



Policy Cloud  
Cloud for Data-Driven Policy Management

## CLOUD FOR DATA-DRIVEN POLICY MANAGEMENT

Project Number: 870675

Start Date of Project: 01/01/2020

Duration: 36 months

### D3.8 CLOUD INFRASTRUCTURE INCENTIVES MANAGEMENT AND DATA GOVERNANCE SOFTWARE PROTOTYPE 3

Dissemination Level	PU
Due Date of Deliverable	31/10/2022, Month 34
Actual Submission Date	26/10/2022
Work Package	WP3 Cloud Infrastructures Utilization & Data Governance
Task	T3.1, T3.3, T3.4, T3.6
Type	Demonstrator
Approval Status	
Version	1.0
Number of Pages	p.1 – p.63

**Abstract:** This document is an accompanying report providing information about the demonstrator of the third version of the Cloud Infrastructure, Incentives Management and Data Governance software prototype. It describes the cloud gateways and APIs, the cloud provisioning mechanisms, the implemented version of the algorithms as well as the data governance tools according to the D3.1, D3.4 and D3.7 specifications.

The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided "as is" without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/ her sole risk and liability. This deliverable is licensed under a Creative Commons Attribution 4.0 International License.



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	08/09/2022	ToC	Konstantinos Oikonomou
0.2	20/09/2022	ATOS Contributions	Ana Luiza Pontual
0.3	27/09/2022	UBITECH Contributions	Konstantinos Oikonomou
0.4	29/09/2022	UPRC Contributions	George Manias
0.5	29/09/2022	EGI Contributions	Sebastian Luna-Valero
0.6	30/09/2022	Contributions Integration and finalization	Konstantinos Oikonomou
0.7	24/10/2022	Internal Review	Argyro Mavrogiorgou, Kostas Nasias
0.8	25/10/2022	Internal Review Comments Addressing	Konstantinos Oikonomou
0.9	26/10/2022	Quality Check	Argyro Mavrogiorgou
1.0	26/10/2022	Ready For Submission	Konstantinos Oikonomou, Giannis Ledakis

## Author List

Organisation	Name
ATOS	Maria Angeles Sanguino, Ana Luiza Pontual, Miquel Milà, Ricard Munné
EGI	Sebastian Luna-Valero
UBITECH	Giannis Ledakis, Konstantinos Oikonomou
UPRC	George Manias, Argyro Mavrogiorgou

## Abbreviations and Acronyms

Abbreviation/Acronym	Definition
ABAC	Attribute Based Access Control
API	Application Programming Interface
EC	European Commission
EOSC	European Open Science Cloud
GUI	Graphical User Interface
OIDC	OpenId Connect
PAP	Policy Administration Point
PDP	Policy Decision Point
PDT	Policy Development Toolkit
PEP	Policy Enforcement Point
PIP	Policy Information Point
XACML	eXtensible Access Control Markup Language

# Contents

Versioning and Contribution History.....	2
Author List.....	2
Abbreviations and Acronyms .....	3
Executive Summary .....	8
1 Introduction .....	9
1.1 Structure of the document.....	9
1.2 Summary of Changes .....	9
2 Cloud Gateways & APIs for Efficient Data Utilization .....	10
2.1 Updates since D3.5 .....	10
2.2 Prototype Overview .....	10
2.3 Main Components of the Prototype .....	12
2.3.1 Microservices .....	12
2.3.2 Monitoring – Metrics .....	14
2.4 Interfaces .....	15
2.4.1 Monitoring.....	15
2.4.2 Microservices .....	16
2.5 Baseline Technologies and Tools.....	28
2.5.1 MoleculerJS.....	28
2.5.2 Traefik.....	28
2.5.3 Docker .....	29
2.5.4 File Parsers .....	29
2.6 Source Code .....	30
2.6.1 Code Overview and Availability.....	30
2.6.2 NPM scripts .....	30
2.7 Deployment Status .....	31
3 Incentives Management.....	32
3.1 Updates since D3.5 .....	32
3.2 Prototype Overview side.....	32
3.3 Main Components of the Prototype .....	34
3.4 Interfaces .....	36

3.5	Baseline Technologies and Tools.....	39
3.6	Source Code .....	39
	Code Overview and Availability .....	39
3.7	Deployment Status .....	40
4	Data Governance Model and Privacy Enforcement mechanism.....	41
4.1	Updates since D3.5 .....	41
4.2	Prototype Overview .....	41
4.3	Main Components of the Prototype .....	42
4.3.1	ABAC Server .....	42
4.3.2	KeyCloak .....	42
4.3.3	ABAC Client Filter .....	47
4.3.4	ABAC Proxy.....	48
4.3.5	Test Web Client.....	49
4.3.6	EGI Check-In integration.....	50
4.3.7	XACML Editor .....	52
4.3.8	Custom Access Policies for Gateways .....	53
4.4	Interfaces .....	55
4.5	Baseline Technologies and Tools.....	56
4.5.1	Balana.....	56
4.5.2	Keycloak.....	58
4.6	Source Code .....	59
	Code Overview and Availability .....	59
4.7	Deployment Status .....	60
5	Conclusions.....	61
	References.....	62

## List of Tables

Table 1 - Keycloak Common APIs .....	55
--------------------------------------	----

## List of Figures

Figure 1 - Cloud gateways and apis architecture and integrations.....	11
Figure 2 – Swagger-Stats Summary UI .....	15
Figure 3 – Swagger-Stats Request Page .....	16
Figure 4 – MolecularJS – Service Discovery.....	16
Figure 5 – MolecularJS – Service Registry.....	17
Figure 6 - OpenAPI specification page.....	18
Figure 7 - GTD'S Swagger OpenAPI Interface.....	19
Figure 8 – Tweets Filtered-SwaggerUI .....	20
Figure 9 – Twitter Streaming Process .....	21
Figure 10 - Aragon swagger ui.....	22
Figure 11 - London Swagger UI.....	23
Figure 12 – Sofia Road Swagger UI.....	24
Figure 13 - Sofia Transport Swagger UI.....	25
Figure 14 - Sofia Waste Swagger UI .....	26
Figure 15 - Sofia Parking Swagger UI.....	26
Figure 16 - Sofia Violation Of Public Order Swagger UI.....	27
Figure 17 – Sofia Air Quality Swagger UI.....	27
Figure 18 - Rand microservice swagger UI .....	28
Figure 19 – Traefik Web UI .....	29
Figure 20 - Access to Incentive Management Tool options .....	34
Figure 21 - Initial page for Incentive's option.....	34
Figure 22 - Form to create a new Incentive .....	35
Figure 23 - Form to create a new Action.....	35
Figure 24 - Swagger for Incentive Management Back End component .....	38
Figure 25 - docker-compose.yaml file .....	43
Figure 26 - Keycloak Realm creation.....	43
Figure 27 - Keycloak Client Creation .....	44
Figure 28 - Keycloak Realm Roles.....	44
Figure 29 - Keycloak Users .....	45
Figure 30 - Initial Role Mappings .....	45
Figure 31 – Admin Role Grant .....	46
Figure 32 - Custom User Attribute .....	46
Figure 33 - Client Attribute Mapper .....	47
Figure 34 – ABAC proxy application.yml .....	49
Figure 35 - Intercept Login .....	50
Figure 36 - Successful Attributes Retrieval .....	50

Figure 37 - Integrated EGI Check-In .....	51
Figure 38 - Sample access policy in editor.....	52
Figure 39 - Sample access rule in editor .....	53
Figure 40 - ABAC Custom Rule For Access During Shift.....	54
Figure 41 - Balana PDP .....	56
Figure 42 - Carbon Policy Filter .....	57
Figure 43 - Carbon Attribute Finder .....	58
Figure 44 - OIDC Signalling .....	59

## Executive Summary

The third and final version of the Cloud Infrastructure, Incentives Management and Data Governance software prototype includes the cloud gateways and APIs, the cloud provisioning mechanisms, the implemented version of the algorithms as well as the data governance tools according to the D3.1 [1], D3.4 [2] and D3.7 [3] specifications and is built upon the first and second versions of the prototype described in D3.2 [4] and D3.5 [5] respectively. The prototype's cloud infrastructure is supported by RECAS-BARI and is utilized by EGI through cloud gateways. These gateways allow the prototype to gather data from heterogeneous data sources, such as Twitter and the global terrorism database and have integrated microservices to serve the needs of the different PolicyCLOUD pilots.

The final version of the Incentives Management tool is also provided in this deliverable. This final version has been integrated with the Policy Development Toolkit (PDT) and has been deployed in the EGI Cloud.

This third version of the prototype also includes the latest updates of the ABAC based access control mechanism and the Keycloak integration. This is broken down to 8 key components that have been combined to provide fine-tuned and secure access control and authentication. Specifically, Keycloak has been integrated with the Marketplace, the Gateways and the PDT and custom access policies have been developed for the gateways microservices. Finally, both the introduction of a XACML editor to ease access policies creation and the integration of EGI Check-in, an alternative way to authenticate to the prototype with academic and social credential, enhance the user experience in the PolicyCLOUD platform



# 1 Introduction

This document is an accompanying report for the third and final iteration of “Cloud Infrastructure Incentives Management and Data Governance: Software Prototype” and is the eighth deliverable of WP3, covering tasks T3.1, T3.3, T3.4, and T3.6. Based on the design and the specifications provided in D3.1 [1], D3.4 [2] and D3.7 [3], all task participants finalized their collaboration and implementation of their corresponding outcomes that were utilized and integrated into the final stages of the platform in later stages. In the scope of *T3.1 - Cloud Provisioning of the PolicyCLOUD Infrastructure*, INDIGO-DataCloud PaaS Orchestrator continues to be the tool of choice, as reported in D3.2 [4], and thus no further mention or update is required. In the scope of *T3.3 - Cloud Gateways & APIs for Efficient Data Utilization* the final prototype of the cloud gateways with the latest updates has been described. Regarding *T3.4 - Incentives Management*, development has ceased based on the reviewers’ recommendation, but integration actions have taken place and are also reported. Finally, for *T3.6 - Data Governance Model, Protection and Privacy Enforcement* the third and final prototype of the mechanism that is used for privacy enforcement and the protection of data is described, along with all the integration effort and a new XACML Editor that was. For all these tasks, this document provides the work performed until M34.

## 1.1 Structure of the document

The rest of the document is structured as follows. Section 2 presents the Cloud Gateway components of the final prototype, while Section 3 describes the Incentives Management prototype for the efficient utilization of citizen and policy maker data. Section 4 presents the Data Governance model and the Privacy enforcement mechanism for the security of the prototype. Finally, Section 5 provides the conclusion of the document.

## 1.2 Summary of Changes

Concerning the project’s Cloud Gateways and APIs component several core updates and changes have occurred compared to D3.5 [5], as additional microservices and functionalities have been implemented to cover the needs of the pilot scenarios. More specifically, two new microservices with their corresponding APIs have been developed for the SOFIA scenario, one for the Maggioli radicalization scenario and one for the Aragon use case. Specifications for these microservices were finalized in D6.11 [6] and details about these changes will be further analyzed in Section 2. Furthermore, the APIs introduced in D3.2 and extended in D3.5 have been further enhanced with more API endpoints, where each one triggers the parsing of respective datasets. Regarding the Incentives Management, both integration with the Policy Development Toolkit (PDT) and the deployment of the Incentives Management backend in the EGI Cloud Infrastructure have been completed. Finally, the Privacy Enforcement mechanism has been also integrated with the PolicyCLOUD marketplace, a new XACML editor has been developed for easier creation of access policies and custom access policies are available for the new APIs mentioned in Section 2.

## 2 Cloud Gateways & APIs for Efficient Data Utilization

The Cloud Gateway and API component will enhance the abilities and services offered by a unified Gateway to move streaming and batch data from data owners into PolicyCLOUD data stores layers. Hence, it seeks to offer a unified framework for various microservices that are responsible for obtaining data from external data sources e.g., 3rd party APIs, files, streams etc. Based on the specifications provided in D3.4 [2] of the PolicyCLOUD project, the effort related to Cloud Gateways & APIs component will be focused on providing a complete and “smart” entryway into PolicyCLOUD project, allowing multiple APIs or microservices to act cohesively and thus provide a uniform, gratifying experience to each stakeholder. To this end, single API endpoints will be provided to its stakeholder in order to access all the data obtained from the external data sources in JSON format without having to care about complex integrations.

### 2.1 Updates since D3.5

As the project progresses additional microservices and functionalities have been implemented within the Cloud Gateways & APIs component. In this respect, two additional microservices and corresponding APIs have been introduced with regards to the Sofia scenario and the ingestion of several datasets that have introduced under the scopes of these scenarios. More specifically, corresponding microservice has been implemented to serve the integration of datasets provided by two different data sources with regards to the Scenario B (SC2) “Environment and Air quality”, as presented in the context of D6.11 [6]. The latter enhances the analysis for the Air Quality of the city of Sofia through a concrete processing and analysis of data derived from different datasets. The second added microservice is related with the Scenario B (SC6) “Violation of Public Order” of Sofia pilot, as presented in the context of D6.11. Moreover, an additional microservice has been created to support the ingestion of the RAND Database of Worldwide Terrorism (RAND) and the related scenarios from Maggioli use case, as also introduced in the context of D6.11. Finally, the Aragon microservice has also implemented and utilized in the context of this 3<sup>rd</sup> Software prototype to support the scopes and objectives of the Scenario A.2 Price Evolution of the SARGA use case, as presented in the context of D6.11.

### 2.2 Prototype Overview

In the context of this deliverable, the 3<sup>rd</sup> Software Prototype of the Cloud Gateways & APIs component consists of six main microservices. The initial design of these six specific microservices, that can be used either for fetching data from an external file or from a social media platform like Twitter, includes complete workflows and pipelines for pushing data into the PolicyCLOUD platform. These five sub-components base their functionality on all four different Use Cases according to D6.11 [6] and are being described into the below subsections. As also introduced in the previous deliverable all sub-components integrate with PolicyCLOUD’s authentication mechanism, the Keycloak, which is further introduced and analysed in Section 4 of this specific deliverable. The integration between Cloud Gateways & APIs component and the User Authorization mechanism ensures that all the required security standards are

being met and controls the access to the Cloud Gateways. More specifically, by using the *KeycloakConnect* npm package and a custom middleware for MoleculerJS, the integration with the Keycloak has been established in order to access and authenticate each microservice with an access token that should be obtained by the Keycloak service and must be used in every future request as Authorization header. To this end, only authorized users can have access on Cloud Gateway's resources. Moreover, as introduced in Section 1.2 the Cloud Gateways and APIs component integrates with the project's Interim Repository and acts as project's repository where each stakeholder and data provider have the ability to upload their own datasets. This integration has a two-fold advantage. In one hand it offers the ability of not having to set up a potentially high number of adaptors in order to fetch data from various kinds of data sources, while on the other hand use case partners are able to upload fast and easy their respective datasets, which are in various file formats (e.g., xlsx and csv) and which can further be fetched from the Cloud Gateways component. Furthermore, in terms of providing an end-to-end ingestion pipeline the Cloud Gateways component integrates directly with the PolicyCLOUD's identified message bus, the Kafka message broker, and the PolicyCLOUD's identified serverless platform, the OpenWhisk, where functions are activated on demand. In the figure below, the overall architecture of the Cloud Gateways component as also the integrations with other project's tools and components (e.g., OpenWhisk, Kafka, and Interim Repository) are being depicted.

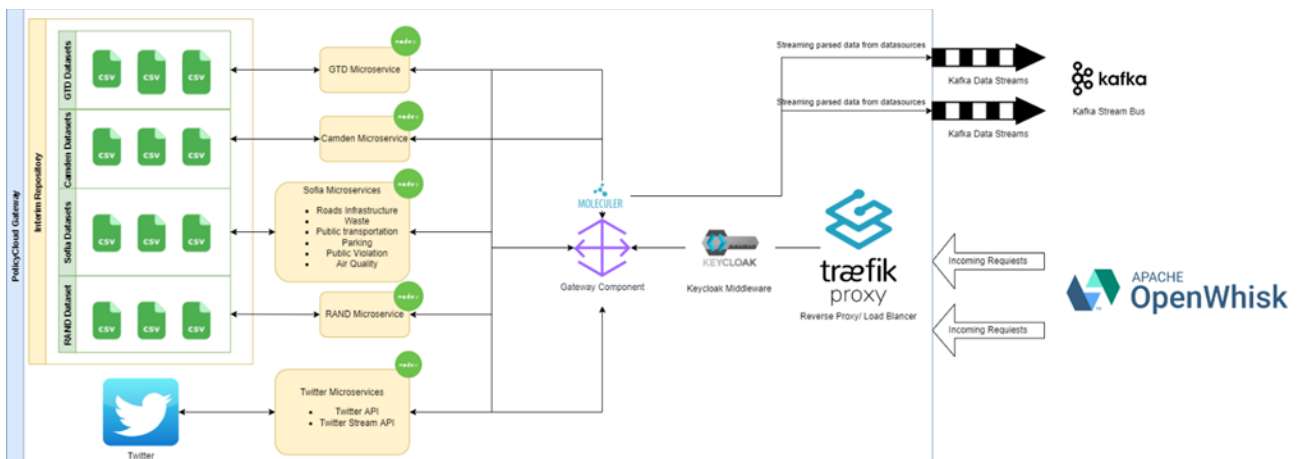


FIGURE 1 - CLOUD GATEWAYS AND APIS ARCHITECTURE AND INTEGRATIONS

## 2.3 Main Components of the Prototype

### 2.3.1 Microservices

In a microservices environment the running instances of services dynamically change location inside networks. In order for the client or services to be able to make requests to a service it must use a service-discovery mechanism. As introduced in D3.4, the Cloud Gateways and APIs component rely on the utilization of a unified set of microservices. To this end, MoleculerJS has been utilized as it provides a built-in module to handle service discovery and registry. The underlying concept of service discovery is the exchange of heartbeats packets between the registry and the available nodes, to list the working services. If a node fails to broadcast a heartbeat is not used to serve requests made for this particular service.

