpoint.com/v2/url?u=http-3A__epinoetic.org_Politika_politika-5Fdemo.tar.gz&d=DwIDaQ&c=jf_iaSHvJObTbx-siA1ZOg&r=rD9swZzFdAz-3cT2OKIhpw&m=9FaeL53yxUXGs6uX6wSYsiLuLG5Ph2yNEuFere5mVM8&s=a7Im2TlliQu_EboFwlqqPGnd-_XgvcLsYQ12rVXEL1o&e=

3.1.6 Operational Data Repository

The source code of the operational datastore is based on the background technology that LXS is bringing to the project, and therefore, it is under propitiatory rights of the company. Due to this, no source code for this component will be provided to the consortium. Instead, LXS provides precompiled and pre-packaged binaries that can be used instead, containing all functionality developed so far. The binaries are accessible to the members of the consortium and have been uploaded in the private gitlab repository that can be found here: http://snf-877903.vm.okeanos.grnet.gr/pavlos/lxs-store

3.2 Exploitation

This section provides the information about where components are deployed and how they can be can be accessed and run.

3.2.1 DAA API Gateway

Setup of DAA prototype environment:

1. Install Kubernetis cluster [13]
2. Install OpenWhisk and kafka on the k8s cluster [14]
3. Follow Leanxcale installation in section 3.2.6, run the docker image as docker run -d -p 1529:1529 --name datastore policy-store:latest
4. Setup OpenWhisk environment:
    4.1. Git clone DAA code from Section 3.1.1
    4.2. Create namespace policycloud
    4.3. Create package daa
    4.4. Create the web actions components for data source and analytic functions:

[13] https://kubernetes.io/docs/tasks/tools/
[14] https://github.com/apache/openwhisk-deploy-kube

4.4.1. wsk package update daa -p brokers '[\"10.111.3.60:9092\"]' -p accessToken 897df16803b16d34a750c79fde5d120d2dff31d5 -p repo policycloud -p branch master -p owner OSHRITF (Supply the gitlab credentials to the package creation)
4.4.2. to pull node-modules run: npm install .
4.4.3. create analytic function web action zip file: zip –r action.zip
4.4.4. wsk action update /policycloud/daa/analytic-function ./analytic-function/action.zip --kind nodejs:10 -a provide-api-key true --web true
4.4.5. create data sources web action file: zip –r action.zip
4.4.6. wsk action update /policycloud/daa/datasource ./datasource/action.zip --kind nodejs:10 -a provide-api-key true --web true

Reference sample functions can be found in the repository, as java source code and jar packaged for use in OpenWhisk, and were used in the prototype demonstration:

- SampleAnalyticFunction – used as ingest function to simulate dataset record "read" and transformed from the GTD usecase data source csv url file. Sequenced by kafka-producer, the json record is placed on output topic which in turn is written into the backend Leanxcale database using LeanXcale Kafla connector.
- DisplayAnalyticFunction – used as function to retrieve dataset from the backend Leanxcale backend database and return as json.

3.2.2 Enhanced Interoperability & Data Cleaning

3.2.2.1 Data Cleaning Component

The Data Cleaning software prototype is a Python project. As a result, in order to be able to run the prototype manually, Python should be properly preinstalled and preconfigured on the system and PYTHONPATH should include all the modules of the Data Cleaner prototype. In order to facilitate the exploitation process, the source code has to be downloaded from the repository, and then has to be opened as a Python project in the preferred IDE. On top of that, since two (2) of the used modules are not pre-installed, the user has to manually install them. To do so, the user has to use the de facto standard package-management system used to install and manage software packages written in Python (pip). In that case, the user has to run a cmd.exe terminal with the current Python environment, and write the following commands:

Installing pandas-schema

$ pip install pandas-schema

Installing flask

$ pip install flask

As soon as this process gets complete, the user has to simply run the program in a development server, since it is the first version of the prototype, and visit the following URL in a web browser:

http://127.0.0.1:5000/cleaning

This will start the Data Cleaner prototype as a web application and the exposed external interface will be available at *localhost:5000*.

3.2.2.2 Enhanced Interoperability Component

As analyzed in the previous section several dependencies and tools must be fulfilled and installed in order the provided code to be executed. At first there are some prerequisites that need to be fulfilled in order someone to be able to open and execute a IPython Notebook. The latter can easily be executed through Jupyter Notebook [12] tool which is available both as standalone software and as inbuild software in Anaconda tool [13]. Moreover, Python3 and pip installer [14] should have been installed in the used machine.

Finally, below there is a list of all the wanted tools and libraries and their installation guides.