#### 2.3.1.1 GLOBAL TERRORISM DATABASE MICROSERVICE (GTD MICROSERVICE)

The GTD microservice provides a REST application interface following the OpenAPI specification in order to be easier for the end user to discover the capabilities of the microservice. Furthermore, the microservice includes an API documentation page, by using Swagger UI, so that a graphical interface for interacting with the API can be provided. This makes it easier for the developer to explore all available requests and responses are listed including the required parameters, without the need of setting up a client on his own.

#### 2.3.1.2 RAND DATABASE MICROSERVICE (GTD MICROSERVICE)

As of the GTD microservice, also the RAND microservice provides a REST application interface following the OpenAPI specification. Furthermore, this component includes an API documentation page, by using Swagger UI, so that a graphical interface for interacting with the API can be provided. This makes it easier for the developer to explore all available requests and responses are listed including the required parameters, without the need of setting up a client on his own.

#### 2.3.1.3 ARAGON MICROSERVICE

As in the case of the two aforementioned microservices, the Aragon microservice provides a REST application interface following the OpenAPI specification in order to 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. The Aragon microservice integrates with the project's Interim Repository to fetch the needed datasets for this Wine Price monitoring scenario. What is more and in contrary to the other microservices, this specific microservice also offers the ability to the end user to indicate the exact dataset to be fetched by placing the name of the dataset as a parameter in the exposed endpoint. This microservice also includes an integration mechanism with the Interim Repository, in which data providers are responsible for uploading the files that are going to be processed by the Cloud Gateway's microservices, as introduced also in the Prototype Overview section.

#### 2.3.1.4 TWITTER MICROSERVICE

Twitter Microservice is a component of the Cloud Gateway, providing access to Twitter data to the Gateway's clients without the need to directly connect to Twitter API. It has been implemented in NodeJS utilizing the twitter-v2 npm package<sup>7</sup>. Furthermore, Twitter is launching the API v2, a new improved version which promises to provide a better developer experience by giving access to a wide variety of data sources and tools. Twitter assures the quality of its data by applying spam filters, access to all results of a query and not only to a partition of results, user-friendly and simplified JSON objects, shorter URLs and OpenAPI specification to test endpoints and watch for any change.

This specific microservice has two (2) basic functionalities:

- **Searching and filtering tweets:** By utilizing the Search Tweets endpoints [8] of the Twitter API v2, Cloud Gateway's users have access to the most recent tweets. The Twitter Connector sub-component provides a REST application interface following the OpenAPI specification in order to 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. The filters that can be applied include:
  - Keyword search
  - The start and end time parameters to limit tweet results to a specific period
  - Max number of tweets in order to limit the results returned
- **Stream tweets for a specific period:** By utilizing the Filtered Stream endpoints [9], Cloud Gateways component allow users to capture tweets in real-time related to a specific topic for a pre-selected time window. The available parameters for the endpoint are including:
  - Keyword
  - Duration (milliseconds): The duration parameter specifies the duration of the stream capturing process. There is a max-duration limit, in order to ensure gateway's users are not abusing the service and also Twitter's rate limits policy [10].

#### 2.3.1.5 LONDON MICROSERVICE

The London sub-component provides a REST application interface following the OpenAPI specification in order to 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. Furthermore, the component includes an API documentation page, by using Swagger UI, so that a graphical interface for interacting with the API can be provided. This makes it easier for the developer to explore all available requests and responses are listed including the required parameters, without the need of setting up a client on his own.

#### 2.3.1.6 SOFIA MICROSERVICES

The Sofia microservice also provides a REST application interface following the OpenAPI specification in order to be easier for the end user to discover the capabilities of the component and to provide well-structured documentation for each of the internal services.

In contrary to the two previously introduced microservices this incorporates six sub-basic functionalities that are being offered and its one providing a corresponding microservice for its specific scenario of the Sofia use case.

- Roads
- Transport
- Waste
- Parking
- Public order
- Air Quality

All the aforementioned scenarios are based on datasets including signals and complaints from the citizens of Sofia city with regards to the different topics of interest to further support the urban policy making in key areas of the city. In the next iterations and prototype version of this component two more functionalities will be added regarding the different scenarios of the Sofia Use Case as introduced in D6.11 and more specifically to the Violation of Public Order and Air Quality scenarios. Both these scenarios will use data from Sofia Municipality's Contact Centre Call, while the Air Quality scenario will also use data from the air quality measurement stations of the city.

### 2.3.2 Monitoring – Metrics

Monitoring is a very important aspect in developing microservices as it ensures that all exposed APIs are working as expected without errors. Moreover, it also offers the ability to get metrics for API calls to ensure that the provided infrastructures can handle the incoming traffic load or even to detect attacks. For the Cloud Gateways monitoring the swagger-stats [11] tool have been utilized, which is a Node.js library that collects metrics and traces API requests. Metrics are in Prometheus [12] format, so that enables possible future integration with other widely used metrics and alerting systems like Grafana [13].

Some very interesting and important metrics that are provided include:

- CPU and memory utilization
- Error logging and latest error that occurred
- Request tracing and long request tracing with details request details like headers, parameters, and times
- Statistics and summaries, overall payload measurement during periods of time
- Timelines to help you analyze trends of each request and peak periods
- Built-in Telemetry UI with minimum configuration and many settings that provides good user experience

For utilizing swagger-stats for Cloud Gateways, the npm package @slanatech/swagger-stats was installed, and in order to collect metrics it was configured as middleware on every request in the gateway microservice.

## 2.4 Interfaces

### 2.4.1 Monitoring

Through the utilization of the swagger-stats REST API requests and responses in Node.js Microservices and collects statistics per API Operation. This monitoring tool and its interface can be accessed via the below URL <http://90.147.75.215:3000/api/swagger-stats/>

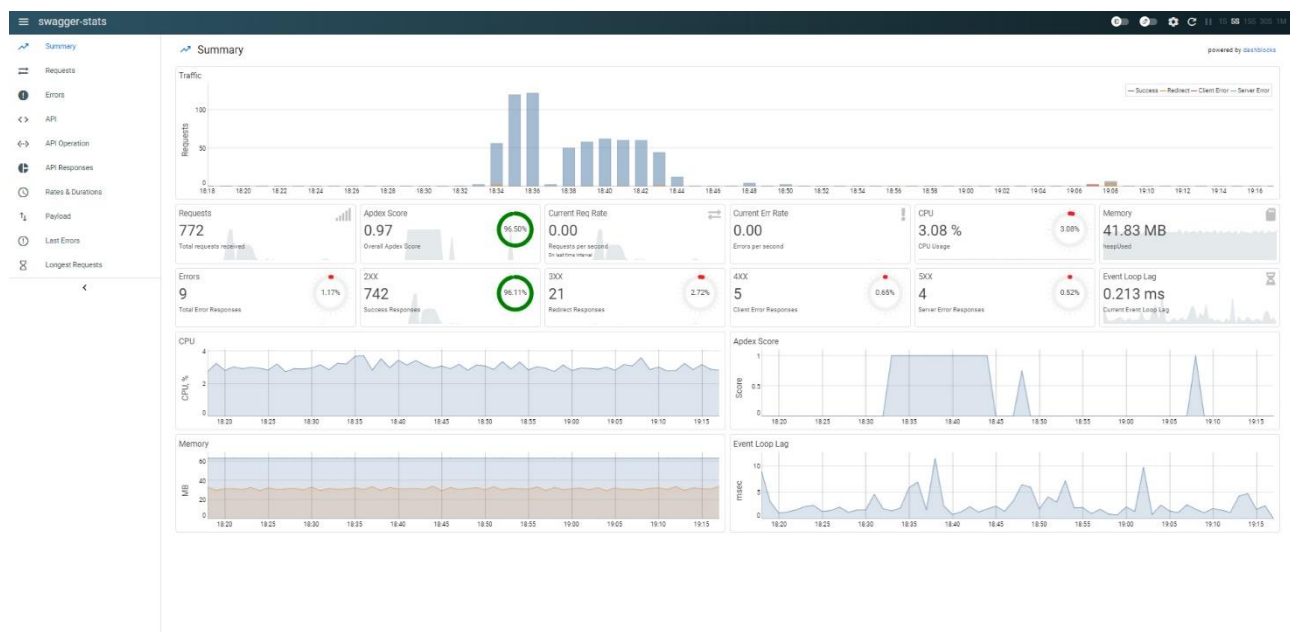


FIGURE 2 – SWAGGER-STATS SUMMARY UI



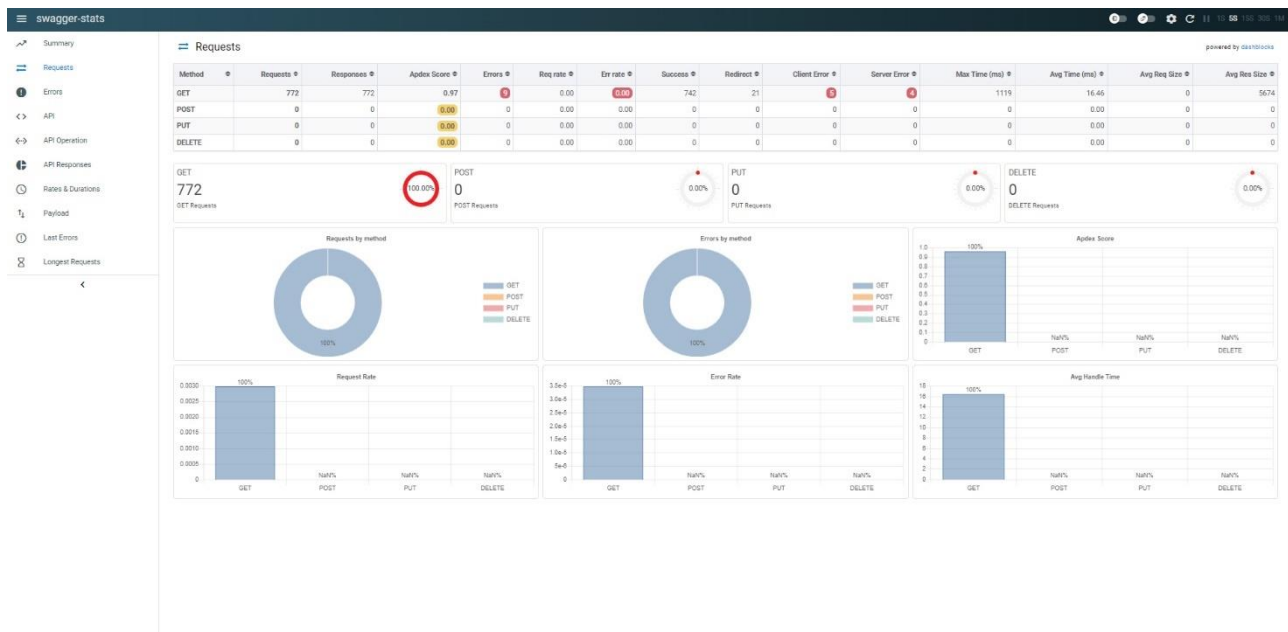


FIGURE 3 – SWAGGER-STATS REQUEST PAGE

## 2.4.2 Microservices

### 2.4.2.1 MOLECULERJS API

MoleculerJS API microservice (<http://90.147.75.215:3000/api/>) is responsible for load balancing, service registry, monitoring and also serves the OpenAPI specification and any other needed page. In the Service Directory of the MoleculerJS are being depicted the nodes that host each microservice. Moreover, this interface offers information about the CPU utilization for each microservice as long as some other information about the nodes that are running, such as the type of service (nodejs), the ID node and the status of each corresponding microservice/node.

Cloud Gateway						
Home Nodes Services						
Node ID	Type	Version	IP	Hostname	Status	CPU
08817dbab3ed-19	nodejs	0.14.14	172.21.0.7	08817dbab3ed	Online	11%
3d048c579d21-19	nodejs	0.14.14	172.21.0.8	3d048c579d21	Online	5%
6eb6d17e4603-18	nodejs	0.14.14	172.21.0.10	6eb6d17e4603	Online	6%
72dd2fe98069-20	nodejs	0.14.14	172.21.0.11	72dd2fe98069	Online	5%
791fe2cd63c4-19	nodejs	0.14.14	172.21.0.9	791fe2cd63c4	Online	7%
eb1fb86b12bc-18	nodejs	0.14.14	172.21.0.6	eb1fb86b12bc	Online	6%

FIGURE 4 – MOLECULERJS – SERVICE DISCOVERY

Furthermore, in the “Services” tab it is being depicted the Service Registry visual graphical interface and the full list of the running microservices along with their endpoints, the node instance that are depending on, the exposed REST API and Status for each one of the available microservices.

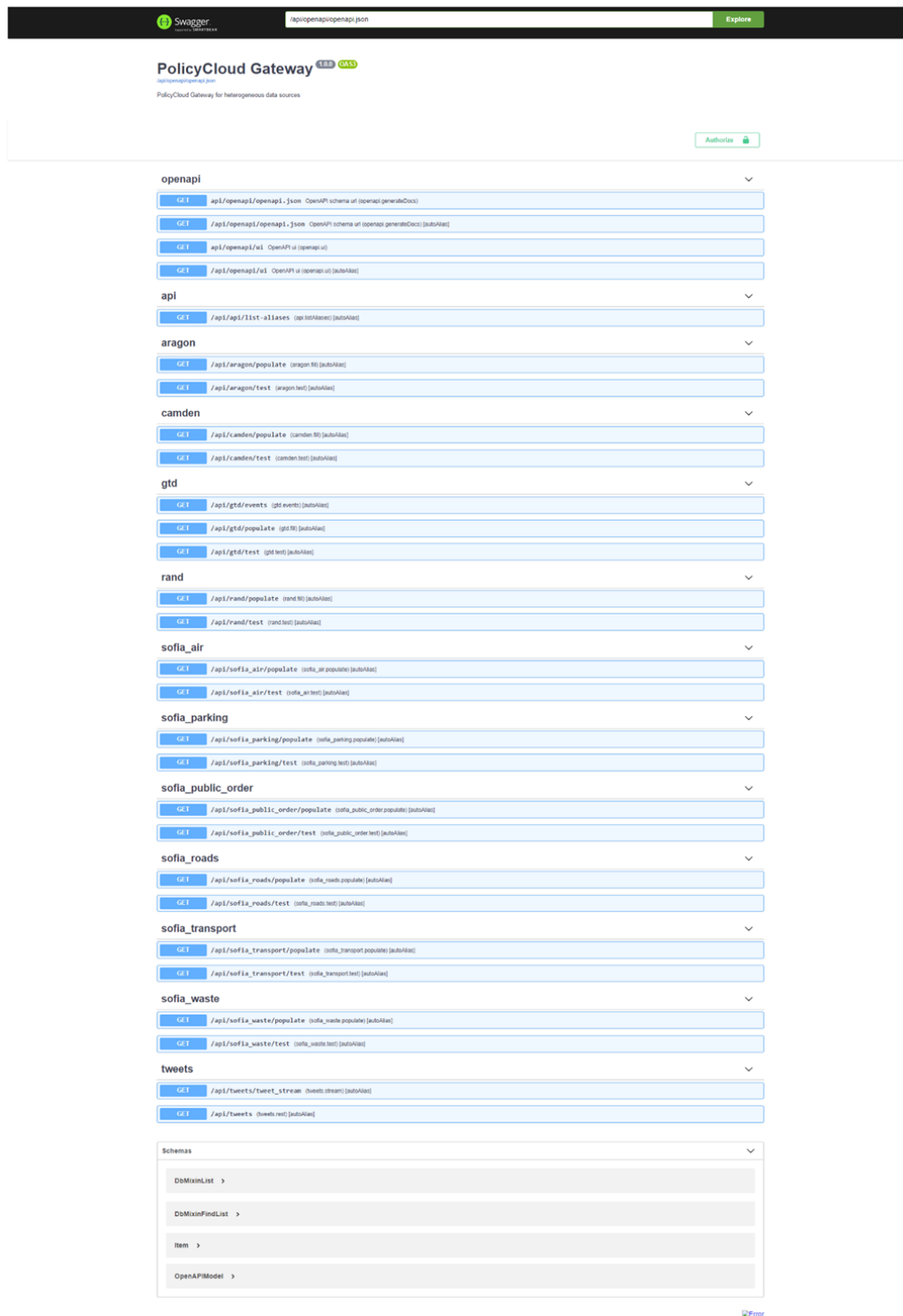


Cloud Gateway				
Home Nodes <u>Services</u>				
Service/Action name	REST	Parameters	Instances	Status
<b>api</b>	api		eb1f80eb12bc-18	Online
api.listAliases	GET api/list-aliases	grouping, withActionSchema		Online
<b>camden</b>	camden		00a17dbab3ed-19	Online
camden.events	GET camden/events	page, pageSize		Online
camden.fill	GET camden/populate	policy_topic		Online
<b>gtd</b>	gtd		6eb6d17e4603-18	Online
gtd.events	GET gtd/events	page, pageSize		Online
gtd.fill	GET gtd/populate	policy_topic		Online
<b>openapi</b>	openapi		eb1f80eb12bc-18	Online
openapi.generateDocs		-		Online
openapi.ui		url		Online
<b>sofia-parking</b>	sofia-parking		791f62cd03c4-19	Online
sofia-parking.parking	GET sofia-parking/parking	page, pageSize		Online
sofia-parking.parkingFill	GET sofia-parking/parkingFill/populate	policy_topic		Online
<b>sofia-roads</b>	sofia-roads		791f62cd03c4-19	Online
sofia-roads.roads	GET sofia-roads/roads	page, pageSize		Online
sofia-roads.roadsFill	GET sofia-roads/roads/populate	policy_topic		Online
<b>sofia-transport</b>	sofia-transport		791f62cd03c4-19	Online
sofia-transport.list	GET sofia-transport/transport	page, pageSize		Online
sofia-transport.add	GET sofia-transport/transport/populate	policy_topic		Online
<b>sofia-waste</b>	sofia-waste		791f62cd03c4-19	Online
sofia-waste.waste	GET sofia-waste/waste	page, pageSize		Online
sofia-waste.roadsFill	GET sofia-waste/waste/populate	policy_topic		Online
<b>storage</b>	storage		3d048c579d21-19	Online
<b>tweets</b>	tweets		72dd2fe98069-20	Online
tweets.rest	GET tweets/	keyword, max_results, topic		Online
tweets.stream	GET tweets/tweet-stream	keyword, duration, topic		Online

FIGURE 5 – MOLECULERJS – SERVICE REGISTRY

#### 2.4.2.2 OPENAPI

An OpenAPI specification page (<http://90.147.75.215:3000/api/openapi/ui#>) is also being provided for describing and documenting all API endpoints. This page contains all the required parameters for the request and gives the ability to the user to perform requests directly to the data sources and to obtain real data in the exact data schema and format.



The screenshot displays the Swagger UI for the PolicyCloud Gateway OpenAPI specification. At the top, the Swagger logo and the URL `api/openapi/openapi.json` are visible, along with an "Explore" button. Below this, the title "PolicyCloud Gateway" is shown with version "0.0.0" and a "GAS" label. A subtitle reads "PolicyCloud Gateway for heterogeneous data sources". An "Authorize" button is located in the top right corner of the main content area.

The main content area lists several API groups, each with a dropdown arrow to its right:

- openapi**
  - GET `/api/openapi/openapi.json` OpenAPI schema of (openapi:generateDoc)
  - GET `/api/openapi/openapi.json` OpenAPI schema of (openapi:generateDoc) (auth:all)
  - GET `/api/openapi/v1` OpenAPI of (openapi:all)
  - GET `/api/openapi/v1` OpenAPI of (openapi:all) (auth:all)
- api**
  - GET `/api/api/list-aliases` (api:listAliases) (auth:all)
- aragon**
  - GET `/api/aragon/populate` (aragon:fill) (auth:all)
  - GET `/api/aragon/test` (aragon:test) (auth:all)
- camden**
  - GET `/api/camden/populate` (camden:fill) (auth:all)
  - GET `/api/camden/test` (camden:test) (auth:all)
- gtd**
  - GET `/api/gtd/events` (gtd:events) (auth:all)
  - GET `/api/gtd/populate` (gtd:fill) (auth:all)
  - GET `/api/gtd/test` (gtd:test) (auth:all)
- rand**
  - GET `/api/rand/populate` (rand:fill) (auth:all)
  - GET `/api/rand/test` (rand:test) (auth:all)
- sofia\_air**
  - GET `/api/sofia_air/populate` (sofia\_air:populate) (auth:all)
  - GET `/api/sofia_air/test` (sofia\_air:test) (auth:all)
- sofia\_parking**
  - GET `/api/sofia_parking/populate` (sofia\_parking:populate) (auth:all)
  - GET `/api/sofia_parking/test` (sofia\_parking:test) (auth:all)
- sofia\_public\_order**
  - GET `/api/sofia_public_order/populate` (sofia\_public\_order:populate) (auth:all)
  - GET `/api/sofia_public_order/test` (sofia\_public\_order:test) (auth:all)
- sofia\_roads**
  - GET `/api/sofia_roads/populate` (sofia\_roads:populate) (auth:all)
  - GET `/api/sofia_roads/test` (sofia\_roads:test) (auth:all)
- sofia\_transport**
  - GET `/api/sofia_transport/populate` (sofia\_transport:populate) (auth:all)
  - GET `/api/sofia_transport/test` (sofia\_transport:test) (auth:all)
- sofia\_waste**
  - GET `/api/sofia_waste/populate` (sofia\_waste:populate) (auth:all)
  - GET `/api/sofia_waste/test` (sofia\_waste:test) (auth:all)
- tweets**
  - GET `/api/tweets/tweet_stream` (tweets:stream) (auth:all)
  - GET `/api/tweets` (tweets:rest) (auth:all)

At the bottom, a "Schemas" section is expanded, showing a list of schema references:

- DBMixinList >
- DBMixinFindList >
- Item >
- OpenAPIModel >

FIGURE 6 - OPENAPI SPECIFICATION PAGE

### 2.4.2.3 MAGGIOLI

For the Maggioli use case, the Global Terrorism Database (GTD) Service is utilized.

gtd

GET
/api/gtd/events
(gtd.events) [autoAlias]

Parameters

Name	Description
page number (query)	1
pageSize number (query)	50

Execute
Clear

Responses

Curl

```
curl -X GET "http://90.147.75.215:3000/api/gtd/events?page=1&pageSize=50" -H "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/gtd/events?page=1&pageSize=50
```

Server response

Code
Details

200

Response body

```
{
  "rows": [
    {
      "_id": "61668d03cd75380013de4269",
      "title": "197404160001",
      "content": {
        "eventId": "197404160001",
        "year": 1974,
        "month": 4,
        "day": 16,
        "approxdate": null,
        "extended": 0,
        "resolution": null,
        "country": "GB",
        "country_txt": "United Kingdom",
        "region": 0,
        "region_txt": "Western Europe",
        "provstate": "Northern Ireland",
        "city": "Newtownhamilton",
        "latitude": "54.192934",
        "longitude": "-6.576851",
        "specificity": 0,
        "vicinity": 0,
        "location": null,
        "summary": null,
        "crit1": 1,
        "crit2": 1,
        "crit3": 1,
        "doubtterr": 0,
        "alternative": null
      }
    }
  ]
}
```

FIGURE 7 - GTD'S SWAGGER OPENAPI INTERFACE

#### 2.4.2.4 ARAGON

For the Aragon use case, the Twitter API and Twitter Streaming API Services are utilized, as also the Aragon microservice that expects as parameters the Kafka topic where the data will be populated as well as the exact filename of the dataset to be fetched

- Utilizing Twitter API

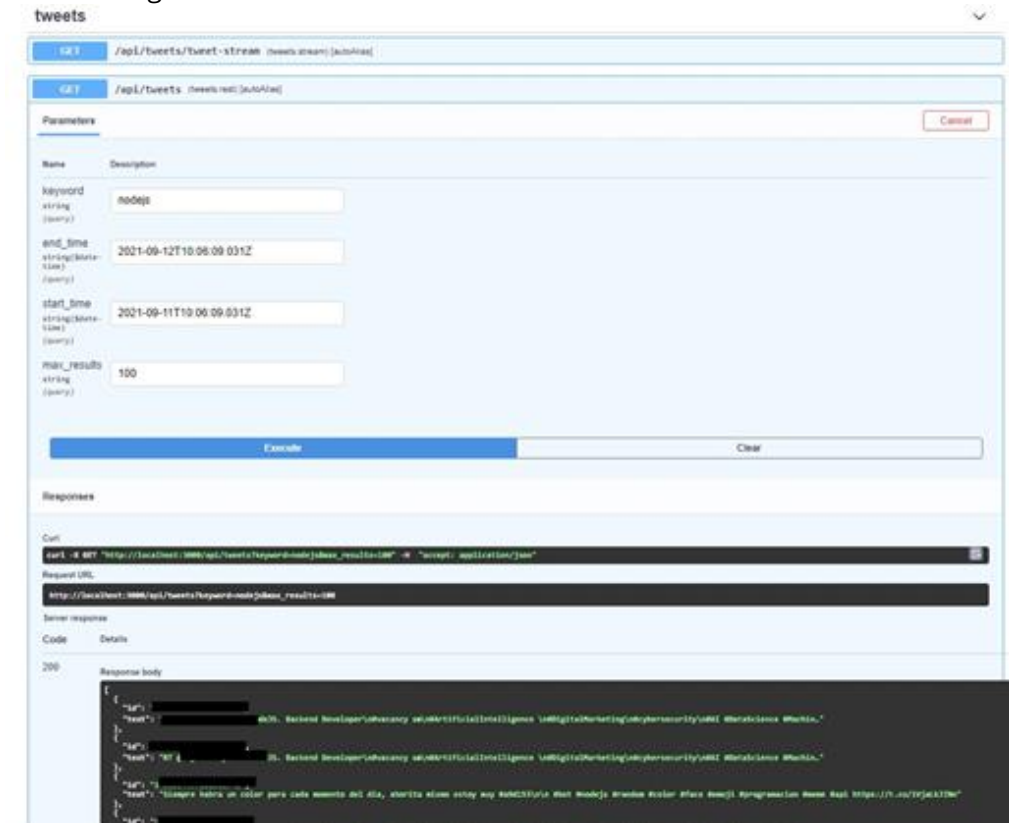


FIGURE 8 – TWEETS FILTERED-SWAGGERUI

- Utilizing Twitter Streaming API

GET

/api/tweets/tweet-stream (tweets.stream) [autoAlias]

Parameters

Cancel

Name	Description
keyword string (query)	<input type="text" value="carinena"/>
duration number (query)	<input type="text" value="10"/>
topic string (query)	<input type="text" value="input_topic"/>

Execute

Clear

Responses

Curl

curl -X GET "http://90.147.75.215:3000/api/tweets/tweet-stream?keyword=carinena&duration=10&topic=input\_topic" -H "accept: application/json"

Request URL

http://90.147.75.215:3000/api/tweets/tweet-stream?keyword=carinena&duration=10&topic=input\_topic

Server response

Code	Details
200	<div>Response body</div> <div>"tweet capture started"</div> <div>Download</div> <div>Response headers</div> <div> content-length: 23  content-type: application/json; charset=utf-8  date: Fri, 15 Oct 2021 15:48:09 GMT  x-request-id: c4f64043-2a48-4010-880f-58b7ce40d568 </div>

Responses

FIGURE 9 – TWITTER STREAMING PROCESS

- Utilizing the Aragon microservice

## aragon

GET
/api/aragon/populate (aragon.fill) [autoAlias]

Parameters

Name	Description
policy_topic string (query)	aragon
filename string (query)	similar_wines2022-05-19.csv

Execute
Clear

Responses

Curl

```
curl -X GET "http://90.147.75.215:3000/api/aragon/populate?policy_topic=aragon&filename=similar_wines2022-05-19.csv" -H "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/aragon/populate?policy_topic=aragon&filename=similar_wines2022-05-19.csv
```

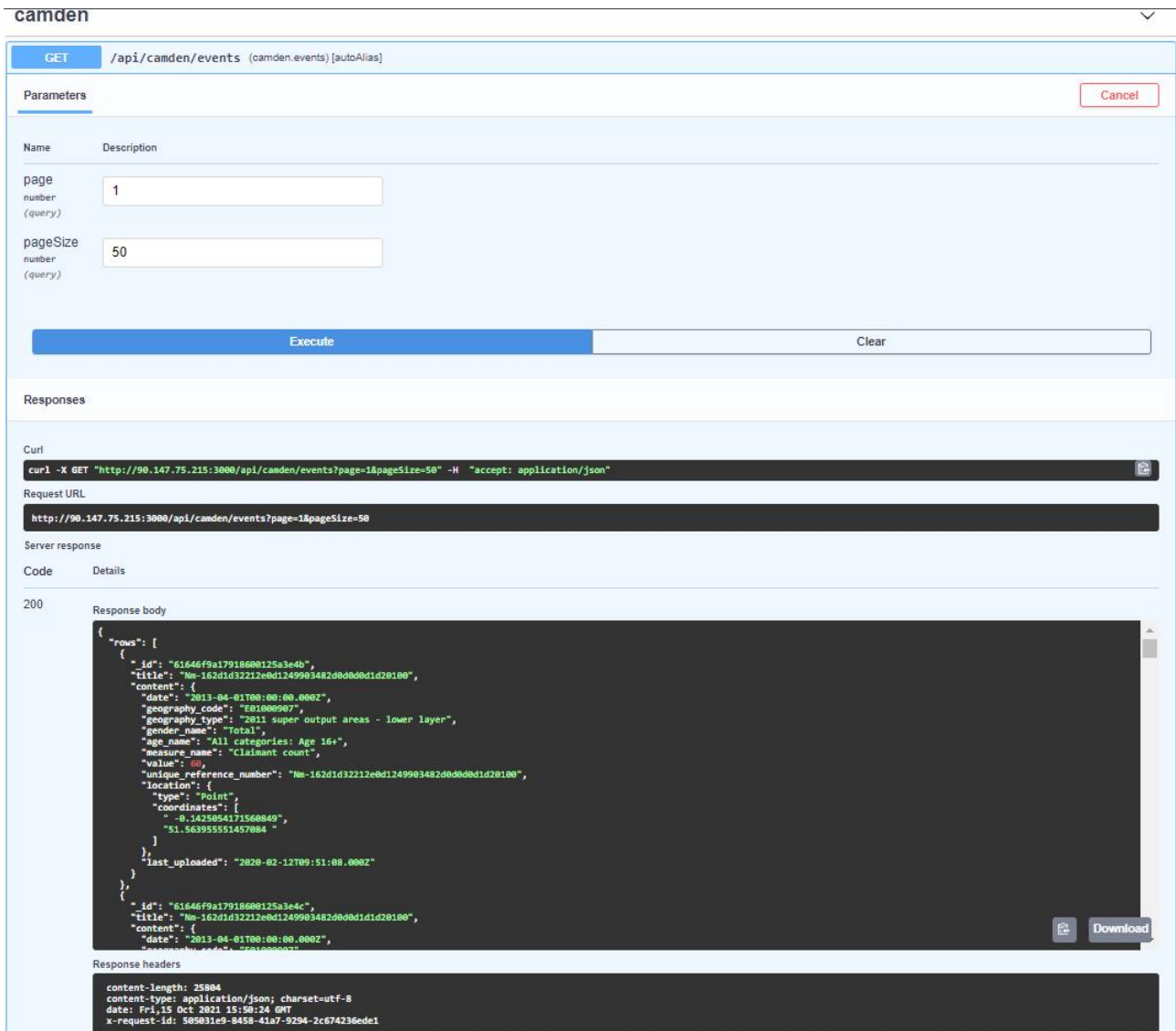
Server response

Code	Details
200	<div>Response body</div> <pre>"OK"</pre> <div>Response headers</div> <pre>content-length: 4 content-type: application/json; charset=utf-8 date: Thu, 29 Sep 2022 16:45:14 GMT x-request-id: 6ec4a65e-c450-4df6-b5e5-c5124b75f841</pre>

FIGURE 10 - ARAGON SWAGGER UI

### 2.4.2.5 LONDON

For the London use case, the Camden Service is utilized.



**camden**

**GET** /api/camden/events (camden.events) [autoAlias]

**Parameters**

Name	Description
page number (query)	1
pageSize number (query)	50

**Execute** **Clear**

**Responses**

**Curl**

```
curl -X GET "http://90.147.75.215:3000/api/camden/events?page=1&pageSize=50" -H "accept: application/json"
```

**Request URL**

```
http://90.147.75.215:3000/api/camden/events?page=1&pageSize=50
```

**Server response**

**Code** **Details**

200

**Response body**

```
{
  "rows": [
    {
      "id": "61646f9a17918600125a3e4b",
      "title": "Nm-162d1d32212e0d1249903482d0d0d1d20100",
      "content": {
        "date": "2013-04-01T00:00:00.000Z",
        "geography_code": "E01000907",
        "geography_type": "2011 super output areas - lower layer",
        "gender_name": "Total",
        "age_name": "All categories: Age 16+",
        "measure_name": "Claimant count",
        "value": 0,
        "unique_reference_number": "Nm-162d1d32212e0d1249903482d0d0d1d20100",
        "location": {
          "type": "Point",
          "coordinates": [
            -0.1423054171560849,
            51.56395551457084
          ]
        },
        "last_uploaded": "2020-02-12T09:51:08.000Z"
      }
    },
    {
      "id": "61646f9a17918600125a3e4c",
      "title": "Nm-162d1d32212e0d1249903482d0d0d1d20100",
      "content": {
        "date": "2013-04-01T00:00:00.000Z",

```

**Response headers**

```
content-length: 25804
content-type: application/json; charset=utf-8
date: Fri, 15 Oct 2021 15:30:24 GMT
x-request-id: 505031e9-8458-41a7-9294-2c674236ede1
```

FIGURE 11 - LONDON SWAGGER UI

### 2.4.2.6 SOFIA

For the Sofia use case, the Roads, Transport, Waste, Parking Services, Violation of Public Order and Air Quality are utilized.