Installing spaCy

SpaCy can easily being installed via a pip installer, which is utilized to install Python libraries, and the following statement:

$ pip install -U spacy

Otherwise if Anaconda tool is already installed in the machine the following command on the Anaconda prompt needs to be executed:

$ conda install -c conda-forge spacy

Once the spaCy has been downloaded and installed, the next step is to download the language model, which in this case is the English language model. The language model is utilized to perform a variety of NLP tasks, which were introduced in previous sections. To this end, the following command downloads the appropriate language model:

$ python -m spacy download en

Installing qwikidata

A Python package with tools that allow to integrate and interact the provided data with Wikidata. The package defines a set of classes that allow to represent Wikidata entities, which is being utilized by Interoperability component in order to interlink recognized entities from raw texts with Wikidata entities with the ultimate goal to extract JSON URI annotated data. This specific tool can be executed only in machines and environments with installed Python 3.5 or latter version. Moreover, this package incorporates SPARQL and gives the availability to the end user to utilize SPARQL queries.

The most recent version of this package can be installed by using pip installer, *$ pip install qwikidata*

3.2.3 Situational Knowledge Acquisition & Analysis

For this prototype, the functionalities described for this component have been deployed in a private environment. However, it can be replicated following the instructions and files provided in the git repository from Section 3.1.3

3.2.4 Opinion Mining & Sentiment Analysis

For this prototype, the functionalities described for this component have been deployed in a private environment. However, it can be replicated following the instructions and files provided in the git repository from Section 3.1.4.

3.2.5 Social Dynamics & Behavioural Data Analytics

Politika is currently deployed at the GRNet cloud. It can be accessed at: http://epinoetic.org:4000

The installation instructions that follow refer to a Linux machine and have been tested with the Ubuntu distribution.

- To install Erlang, Elixir, Phoenix and Postgres follow the instructions at:

  - https://hexdocs.pm/phoenix/installation.html
- Extract the politika_demo.zip file. A politika_demo directory will appear at the site of extraction.
- Open with a text editor (e.g. gedit) the politika_demo/config/config.exs file and update the following text section with the credentials for the Postgres installation in your system assuming that Postgres runs locally using port 5432 (otherwise update the fields hostname: and port: accordingly):
  - config :politika, Politika.Repo,
  - database: "politika",
  - username: "XXXXXXXXX",
  - password: "XXXXXXXX",
  - hostname: "localhost",
  - port: "5432"
- Open a terminal window go to the politika_demo directory and setup the database from within ecto by typing:
  - mix ecto.create
  - mix ecto.migrate
- Run the politika system by typing in the politika_demo directory:
  - mix phx.server
- The system will start running in your machine at port 4000. For example, you can access Politika by typing the following address in your browser:
  - http://<Your IP address>:4000

3.2.6 Operational Data Repository

In order to use the operational data repository, it is highly recommended to make use of a docker container. Firstly, the user needs to download the current version of the binaries by executing the following:

git clone http://snf-877903.vm.okeanos.grnet.gr/pavlos/lxs-store

Then assuming the docker is already installed in the host machine, she needs to create the corresponding docker image, by executing the following:

docker build –t lx-store .

After building the image, she can check that it is available in the host machine's docker catalogue. To start the container, she would need to execute the following:

docker run –d –p 1529:1529 –name lx-store {image_id}

This will create a single container running in the background where the data store will be installed and start in an all-in-one fashion. It is important to notice the –p 1529:1529 parameters that exposes the 1529 port to the host machine, so that it can be reachable by other process outside of the container that have access to the network of the host machine.

To connect to the container via the console, she should execute the following:

docker exec –it lx-store bash

Then she should able to verify that all components are up and running, by executing the following:

${BASEDIR}/LX-BIN/bin/lxConsole 3

4 Conclusion

We provided in this a description of the software demonstration for the components of the Integrated Data Acquisition and Analytics (DAA) Layer, which provides the analytical capabilities of the PolicyCloud platform. At this phase of the project the demonstrators are provided at component level and not integrated in end-to-end scenario. The integration of all components on the project's testbed with actual end to end use cases will be provided in the next SW demonstrator deliverable – D4.4.

References

[1] PolicyCLOUD. *D4.1* - Reusable Model & Analytical Tools: Design and Open Specification 1. Biran Ofer. 2020

[2] PolicyCLOUD, D2.2 CONCEPTUAL MODEL & REFERENCE ARCHITECTURE, 2020.

[3] PolicyCLOUD. *D6.3* - Use Case Scenarios Definition & Design. Sancho Javier. 2020

[4] Wikidata, https://www.wikidata.org/wiki/Wikidata:Main_Page

[5] Xin J., Afrasiabi C., Lelong S., Adesara J., Tsueng G., Su A. I., and Wu C., Cross-linking BioThings APIs through JSON-LD to facilitate knowledge exploration, BMC bioinformatics, 19(1), 30, https://doi.org/10.1186/s12859-018-2041-5, 2018

[6] Kubernetes, https://kubernetes.io

[7] Kafka, https://kafka.apache.org

[8] OpenWhisk, https://OpenWhisk.apache.org

[9] Python, https://www.python.org

[10] spaCy, https://spacy.io/

[11] SPARQL, https://www.w3.org/TR/rdf-sparql-query

[12] Jupyter, https://jupyter.org/

[13] Anaconds, https://www.anaconda.com/

[14] PIP Installer https://pip.pypa.io/en/stable/