- Roads

**sofia\_roads**

GET /api/sofia\_roads/populate (sofia\_roads.populate) [autoAlias]

Parameters

Name	Description
policy_topic string (query)	sofia_topic

Execute Clear

Responses

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia_roads/populate?policy_topic=sofia_topic" -H "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia_roads/populate?policy_topic=sofia_topic
```

Server response

Code	Details
200	<p>Response body</p> <pre>"OK"</pre> <p>Response headers</p> <pre>content-length: 4 content-type: application/json; charset=utf-8 date: Thu, 29 Sep 2022 16:40:11 GMT x-request-id: 05994f7c-bfc5-43b8-aed1-c073eef3be3c</pre>

FIGURE 12 – SOFIA ROAD SWAGGER UI



- Transport

sofia\_transport

GET /api/sofia\_transport/populate (sofia\_transport.populate) [autoAlias]

Parameters

Name	Description
policy_topic string (query)	<input type="text" value="sofia_topic"/>

Execute Clear

Responses

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia_transport/populate?policy_topic=sofia_topic" -H "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia_transport/populate?policy_topic=sofia_topic
```

Server response

Code	Details
200	<p>Response body</p> <pre>"OK"</pre> <p>Response headers</p> <pre>content-length: 4 content-type: application/json; charset=utf-8 date: Thu, 29 Sep 2022 16:38:54 GMT x-request-id: e5d846a8-e6f2-4619-8cb3-1d37fb233ad0</pre>

Download

FIGURE 13 - SOFIA TRANSPORT SWAGGER UI

- Waste

**sofia\_waste**

GET /api/sofia\_waste/populate (sofia\_waste populate) [autoAlias]

Parameters

Cancel

Name	Description
policy_topic string (query)	sofia_topic

Execute Clear

Responses

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia_waste/populate?policy_topic=sofia_topic" -H "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia_waste/populate?policy_topic=sofia_topic
```

Server response

Code	Details
200	<p>Response body</p> <pre>"OK"</pre> <p>Response headers</p> <pre>content-length: 4 content-type: application/json; charset=utf-8 date: Thu, 29 Sep 2022 16:38:29 GMT x-request-id: 48eeb9b7-2eb9-49ab-80c5-09dc52f6505e</pre>

Download

FIGURE 14 - SOFIA WASTE SWAGGER UI

- Parking

**sofia\_parking**

GET /api/sofia\_parking/populate (sofia\_parking populate) [autoAlias]

Parameters

Cancel

Name	Description
policy_topic string (query)	sofia_topic

Execute Clear

Responses

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia_parking/populate?policy_topic=sofia_topic" -H "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia_parking/populate?policy_topic=sofia_topic
```

Server response

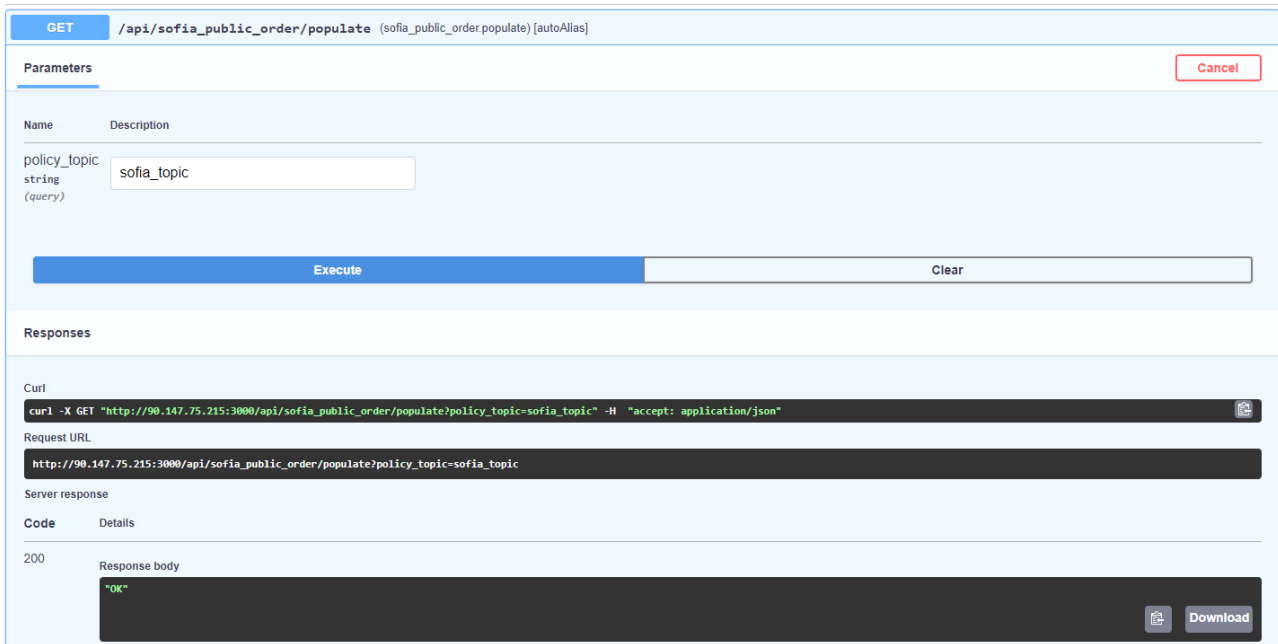
Code	Details
200	<p>Response body</p> <pre>"OK"</pre> <p>Response headers</p> <pre>content-length: 4 content-type: application/json; charset=utf-8 date: Thu, 29 Sep 2022 16:37:56 GMT x-request-id: 9181c9c2-d4df-47fa-8c3b-c8bea78d4c97</pre>

Download

FIGURE 15 - SOFIA PARKING SWAGGER UI

- Violation of Public Order

#### sofia\_public\_order



GET /api/sofia\_public\_order/populate (sofia\_public\_order.populate) [autoAlias]

**Parameters**

Name	Description
policy_topic string (query)	sofia_topic

**Execute** **Clear**

**Responses**

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia_public_order/populate?policy_topic=sofia_topic" -H "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia_public_order/populate?policy_topic=sofia_topic
```

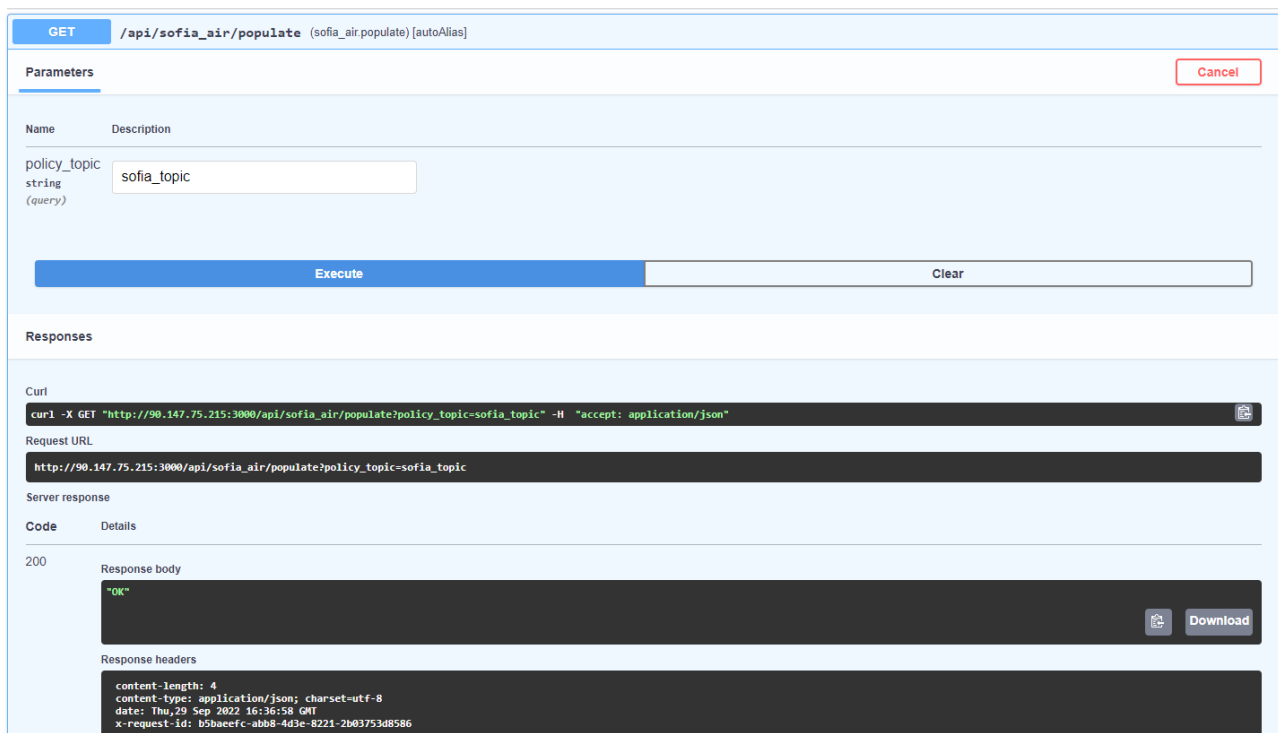
Server response

Code	Details
200	<p>Response body</p> <pre>"OK"</pre> <p><b>Download</b></p>

FIGURE 16 - SOFIA VIOLATION OF PUBLIC ORDER SWAGGER UI

- Air Quality

#### sofia\_air



GET /api/sofia\_air/populate (sofia\_air.populate) [autoAlias]

**Parameters**

Name	Description
policy_topic string (query)	sofia_topic

**Execute** **Clear**

**Responses**

Curl

```
curl -X GET "http://90.147.75.215:3000/api/sofia_air/populate?policy_topic=sofia_topic" -H "accept: application/json"
```

Request URL

```
http://90.147.75.215:3000/api/sofia_air/populate?policy_topic=sofia_topic
```

Server response

Code	Details
200	<p>Response body</p> <pre>"OK"</pre> <p><b>Download</b></p> <p>Response headers</p> <pre>content-length: 4 content-type: application/json; charset=utf-8 date: Thu, 29 Sep 2022 16:36:58 GMT x-request-id: b5baefc-abb8-4d3e-8221-2b03753d8586</pre>

FIGURE 17 – SOFIA AIR QUALITY SWAGGER UI

### 2.4.2.7 RAND

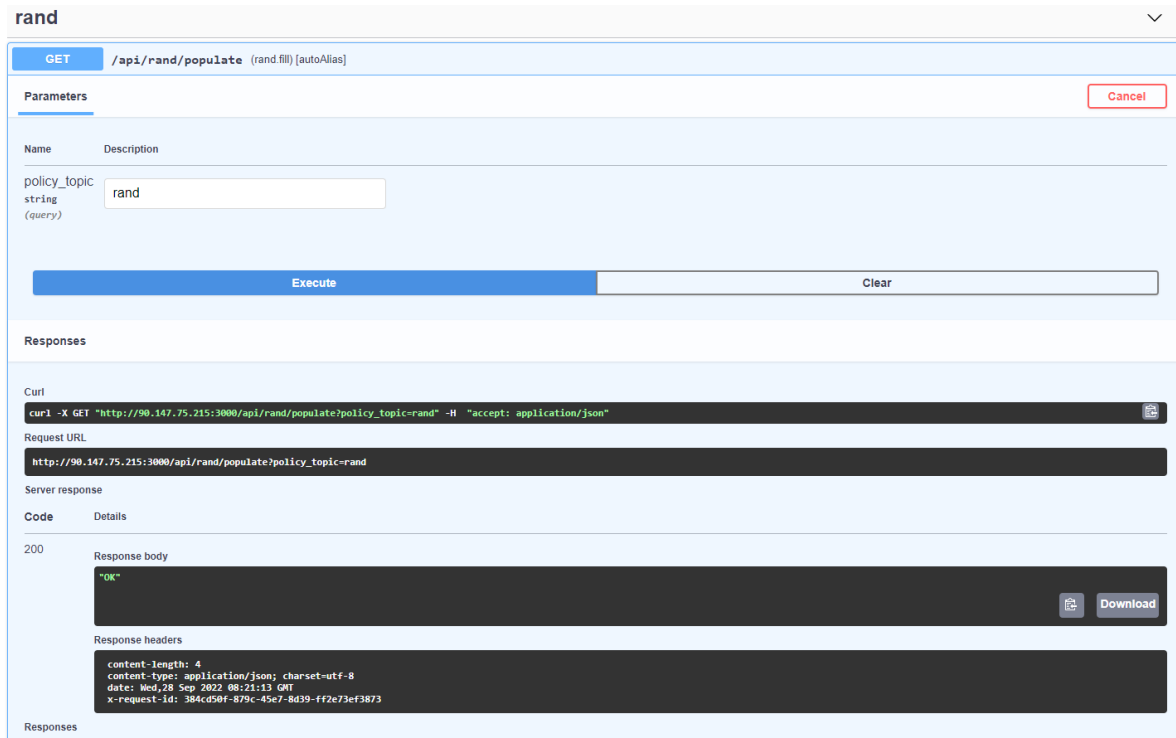


FIGURE 18 - RAND MICROSERVICE SWAGGER UI

## 2.5 Baseline Technologies and Tools

### 2.5.1 MoleculerJS

MoleculerJS is a lightweight Node.js framework oriented in building and managing microservices. It provides many out-of-the-box features that make the development process faster easier. Moreover, this framework facilitated the focus on the business logic and the providing of the required data from external sources to the PolicyCLOUD in order to be further processed and analyzed by other components. Finally, MoleculerJS is a NodeJS framework meaning that most business logic parts can be transferred and mitigated to other frameworks and distributed systems without major changes and time drawbacks.

### 2.5.2 Traefik

The Cloud Gateways component utilizes Traefik, a popular HTTP reverse proxy and a load balancer software. The latter requires minimum effort for integration with infrastructure components like Docker and Kubernetes. Traefik instead of requiring manual route configuration for each service component, bundles to the registry service or orchestrator API and generates all routes automatically so this services to be available for public and ready to use. Continuously updating its configurations can be really helpful feature especially in a microservices environment that changes in each microservice are very common issue and component restarts are required [54]. Traefik also provides a UI to monitor metrics and toggle configurations.

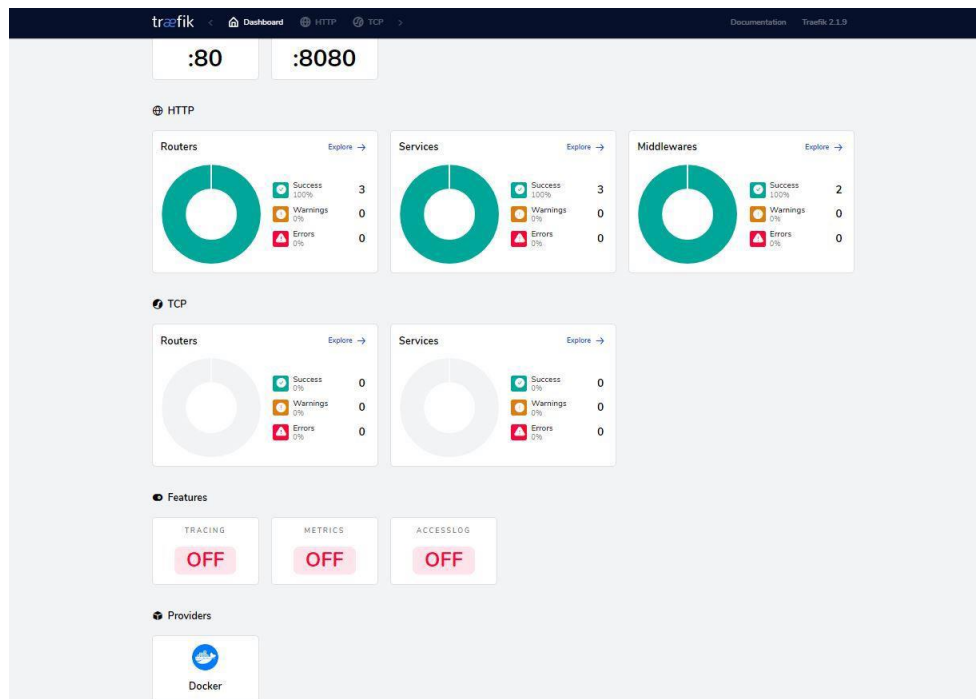


FIGURE 19 – TRAEFIK WEB UI

### 2.5.3 Docker

Docker is a virtualization platform that gives the ability to run applications in isolated environments called containers. Each container contains all necessary software to run an application and is very size efficient. Multiple containers can be running simultaneously on the same host. Docker has been selected in order to provide the Cloud Gateways component and its subcomponents and microservices as docker images that can be initialized as a virtualized container inside a Kubernetes pod. To this end, the development progress can dramatically speed up and the unified component can be deployed faster and safer in production. One other advantage of using containers is the ability to share between developers exactly the same developer environment, thus Cloud Gateways component can be offered to every stakeholder in the same environment.

### 2.5.4 File Parsers

Parsing data from files of different format has always been a challenge for data science. Different formats, different delimiters and compressions systems can lead to the complexity of the task. Another serious problem is the parsing of really large data files. In this case common parsing techniques are not working because it is not possible to fit the entire file in the heap memory and the resources are limited.

The Cloud Gateway provides two (2) different file reading microservices. This scenario also includes an integration mechanism with the Interim Repository, in which data providers are responsible for uploading the files that are going to be processed by the Cloud Gateway's microservices.

#### 2.5.4.1 CSV PAPA.PARSE

For parsing CSV file the `papa.parse` JS package is being utilized since it provides parsing capabilities over huge files without loading the whole file into memory and thus causing memory leaks and timeouts in the requests. The parsing process and the streaming to the Kafka message broker is an asynchronous process and does not affect the performance or causes any timeout.

#### 2.5.4.2 XLSX FILE PARSER

Along with the CSV datasets, also XLSX datasets are able to be parsed by the microservices provided in the Cloud Gateways and APIs component. These microservices are responsible for parsing the dataset files in XLSX format in order to extract data. Similarly, to the GTD microservice, the parsing procedure is a scheduled job that run in the background and while parsing the data are formatted in Kafka suitable format and sent to the Kafka's water-topic in order to be later sent and stored in the storage component. The service is implemented in NodeJS and for the CSV parsing the `@SheetJS/sheetjs` npm package is utilized [14].

## 2.6 Source Code

### 2.6.1 Code Overview and Availability

The code and instructions to build the project locally are available at the repository located at <https://registry.grid.ece.ntua.gr/george/cloudgatewayv2>. Access is restricted to members of the project consortium.

#### Usage

Start the project with `npm run dev` command.

After starting, open the `http://localhost:3000/` URL in your browser.

On the welcome page you can test the generated services via API Gateway and check the nodes & services.

In the terminal, try the following commands:

### 2.6.2 NPM scripts

- `npm run dev`: Start development mode (load all services locally with hot-reload & REPL)
- `npm run start`: Start production mode (set `SERVICES` env variable to load certain services)
- `npm run cli`: Start a CLI and connect to production. Don't forget to set production namespace with `--ns` argument in script
- `npm run lint`: Run ESLint
- `npm run ci`: Run continuous test mode with watching
- `npm test`: Run tests and generate coverage report
- `npm run dc:up`: Start the stack with Docker Compose
- `npm run dc:down`: Stop the stack with Docker Compose

```
Commands to setup VPS sudo apt update sudo apt install apt-transport-https ca-certificates curl
software-properties-common curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add - sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic
stable" sudo apt update apt-cache policy docker-ce sudo apt install docker-ce sudo systemctl status
doker docker sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose sudo chmod +x /usr/local/bin/
docker-compose docker-compose --version sudo apt install git git clone docker-compose up
```

## 2.7 Deployment Status

Currently the Gateway has been deployed on a public VPS server in the project's cloud infrastructure that has been provided and is supported by RECAS-BARI and EGI. However, in the project's Gitlab repository are being provided detailed instructions and guidelines that facilitate the deployment of the Cloud Gateways and APIs component and its microservices in any environment.

## 3 Incentives Management

The Incentive Management (IM) software prototype presented here is one more step towards the realization of the final objective of the Incentive Management task, which is based, as we described in D3.1 [1], on the idea of including citizens as participants in the policy development and management process, having the policy maker as main actor. In this way, we present below the prototype details of the Incentive Management Tool, that aims to help policy makers to achieve this objective.

### 3.1 Updates since D3.5

The improvements of this component were stopped during the last period of the project, after the recommendation of the reviewers and thus the most important updates reported in this deliverable since **D3.5** are related to the final integration of the component in the PolicyCLOUD infrastructure. This integration involves:

- the integration of the IM Front End inside the Policy Development Toolkit (PDT) component
- and the deployment of the IM Back End into the Cloud infrastructure.

In this way, the component concludes its development phase, unless if it's necessary to implement some minimal variation essential for its use by the use case that possibly makes use of it.

In addition, all the development related to the **Reports** section, mentioned in deliverable **D3.5**, is disregarded, and therefore removed from the final version of the component.

### 3.2 Prototype Overview side

The work started in D3.2 [4] resulted in a set of functionalities obtained from the use cases, that the Incentive Management Tool needs to provide. The functionalities corresponding to the "Incentives Management" capability were defined in D3.2:

- "Declaration of Incentives: The policy maker should be able to register new incentives with a set of attributes", where the Actions Type attribute takes special relevance.
- "Crowdsourced Data Ingestion and Summary: a summary statistic from the crowdsourced data", where crowdsourced task reports, posted in the system, support the statistics which allow the policy maker the evaluation of applied incentives

The Incentive Management Tool has been designed to provide these functionalities to the policy maker. For this first prototype, the Incentive Management Tool component, focusing by now on the first functionality, declaration of incentives, provides:

- **Front End:** The gateway to the component for the policy maker. The Incentive Management Front End will be integrated into the PDT and will be accessed by policy makers through the PolicyCloud platform, although for this first prototype, this first integration is not done yet. To make this



integration possible, the Front End component has been developed in Angular, to be in line with the PDT, and is composed by:

- ***incentives-management-home* component**: This component, composed by its "*incentives-management-home.ts*", "*incentives-management-home.html*", *incentives-management-home.spec.ts* and *incentives-management-home.css* files, provides a common "frame" for the following components.
- ***incentive-management.services.ts***: Is responsible for the communication with the Back End component.
- ***incentives* component**: Represents the incentives given to citizens, to get their participation. At the Incentive Management Tool, this component is responsible for the management of the incentives.
- ***motives* component**: Represents the motives that encourage citizens to participate. At the Incentive Management Tool, this component is responsible for the management of the motives.
- ***actions* component**: Represents the actions needed to launch one specific incentive. At the Incentive Management Tool, this component is responsible for the management of the actions.
- ***producerstypology* component**: Represents the best profile, or best group of people, to perform a selected action. At the Incentive Management Tool, this component is responsible for the management of the producerstypology.

As the communication between Front End and Back End is available through the PDT integration, the policy makers can management of Incentives, Motives, Actions and ProducersTypology (create, edit, and delete) can be done.

- **Back End**: This component has been developed in Python3, it is composed by five controllers, related to the five main elements of the Front End:
  - ***incentives\_controller.py***: This controller contains all the functions related with the incentives management actions.
  - ***actions\_controller.py***: This controller contains all the functions related with the actions management.
  - ***motives\_controller.py***: This controller contains all the functions related with the motives management.
  - ***producerstypology\_controller.py***: This controller contains all the functions related with the producerstypology management.

## 3.3 Main Components of the Prototype

As commented in earlier sections, the main components of this first prototype are:

- **Front End component:** policy makers can access each of the Incentive Management Tool options from the component's initial page, or using the top left menu list:

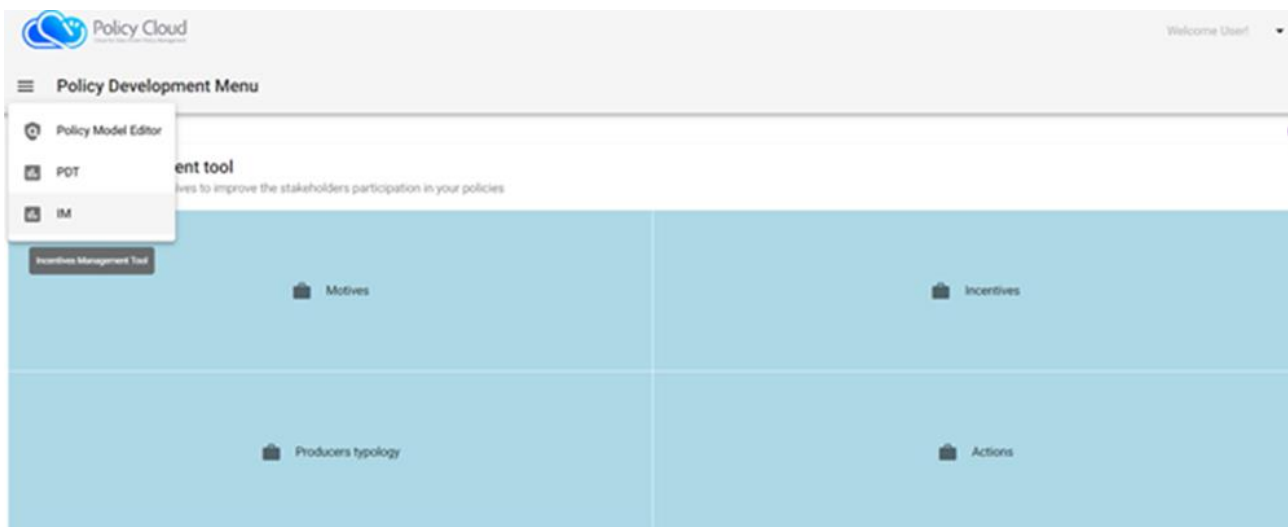


FIGURE 20 - ACCESS TO INCENTIVE MANAGEMENT TOOL OPTIONS

Once selected one option, policy makers will access the Incentives/Motives/Actions/Producers Typology page. As the development was stopped, the Report option has been removed. Policy makers will then see, a list of the incentives they have created so far, if there's any. If no incentive has been created yet, as in the image bellow, two buttons will appear: Add, that allows the creation of a new Incentive, and Back, to return to the initial page:

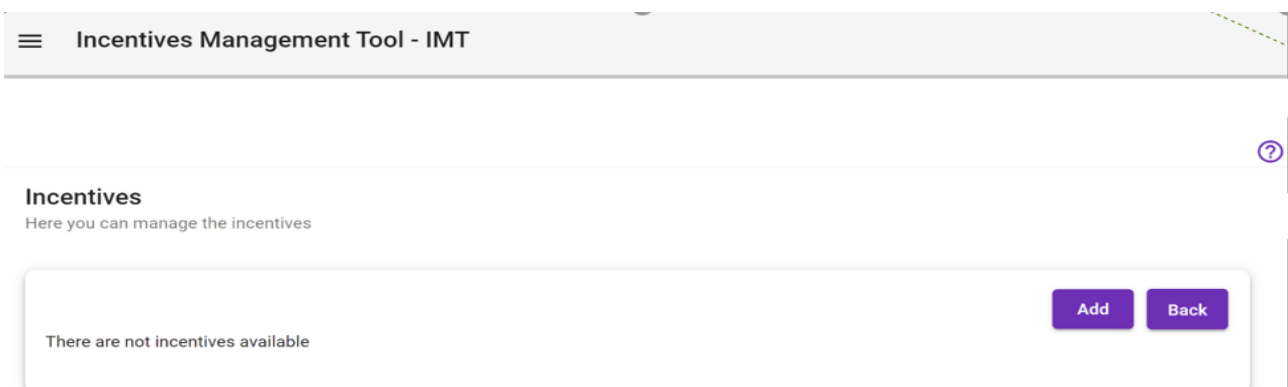


FIGURE 21 - INITIAL PAGE FOR INCENTIVE'S OPTION

In the case of the previous figure, if there is any incentive created a list with all incentives created would appear, with a new button at the bottom right: Remove. This new button allows to remove/delete selected Incentives.

If the policy maker wants to create a new incentive, the button Add, must be pushed, and a pop-up will appear, where the data needed for this creation will appear. For this first approach, the name of the new incentive and if it is a suggested one, are the needed inputs. The policy maker can select a motive from a combo box:



FIGURE 22 - FORM TO CREATE A NEW INCENTIVE

The same happens for Motives and Actions options, although for the creation of an action, the information needed, is larger than the ones for Incentives and Motives:

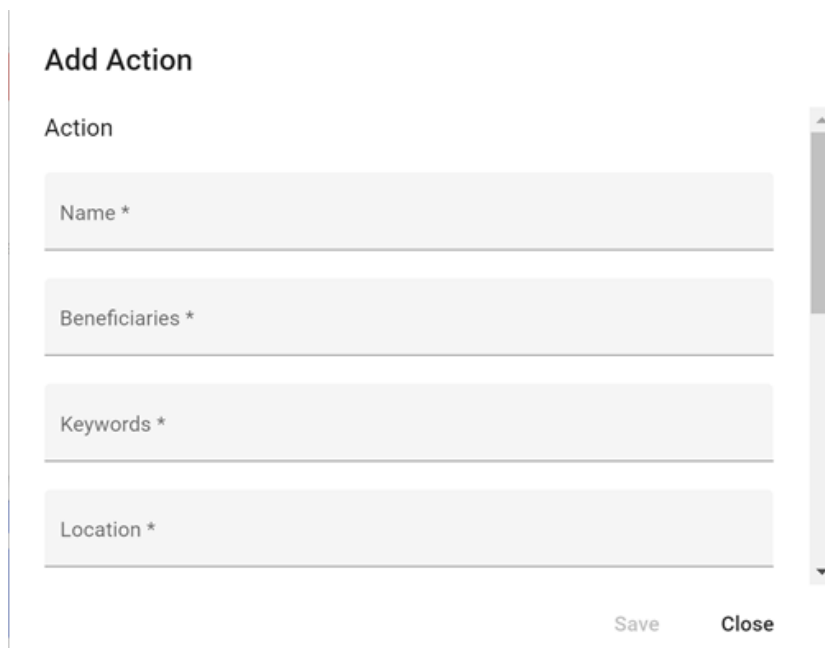


FIGURE 23 - FORM TO CREATE A NEW ACTION

In all pages shown to the policy maker, except in the creation forms, it can be seen a help icon ( ? ) in the top right of the page. Clicking it, policy makers can get some information about the actions they can

perform at that page. Also, trying to help policy makers in the process of incentive creations, many fields have a tooltip that gives a small definition of them.

- **Back End component:** Aims to give all the support to the Front End operations, including the communication with the Storage component. The Incentive Management Back End component is independent from the PDT and is responsible to process all the requests done by the Front End, to store information in the storage component, and to retrieve information from the Storage component. All these operations are done through the functions defined in previous section, provided to the Front End as APIs that can be called.
- **The storage component** for the Incentive Managed component was integrated into the storage component of the PolicyCLOUD platform.

For the Incentive Management Storage component, considers the data model detailed in D3.4 [2], and has 6 tables: action, incentive, motive, policy, report and producertyology.

In the deliverable D3.4 already mentioned, there was also a component initially thought as “Access to policies” component, which would “allows policy makers to see results from their policies, providing them with insights for the creation of new incentives”. This component finally will not be developed since that development tasks will not be continued.

## 3.4 Interfaces

For the Front End component, the Interface, as explained in previous sections, will be integrated in the PDT, and the first page can be seen in Figure 20. As mentioned before, its main components are:

- *incentive-management.services.ts*: Composed by:
  - `getIncentivesList(creatorId:number = null)`: Gets list of Incentives from the server.
  - `getIncentiveDataById(incentiveId: number = null)`:  
Get Incentive by incentiveld from the server.
  - `updateIncentiveDataById(incentiveId: number = null, dataModel: any)`:  
Update Incentive by incentiveld from the server.
  - `addNewIncentive(dataModel: any)`: Create a new Incentive.
  - `deleteIncentive(incentiveId: number )`: Delete an incentive by incentiveld.
  - `getMotivesList()`: `Observable<Incentive[]>`: Get list of Motives from the server.
  - `getMotiveDataById(motiveId: number = null)`:  
Get Motive by motiveld from the server.
  - `updateMotiveDataById(motiveId: number = null, dataModel: any): Observable<any[]>`:  
Update Motive by motiveld from the server.
  - `addNewMotive(dataModel: any): Observable<any[]>`: Create a new Motive.
  - `deleteMotive(motiveId: number )`: `Observable<{}>`: Delete a motive by motiveld.
  - `getActionsList()`: `Observable<Action[]>`: Get list of Actions from the server.

- `getActionDataById(actionId: number = null):`  
Get Action by actionId from the server.
- `updateActionDataById(actionId: number = null, dataModel: any): Observable<any[]>:` Update Action by actionId from the server.
- `addNewAction(dataModel: any): Observable<any[]>:` Create a new Action.
- `deleteAction(actionId: number): Observable<{}>:` Delete an action by actionId
- `getProducerTypologiesList(): Observable<Action[]>:` Get list of Producer typologies from the server.
- `getProducerTypologyDataById(producerTypologyId: number = null): Observable<Action>:` Get Producer Typology by producerTypologyId from the server.
- `updateProducerTypologyDataById(producerTypologyId: number = null, dataModel: any): Observable<any[]>:` Update Producer typology by producerTypologyId from the server.
- `addNewProducerTypology(dataModel: any): Observable<any[]>:` Create a new Producer Typology.
- `deleteProducerTypology(producerTypologyId: number): Observable<{}>:`  
Delete a producer typology by producerTypologyId.
- **incentives component:** Composed by:
  - `addIncentive():` Open a new window where policy makers will introduce the data needed to create an incentive.
  - `editIncentive(id, element):` Make changes in an already created incentive.
  - `deleteIncentive(id, element):` Delete a created incentive.
  - `saveIncentiveForm():` Creates a new incentive with the data introduced for the policy maker.
- **motives component:** Composed by:
  - `addMotive(); editMotive(id, element); deleteMotive(id, element);` and `saveMotiveForm()` with same functionalities as incentives, but with motives type.
- **actions component:** Composed by:
  - `addAction(); editAction(id, element); deleteAction(id, element);` and `saveActionForm()` with same functionalities, but with actions type.
- **producerstypology component:** Composed by:
  - `addProducerTypology(); editPT(id, element); deletePT(id, element); savePTForm()` with same functionalities, but with actions type

The Back End interface that is provided by a Swagger GUI (<https://swagger.io/>) can be seen in the following image:

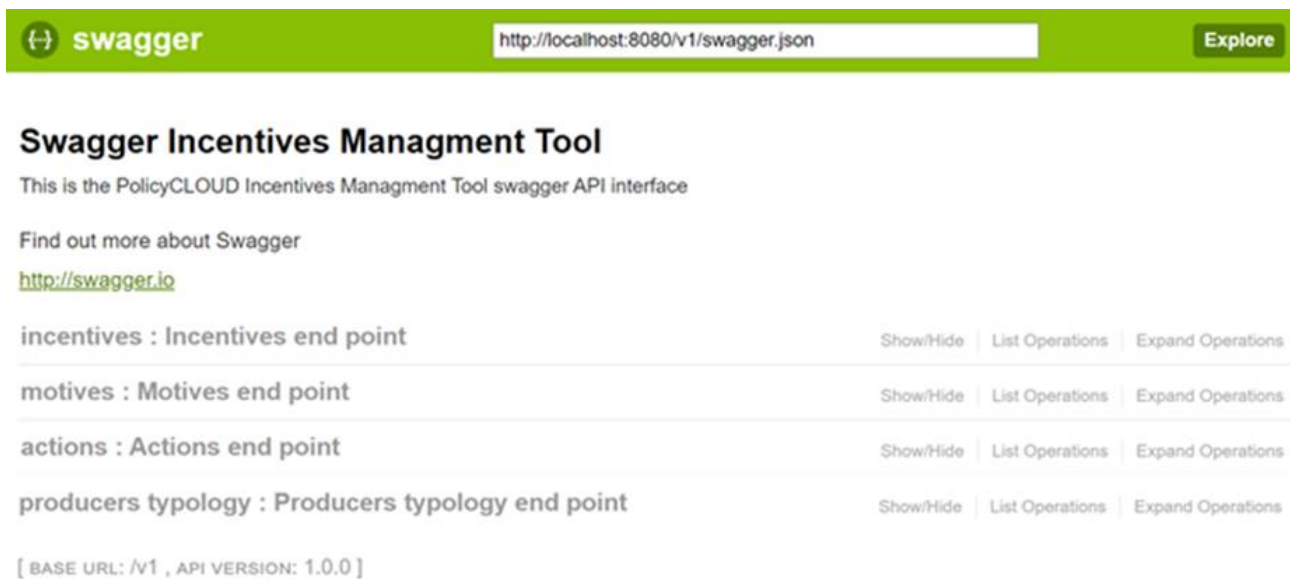


FIGURE 24 - SWAGGER FOR INCENTIVE MANAGEMENT BACK END COMPONENT

Where all APIs provided by the component can be seen. This APIs provide all the functions described in previous section:

- **incentives\_controller.py:** Composed by:
  - `def add_incentive(body):` Add a new incentive to the data base.
  - `def delete_incentive(incentiveId):` Deletes an incentive.
  - `def find_incentives(creatorId=None):` Get all incentives.
  - `def get_incentive_by_id(incentiveId):` Find incentive by ID.
  - `def update_incentive(incentiveId, body):` Updates an incentive by id.
- **actions\_controller.py:** Composed by:
  - `def add_action(body):` Add a new action to the data base.
  - `def delete_action(actionId):` Deletes an action.
  - `def find_actions():` Get all actions.
  - `def get_action_by_id(actionId):` Find action by id.
  - `def update_action(actionId, body):` Updates an action by id.
- **motives\_controller.py:** Composed by:
  - `def add_motive(body):` Add a new motive to the data base.
  - `def delete_motive(motiveId):` Deletes a motive.
  - `def find_motives():` Get all motives.

- `def get_motive_by_id(motiveId):` Find motives by id.
- `def update_motive(motiveId, body):` Updates an motive by id.
- **producerstypology\_controller.py:** Composed by:
  - `def add_producer_typology(body):` Add a new producer typology to the data base.
  - `def producers_typology():` Get all producer typologies.
  - `def get_producer_typology_by_id(producerTypologyId):` Get producer typology by id.
  - `def update_producer_typology(producerTypologyId, body):` Updates a producer typology in the database with form data.
  - `def delete_producer_typology(producerTypologyId):` Deletes a producer typology.

## 3.5 Baseline Technologies and Tools

The technologies used for the Incentive Management Tool can be divided in:

- For the Front End component: the technology used is Angular (<https://angular.io/>), having in mind the integration with the PDT.
- For the Back End component: the technology chosen is Python3 (<https://www.python.org/>).
- For the Storage component: the technology used is MySQL (<https://www.mysql.com/>), having in mind the integration with the PolicyCLOUD storage, LXC, which is based in MySQL.

## 3.6 Source Code

### Code Overview and Availability

The code for the two different Incentive Management Tool components can be found at:

**Front End component:** <https://registry.grid.ece.ntua.gr/kostas/pdt>

**Back End component:** [https://registry.grid.ece.ntua.gr/ana.pontual/incentivesmanagement\\_be](https://registry.grid.ece.ntua.gr/ana.pontual/incentivesmanagement_be)

Access is restricted to members of the project consortium.

## 3.7 Deployment Status

The functionalities described for the Incentive Management Tool have been deployed restricted to the project environment, that is, private. However, in the git repository are provided instructions and files that allow components deployment whenever and wherever needed.

For the Front End component, as it is integrated inside the PDT, deployments will take place together and in the same way as the PDT.

For the Back End component, the instructions of how to deploy it are in the Readme file in the Git repository of the project. This component is deployed using Docker containers into the PolicyCLOUD infrastructure.

The Storage component is deployed inside PolicyCLOUD storage, throw LXC.



## 4 Data Governance Model and Privacy Enforcement mechanism

### 4.1 Updates since D3.5

Since D3.5, the Privacy Enforcement mechanism has been also integrated with the PolicyCLOUD marketplace, in order to authenticate access to it. The same Keycloak realm that the PDT and Gateways use has also been integrated to the marketplace in order for those components to share a common user base and further ease the use of all the PolicyCLOUD services by the users.

Furthermore, a XACML editor has been provided in order to provide an easier way to create, edit and customize access policies and avoid the more complex way of defining policies in the XML format, which is not so human readable.

Finally, custom access policies have been developed to authenticate the gateway and interim repository endpoints, based on the URL of the request and thus based on the pilot needs. This feature further enhances the security of the platform and also allows on the fine-tuned fly adjustments on access to different PolicyCLOUD endpoints.

### 4.2 Prototype Overview

The prototype's purpose is to demonstrate the ability of the ABAC Engine to intercept a request, obtain the attributes of the requestor and evaluate whether the request should be permitted, based on those attributes and the enforced Policies. The two main components responsible for this functionality are the ABAC Server and the ABAC Client Filter, while the prototype also contains a simple Web Client for testing purposes.

Keycloak [15] is selected as the Attribute Provider and User Authenticator for this version of the prototype. In the upcoming months, we will examine the integration of our solution as part of the integrated PolicyCLOUD platform and alternate authentication solutions that could be used.

The Data model has been evolved through continuous discussion with the pilots in order to fit their needs and requirements and has been refined in each version with all the necessary changes to follow the maturity of the project.

## 4.3 Main Components of the Prototype

### 4.3.1 ABAC Server

The ABAC server is the backbone of the ABAC Authorization Engine and is responsible for evaluating the access requests authorization. It communicates with the ABAC Client to retrieve these requests and responds with a Permit message based on the Policies currently applied and the attributes of the requestor. To set up the ABAC Server locally, Maven is required for an initial step

- `mvn clean install`

For the first installation there would be no keystore and truststore files present so they should be created for during the initialization by executing:

- `keytool -genkey -alias YOUR_ALIAS`
- `keytool -list -v -storetype pkcs12 -keystore pdp-server-keystore.p12`
- `keytool -export -storetype PKCS12 -keystore pdp-server-keystore.p12 -storepass <YOUR_KEYSTORE_PASSWORD> -alias pdp-server -file pdp-server.crt`
- `keytool -printcert -file pdp-server.crt`
- `keytool -import -alias pdp-server -file pdp-server.crt -storetype PKCS12 -keystore pdp-server-truststore.p12 -storepass<YOUR_TRUST_STORE_PASSWORD>`
- `keytool -list -v -storetype pkcs12 -keystore pdp-server-truststore.p12`

When trying to run the server, the Configuration directory must be selected and exported via:

- `export CONFIG_DIR=src/main/resources/config`

Finally, the jar produced by the mvn clean install is executed via:

- `java -jar target/abac-authorization-server.jar`

### 4.3.2 KeyCloak

A dockerised version of Keycloak is used as the Attribute Provider and User Authenticator. The image selected is `jboss/keycloak` and the admin username and password, as well as the ports of the docker container are selected in the sample `docker-compose.yml` provided below:

```
version: '3'
services:
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:4.7
    environment:
      PMA_PORT: 3306
      PMA_HOST: database
    depends_on:
      - database
    ports:
      - 8010:80

  database:
    image: mysql:5.5
    environment:
      MYSQL_ROOT_PASSWORD: "!r00t!"
      MYSQL_DATABASE: store
    ports:
      - 3306:3306

  keycloak:
    image: jboss/keycloak:9.0.2
    environment:
      KEYCLOAK_USER: admin
      KEYCLOAK_PASSWORD: admin
    ports:
      - 8180:8080
```

FIGURE 25 - DOCKER-COMPOSE.YAML FILE

Along with Keycloak, a simple mysql database and mphpmyadmin are installed with basic settings. The Keycloak Admin Console can be accessed by visiting localhost:8180 and providing the login credentials specified in Figure 25. The necessary steps for setting up Keycloak to act as a User Authentication provider involve firstly creating the relevant realm. A Keycloak realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm and realms are isolated from one another and can only manage and authenticate the users that they control. It is important that the name of the realm matches the one used in the application.yml of the service we are trying to secure.

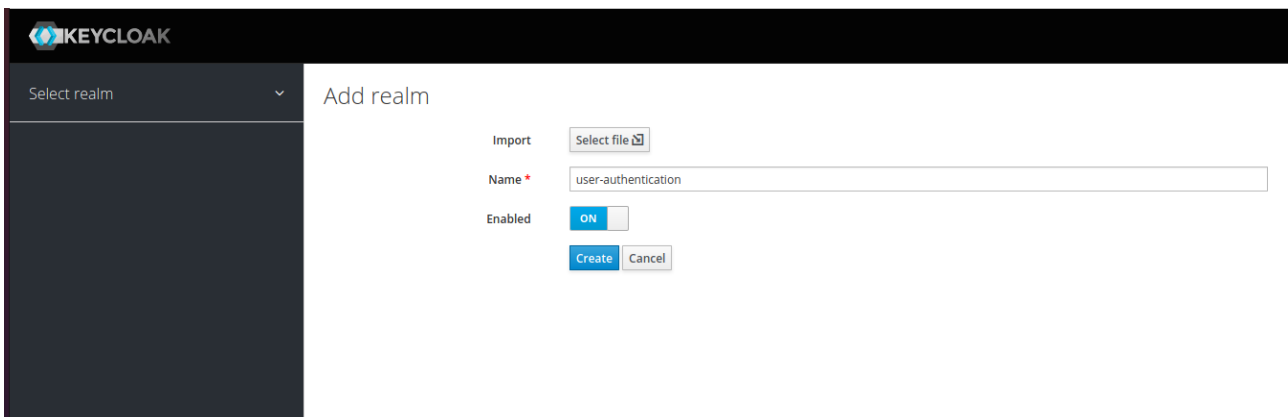
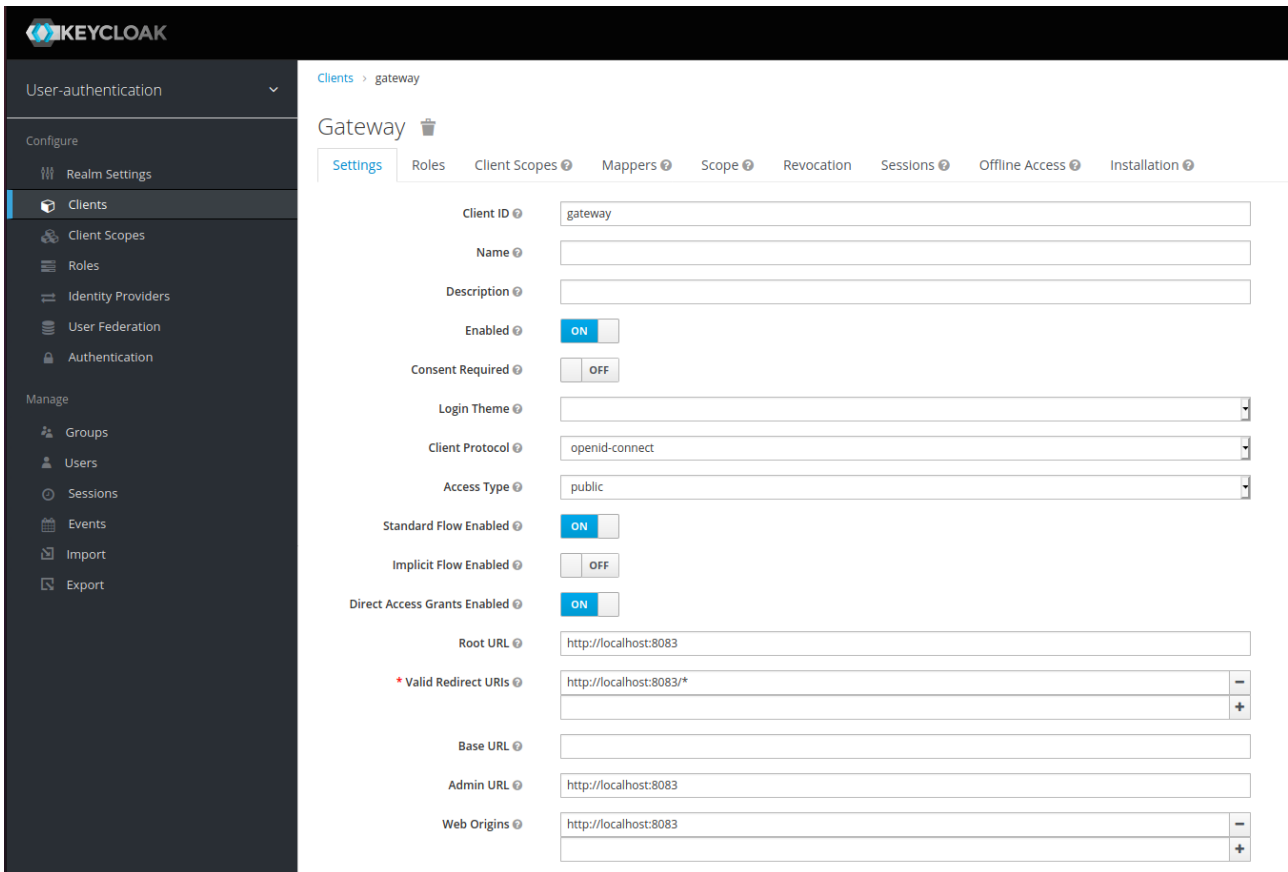


FIGURE 26 - KEYCLOAK REALM CREATION

The next step is the creation of a client inside our newly created realm. In Keycloak, clients are entities that can request Keycloak to authenticate a user. As mentioned before, it is also important here that the

new client is named appropriately compared to the “resource” attribute in the application.yml of the service we are trying to secure.

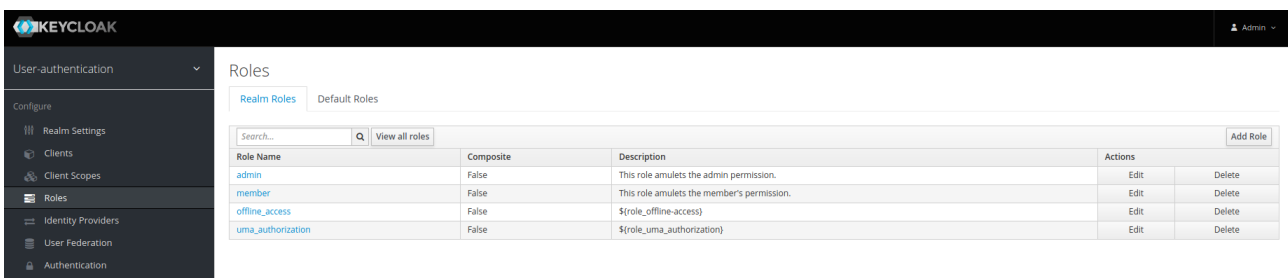


The screenshot shows the Keycloak Admin Console interface for creating a new client. The left sidebar contains navigation options under 'Configure' (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication) and 'Manage' (Groups, Users, Sessions, Events, Import, Export). The main panel is titled 'Clients > gateway' and 'Gateway'. It features several tabs: Settings (active), Roles, Client Scopes, Mappers, Scope, Revocation, Sessions, Offline Access, and Installation. The 'Settings' tab contains the following fields and controls:

- Client ID: gateway
- Name: (empty)
- Description: (empty)
- Enabled: ON
- Consent Required: OFF
- Login Theme: (dropdown)
- Client Protocol: openid-connect
- Access Type: public
- Standard Flow Enabled: ON
- Implicit Flow Enabled: OFF
- Direct Access Grants Enabled: ON
- Root URL: http://localhost:8083
- Valid Redirect URIs: http://localhost:8083/\* (with add and remove buttons)
- Base URL: (empty)
- Admin URL: http://localhost:8083
- Web Origins: http://localhost:8083 (with add and remove buttons)

FIGURE 27 - KEYCLOAK CLIENT CREATION

It is important to ensure that “Standard Flow Enabled” is selected as this enables standard OpenID Connect redirect-based authentication with authorization code. More specifically, in terms of OpenID Connect or OAuth2 specifications, this enables support of “Authorization Code Flow” [16] for this client. The “Valid Redirect URIs field must contain the location of the service we are trying to secure. Moving on, relevant roles for the created realm are defined from the Admin Console as shown in Figure 28.



The screenshot shows the Keycloak Admin Console interface for managing realm roles. The left sidebar is the same as in Figure 27. The main panel is titled 'Roles' and has two tabs: Realm Roles (active) and Default Roles. Below the tabs is a search bar and a 'View all roles' link. The main content is a table with the following data:

Role Name	Composite	Description	Actions	
admin	False	This role amulets the admin permission.	Edit	Delete
member	False	This role amulets the member's permission.	Edit	Delete
offline_access	False	\$(role_offline-access)	Edit	Delete
uma_authorization	False	\$(role_uma_authorization)	Edit	Delete

FIGURE 28 - KEYCLOAK REALM ROLES

User creation can also be handled by the Admin Console and for the purpose of this prototype a sample user with username “user1” is created.

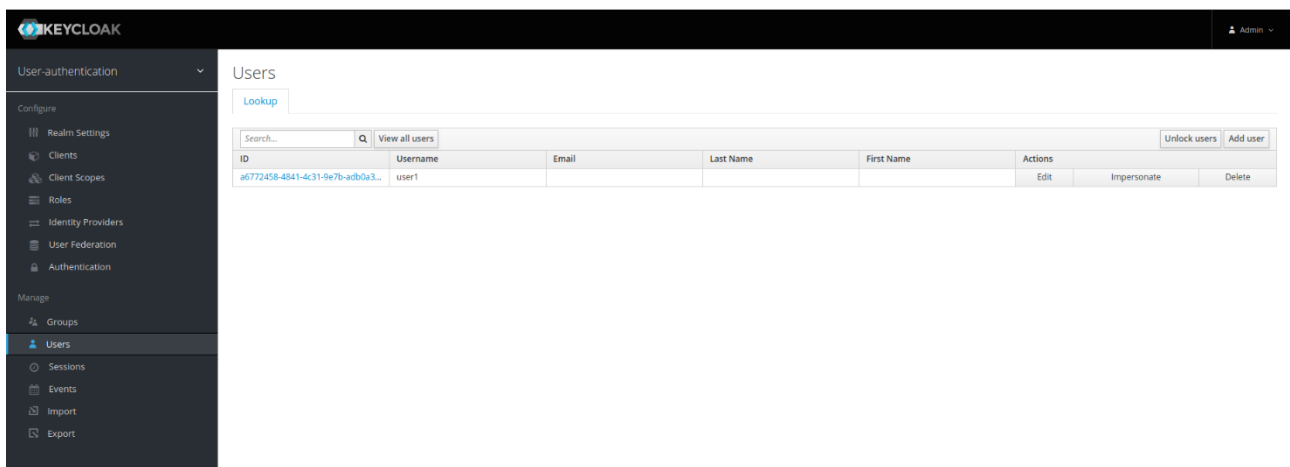


FIGURE 29 - KEYCLOAK USERS

At first the newly created user is not assigned the admin role but is given the member, offline\_access and uma\_authorization roles. This default behaviour can be changed through Keycloak settings accordingly for newly created or registered users.

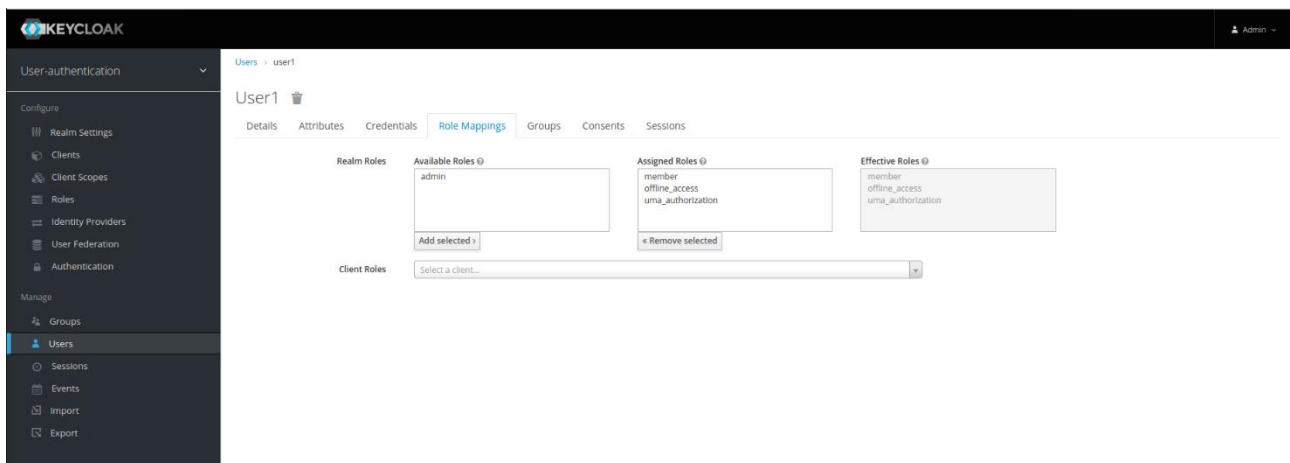
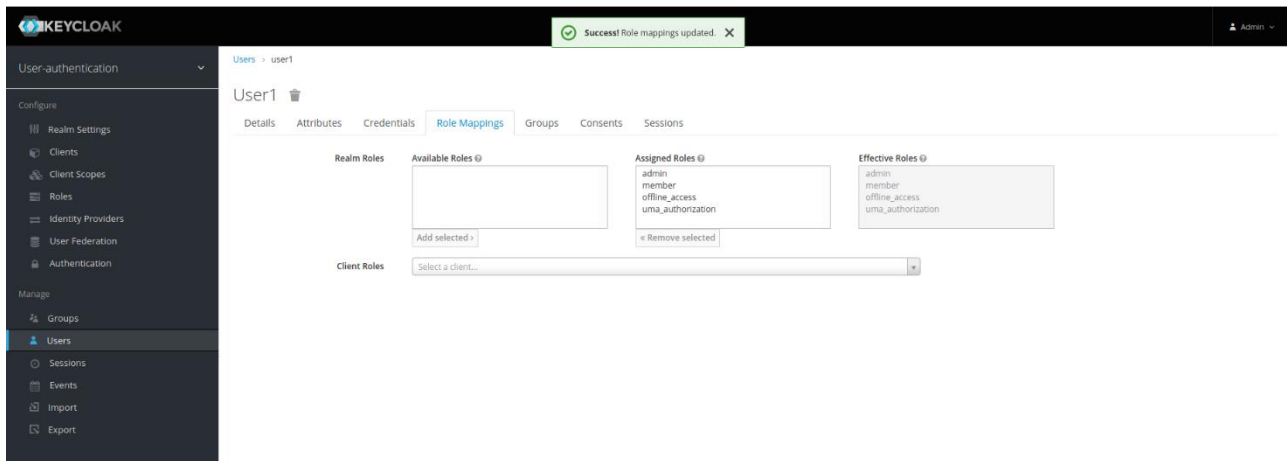


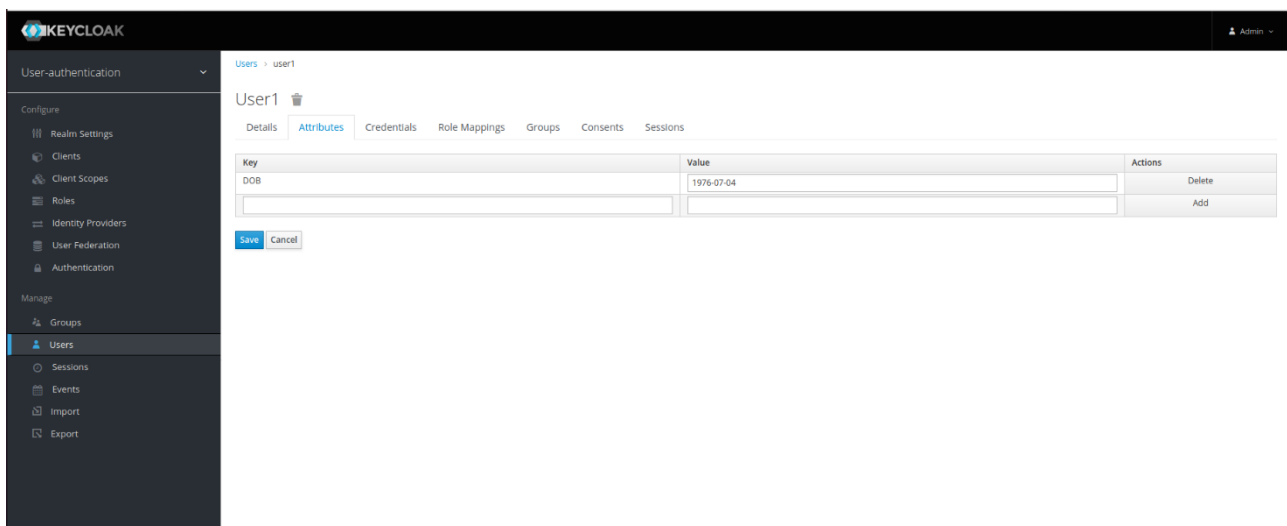
FIGURE 30 - INITIAL ROLE MAPPINGS

As shown in Figure 30, the Role Mappings Tab provides an overview of all the assigned Roles for this particular user. Roles can be added or removed accordingly and furthermore there is the option to assign client-specific roles via the relevant dropdown menu.



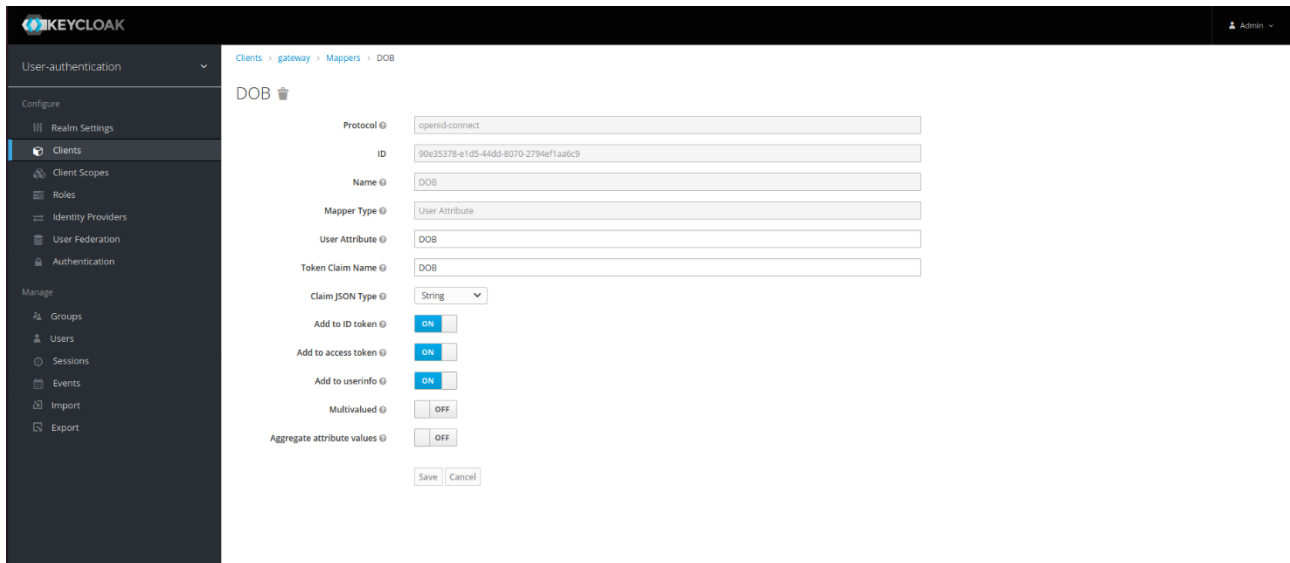
**FIGURE 31 – ADMIN ROLE GRANT**

Figure 31 presents an example of the admin Role assignment to our test user. Information about users in Keycloak is not restricted to Roles and Credentials, as each User can contain custom Attributes to further define their right in the PolicyCLOUD ecosystem. Figure 32 shows such an example with the Date of Birth of the user added as a custom Attribute from the relevant tab.



**FIGURE 32 - CUSTOM USER ATTRIBUTE**

These custom attributes are key for the PolicyCLOUD's ecosystem ability to create and manage complex Policies that contain multiple parameters for each User. It is essential therefore that Keycloak can communicate these attributes in a safe and secure way. This can be achieved via Client Mappers in the Keycloak Admin Console, as shown in Figure 33.



The screenshot shows the Keycloak Admin Console interface. On the left is a sidebar with navigation options: Configure (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication), Manage (Groups, Users, Sessions, Events, Import, Export). The main area is titled 'Clients > gateway > Mappers > DOB'. It shows the configuration for a 'DOB' client attribute mapper. Fields include: Protocol (openid-connect), ID (90e35378-e1d5-44dd-8070-2794ef1aabc9), Name (DOB), Mapper Type (User Attribute), User Attribute (DOB), Token Claim Name (DOB), Claim JSON Type (String), and checkboxes for 'Add to ID token', 'Add to access token', 'Add to userinfo', 'Multivalued', and 'Aggregate attribute values'. 'Save' and 'Cancel' buttons are at the bottom.

FIGURE 33 - CLIENT ATTRIBUTE MAPPER

There are multiple options available for the Mapper Type but our Use Case dictates that we use the “User Attribute”. The “Name” and “User Attribute” fields must match the name of the corresponding User Attribute from Figure 32, while the “Claim JSON Type” dictates the type of the mapped value. By selecting “Add to ID token”, “Add to access token”, “Add to userinfo” the attribute value is added as a claim to the relevant token. These can be used by the ABAC Client and Server in order to securely transmit the claims to the ABAC Engine.

### 4.3.3 ABAC Client Filter

The ABAC Client Filter can be included in a web client and trigger the necessary interception to the ABAC Authorization Engine. It depends on the ABAC Client and acts as an access filter to a specific API or website and restricts access until it receives authorization from the ABAC Engine. Furthermore, it handles the IDToken provided by Keycloak with the custom User Attributes needed for the Policy Evaluation and Enforcement. To include the Filter in a web client the relevant dependency must be added in the pom.xml

```
<dependencies>
  <dependency>
    <groupId>eu.policycloud.authorization.abac</groupId>
    <artifactId>abac-authorization-client</artifactId>
    <version>1.0.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

The next necessary step is to copy the truststore file of ABAC server into the application's resources directory. Take care not to expose this file publicly.

- Cp abac-authorization/server/src/main/resources/config/pdp-server-truststore.p12  
 <YOUR\_APP\_HOME> /src/main/resources/truststore-client.p12

The final step is to include the appropriate variables, either through a .env file or script

```
AZ_CLIENT_TRUST_STORE_FILE=truststore-client.p12
AZ_CLIENT_TRUST_STORE_TYPE=PKCS12
AZ_CLIENT_TRUST_STORE_PASSWORD=YOUR_PASSWORD_HERE
AZ_SERVER_ENDPOINTS=https://localhost:7071/checkJsonAccessRequest
AZ_SERVER_ACCESS_KEY=723568712658723154532175675265723123321765723
AZ_CALL_DISABLED=false
AZ_CALL_LOAD_BALANCE_METHOD=ORDER
AZ_CALL_RETRIES=1

export AZ_CLIENT_TRUST_STORE_FILE AZ_CLIENT_TRUST_STORE_TYPE
AZ_CLIENT_TRUST_STORE_PASSWORD AZ_SERVER_ENDPOINTS AZ_SERVER_ACCESS_KEY
AZ_CALL_DISABLED AZ_CALL_LOAD_BALANCE_METHOD AZ_CALL_RETRIES
```

If you want to change the API Key for both the server and client, create a long, difficult to guess (random) string. We suggest more than 32 characters long, including any combination of capital and plain letters, numbers and symbols. Avoid using phrases or values that can be guessed or extracted from context.

New API Key value must be set in both ABAC Server and ABAC Client configuration files.

- For ABAC Server, edit `authorization-server.properties` file and set property `pdp.access-key`.
- For ABAC Client, edit `authorization-client.properties` file and set property `pdp.access-key` or change the corresponding environment variable `AZ_SERVER_ACCESS_KEY`.

#### 4.3.4 ABAC Proxy

There can be cases where applying the ABAC Client filter mentioned in Section 4.3.3 is not possible due to limitations of the programming language or the technology stack used. That is the case with programs utilizing PHP and in order to integrate ABAC in those cases, the ABAC Proxy has been developed. It is based on the Zuul Proxy [17] and ensures that traffic coming to a specified port are checked against the currently implemented policy and if the conditions are satisfied, allows for the redirection to the underlying php application. In order to configure the proxy for a specific php application, the following values of its `application.yml` file should be set:



```
server:
  port: 80

proxy.rewrite.redirect_url: true

zuul:
  addHostHeader: true
  routes:
    phpapp:
      path: /php/**
      url: http://php-app:8080/
      sensitiveHeaders:
    all:
      path: /**
      url: http://pdp-server:7071/
      sensitiveHeaders:

ribbon:
  eureka:
    enabled: false
```

FIGURE 34 – ABAC PROXY APPLICATION.YML

The above file redirects all traffic to the php-app running locally, if the requests successfully pass through the ABAC PDP server located at `http://pdp-server:7071`. The routes section can be enhanced with more entries to fine-grain the needs of any application and allow for more specific routing to the php application.

### 4.3.5 Test Web Client

A simple Web Client can be used in order to test both the ABAC Server and Filter. The Web Client is a Spring-boot application that contains an ABAC Authorization Filter that is responsible for intercepting any access to the secured API's and instead redirects the user to login to Keycloak. Based on the User Attributes retrieved from the Keycloak login, as well as the implemented Policies evaluated at the ABAC Server, the request is either denied or permitted. An example of the aforementioned login page during the interception of a request to get the user's Date of Birth is shown in Figure 35 below.

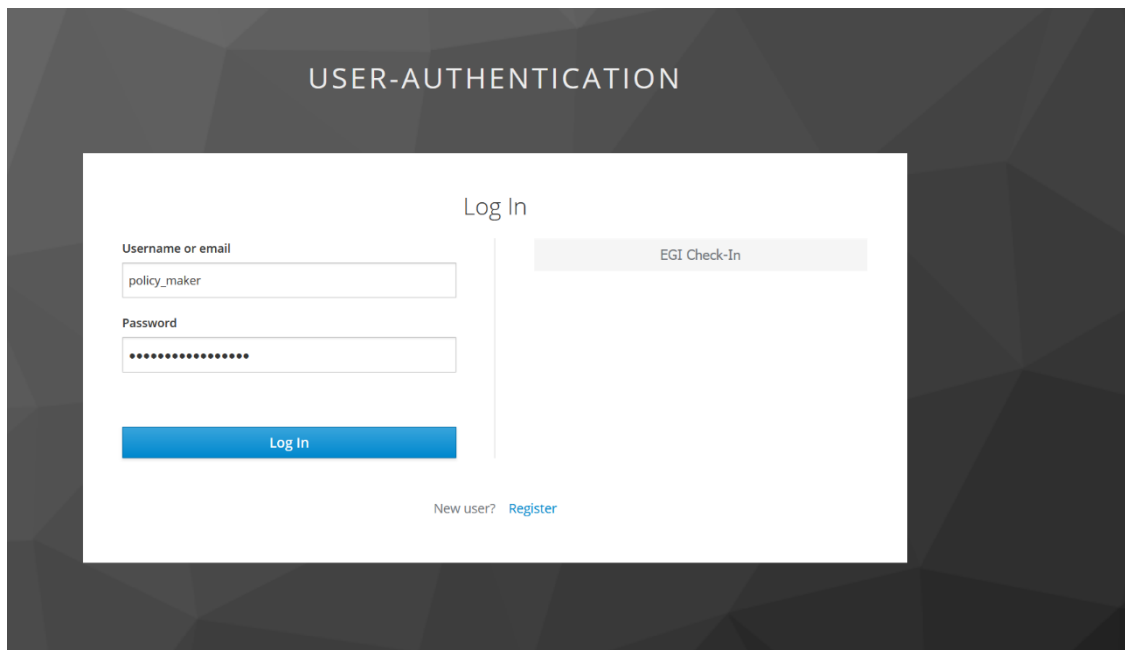


FIGURE 35 - INTERCEPT LOGIN

If the user login is successful and the ABAC Engine permits the request, based on the applied Policies, the Web Client is able to retrieve the userID and Date of Birth, as shown in Figure 36.

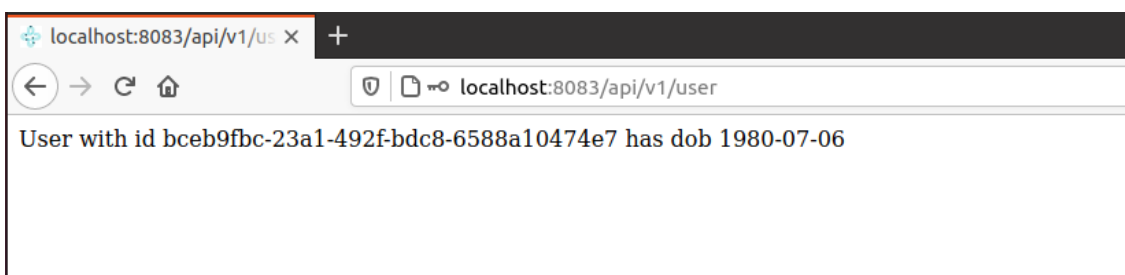
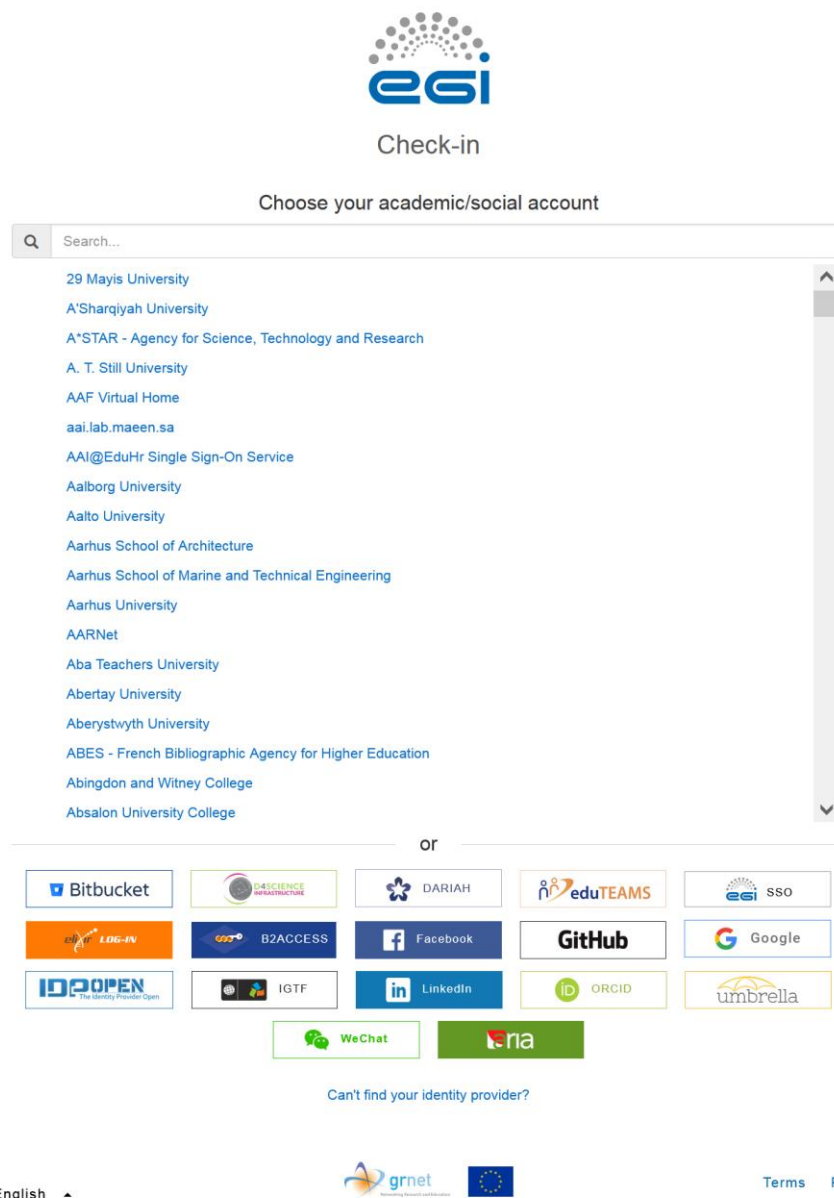


FIGURE 36 - SUCCESSFUL ATTRIBUTES RETRIEVAL

### 4.3.6 EGI Check-In integration

The need for managing user identity across borders between organizations, different domains and services, leads to the creation of federated identity environments. An Identity federation is a group of Identity and Service Providers that sign up to an agreed set of policies for exchanging information about users and resources to enable access to and use of the resources. Home organizations (e.g. a university, research institute, etc.), who operate an Identity Provider (IdP), register users by assigning a digital identity -- in this way, they are able to authenticate their users and provide a limited set of attributes that characterize the user in a given context. Service Providers (SPs) delegate the authentication to IdP, in order to control access to the provided resources. EGI Check-In [18] (a service provided by EGI Foundation) is a solution for federated identity management with the architecture of a proxy that operates as a central hub to connect federated Identity Providers and Service Providers.

In the PolicyCLOUD scenario, EGI Check-in proxy is provided like an alternative login button in the Keycloak service authentication page: through EGI Check-In users will be able to use their eduGAIN [19] credentials (institutional or academic accounts) as well as ORCID [20] credentials (researcher identifier that disambiguates researchers and their work) as well as most popular social media accounts (Google, LinkedIn, etc.) allowing PolicyCLOUD services to be easily accessible and adopted by a broader audience, while also maintaining the ability to have also internal users registered to the Keycloak service. The Keycloak Log In page shown in Figure 35 contains a link on the right side with the EGI Check-In label. This provides an alternative method of logging in and redirects the user to the integrated EGI Check-In page, as shown in Figure 37 below.



EGI  
Check-in

Choose your academic/social account

Search...

- 29 Mayis University
- A'Sharqiyah University
- A\*STAR - Agency for Science, Technology and Research
- A. T. Still University
- AAF Virtual Home
- aai.lab.maeen.sa
- AAI@EduHr Single Sign-On Service
- Aalborg University
- Aalto University
- Aarhus School of Architecture
- Aarhus School of Marine and Technical Engineering
- Aarhus University
- AARNet
- Aba Teachers University
- Abertay University
- Aberystwyth University
- ABES - French Bibliographic Agency for Higher Education
- Abingdon and Witney College
- Absalon University College

or

Bitbucket CAS DARIAH eduTEAMS EGI SSO eIDAS B2ACCESS Facebook GitHub Google IDP OPEN IGTF LinkedIn ORCID umbrella

WeChat

Can't find your identity provider?

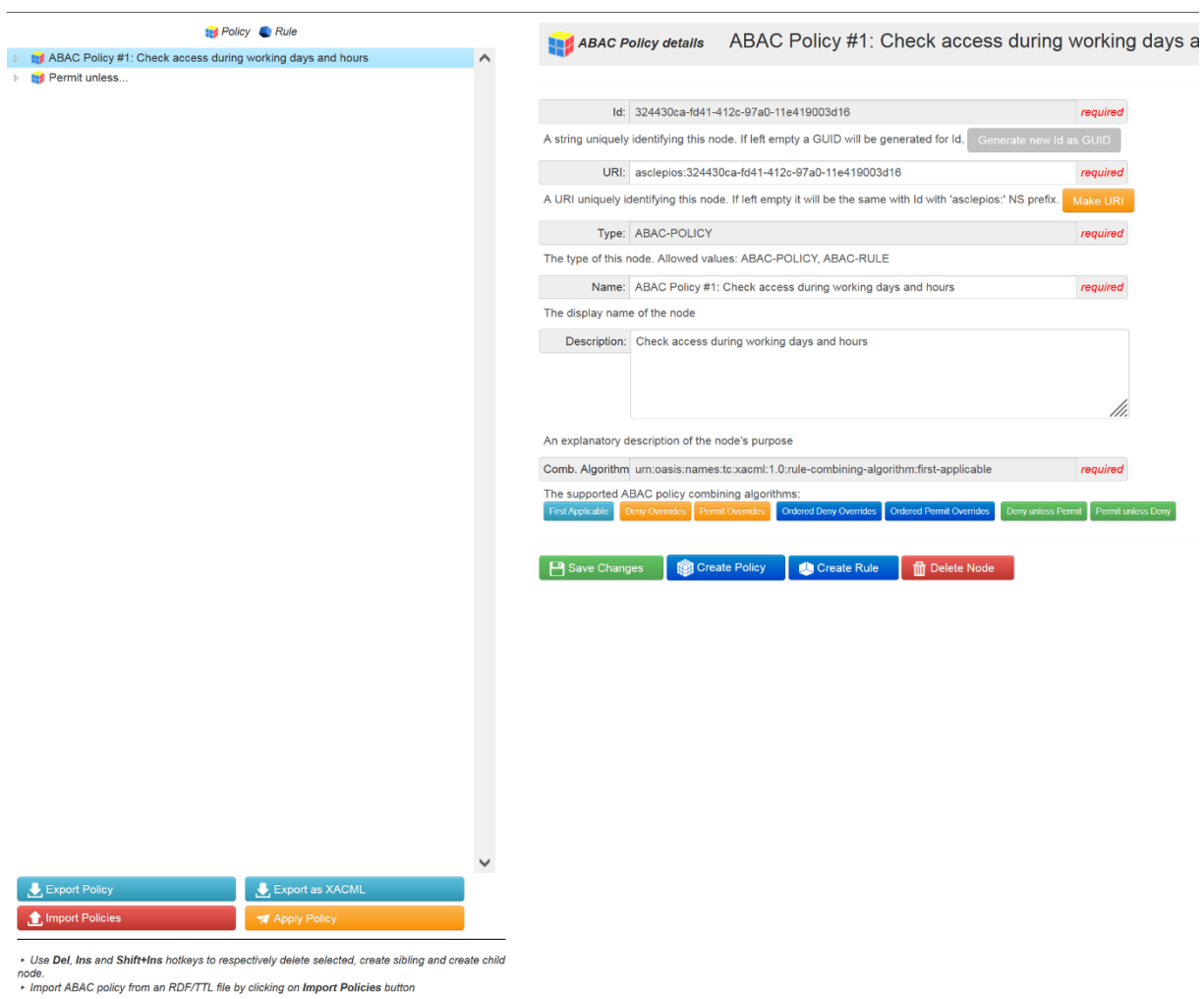
English grnet EU Terms Privacy

FIGURE 37 - INTEGRATED EGI CHECK-IN

A user can select its Identity Provider from that page and input his/her credentials from the Identity Provider login page as usual. This allows users who are not already registered to the PolicyCLOUD ecosystem and Keycloak to use some of the provided tools, using their academic accounts or accounts from popular providers like Google, Facebook, LinkedIn, etc. This integration with EGI Check-In allows PolicyCLOUD to be easily accessible and adopted by a broader audience, while also maintaining the ability to have internal users registered to the Keycloak Server.

### 4.3.7 XACML Editor

In order to further improve the usability of the PolicyCLOUD platform and ease the process of creating and managing access policies the XACML Editor was introduced as a user interface. The editor is an open-source software (<https://gitlab.com/asclepios-project/ample-editor>) that allows users to easily create XACML Policies and enrich them with access rules in a user-friendly graphic environment. In this way, the more complex way of defining policies in the XML format is avoided and the adoption of the PolicyCLOUD solution is further helped for a larger audience, without the need of certain technical knowledge, like the creation and editing of an XML file. A sample of a created XACML file can be found in Figure 38 below.



The screenshot displays the 'ABAC Policy details' page for 'ABAC Policy #1: Check access during working days and hours'. The interface includes a left sidebar with a tree view showing the policy structure. The main area contains various fields for policy configuration, many of which are marked as 'required'.

**Policy Details:**

- Id:** 324430ca-fd41-412c-97a0-11e419003d16 (required)
- URI:** asclepios:324430ca-fd41-412c-97a0-11e419003d16 (required)
- Type:** ABAC-POLICY (required)
- Name:** ABAC Policy #1: Check access during working days and hours (required)
- Description:** Check access during working days and hours
- Comb. Algorithm:** urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable (required)

**The supported ABAC policy combining algorithms:**

- First Applicable
- Deny Overrides
- Permit Overrides
- Ordered Deny Overrides
- Ordered Permit Overrides
- Deny unless Permit
- Permit unless Deny

**Actions:**

- Save Changes
- Create Policy
- Create Rule
- Delete Node

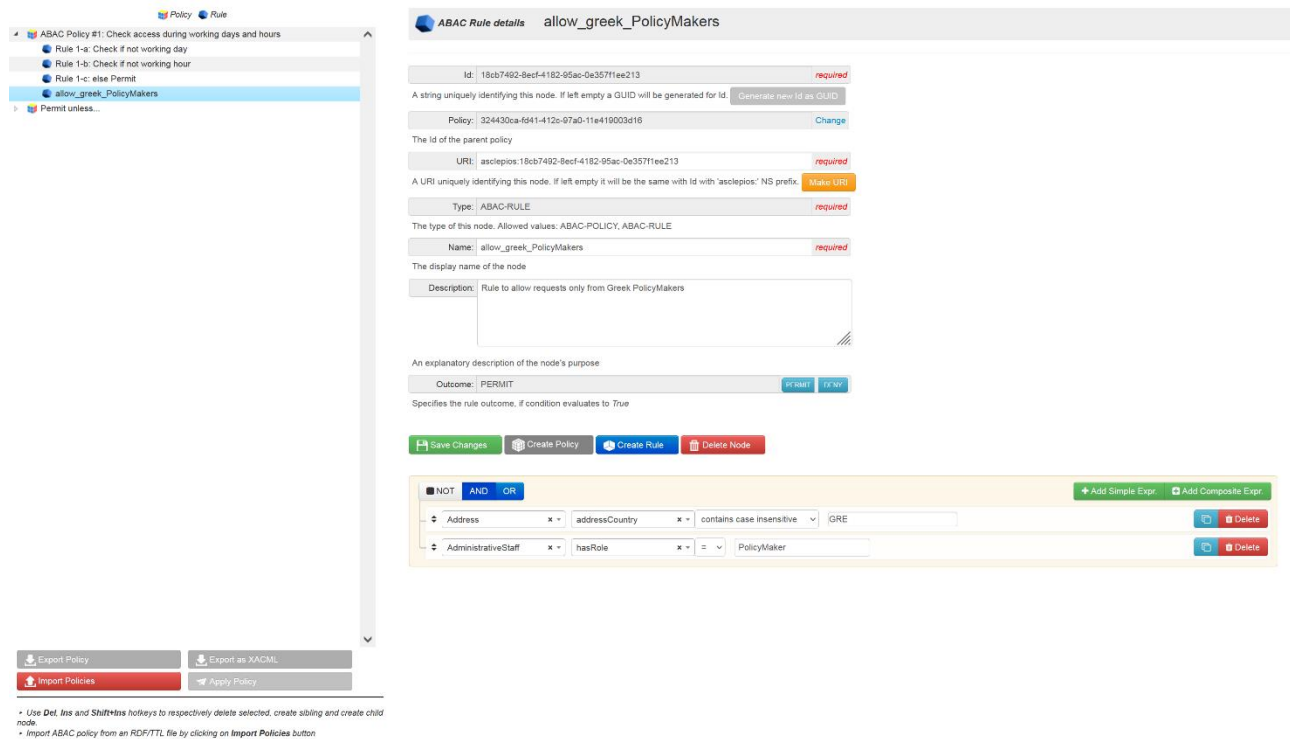
**Bottom Bar:**

- Export Policy
- Export as XACML
- Import Policies
- Apply Policy

• Use **Del**, **Ins** and **Shift+Ins** hotkeys to respectively delete selected, create sibling and create child node.  
 • Import ABAC policy from an RDF/TTL file by clicking on **Import Policies** button

FIGURE 38 - SAMPLE ACCESS POLICY IN EDITOR

This particular XACML policy allows access to the system only during working days and hours. The user can edit all its parameters or create a new policy altogether. An example of adding a new rule to an already existing policy can be found on Figure 39 below.



The screenshot displays the 'ABAC Rule details' for a rule named 'allow\_greek\_PolicyMakers'. The left sidebar shows a tree view with 'ABAC Policy #1: Check access during working days and hours' expanded, containing 'Rule 1-a: Check if not working day', 'Rule 1-b: Check if not working hour', 'Rule 1-c: else Permit', and the selected 'allow\_greek\_PolicyMakers' rule. The main panel shows the rule configuration with fields for Id, Policy, URI, Type, Name, and Description. Below these are buttons for 'Save Changes', 'Create Policy', 'Create Rule', and 'Delete Node'. At the bottom, there is a section for 'Expressions' with a table showing conditions: 'Address' (addressCountry) contains case insensitive 'GRE' and 'AdministrativeStaff' (hasRole) is 'PolicyMaker'. The bottom of the interface includes buttons for 'Export Policy', 'Export as XACML', 'Import Policies', and 'Apply Policy', along with a small note about using Del, Ins, and Shift+Ins hotkeys.

FIGURE 39 - SAMPLE ACCESS RULE IN EDITOR

This newly created rule enhances the previous XACML policy by restricting access during working hours to all but Greek Policy makers, as can be seen from the expressions and attributes on the bottom. A user can in this practical way create, combine, delete and manage XACML policies and rules to fit the security needs of the ecosystem.

### 4.3.8 Custom Access Policies for Gateways

To further refine the access of users in the gateways component, access policies based on the URL of the request have been developed. In this way, each access request is evaluated based on its URL and custom access policies have been developed for the APIs listed in Figure 6. Each rule can be applied to one or more of those endpoints and can further fine-tune access to the corresponding gateway. As a practical example, the /api/sofia\_air/populate URL can be secured by a targeted access policy which includes this URL and restricts access to specific working hours and to users who have the "Sofia Municipality" attribute in their profile. This rule will be applied only to requests for this URL and will not affect access to the other resources of the gateway. A part of the policies implemented for restricting access to working hours and more specifically the rule that checks if the current time is after the shift start of the subject requesting access is presented in Figure 40.

```
<!-- Rule Name: Rule 1 - Current Time after Shift Start -->
<xacml3:Rule Effect="Deny" RuleId="31aa3bfa-c095-4f10-a6c7-a9d759a8e54b">
  <xacml3:Condition>
    <!-- COMPOSITE EXPRESSION - START: operator: AND -->
    <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
        <!-- SIMPLE EXPRESSION - START: manually created -->
        <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal">
          <!-- Current Date/Time -->
          <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:dateTime-one-and-only">
            <xacml3:AttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-dateTime"
              Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
              DataType="http://www.w3.org/2001/XMLSchema#dateTime"
              MustBePresent="true"
            />
          </xacml3:Apply>
        </xacml3:Apply>
      <!-- timestamp-shift-start -->
      <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:dateTime-one-and-only">
        <xacml3:AttributeDesignator
          AttributeId="timestamp-shift-start"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
          DataType="http://www.w3.org/2001/XMLSchema#dateTime"
          MustBePresent="true"
        />
      </xacml3:Apply>
    </xacml3:Apply>
  </xacml3:Apply>
  <!-- SIMPLE EXPRESSION - END -->
</xacml3:Condition>
<!-- Rule Obligations -->
</xacml3:Rule>
```

FIGURE 40 - ABAC CUSTOM RULE FOR ACCESS DURING SHIFT

## 4.4 Interfaces

The following APIs are provided by Keycloak and are the most used ones for requesting access from the component, retrieving user attributes and managing the server.

The following APIs are provided by the Cloud Backbone component.

Method	Path	Description
POST	{realm}/protocol/openid-connect/token	Token retrieval for user or service
PUT	/ {realm}/users/{id}	User Update
DELETE	/ {realm}/users/{id}	User Deletion
POST	/ {realm}/users	Create User and set user attributes
GET	{realm}/users?username={username}	Retrieve user attributes by username
GET	{realm}/users/{id}	Retrieve user attributes by user-id
GET	/ {realm}/users	Retrieve all realm users
PUT	/ {realm}/users/{id}/reset-password	Reset the user password
GET	/ {realm}/users/{id}/role-mappings/clients / {client}/available	Retrieve roles that can be mapped to the user
GET	/ {realm}/clients/{id}/roles	Retrieve all roles of client
GET	/ {realm}/roles	Retrieve all roles of realm
GET	/ {realm}/users/{id}/role-mappings/realm	Get realm level role mapping

TABLE 1 - KEYCLOAK COMMON APIS

All the APIs provided by the Keycloak ADMIN REST API [21] are also available to be used, with the retrieval of a valid Keycloak Admin token.

## 4.5 Baseline Technologies and Tools

### 4.5.1 Balana

Balana [22] was the first open-source reference implementation of the XACML protocol, is a widely adopted solution and has been used for the needs of PolicyCLOUD. It supports the entire lifecycle of authorization processing. It is tightly integrated into the WSO2 Identity Server [23]. Balana, as XACML engine of the WSO2 Identity Server has two major components, the Policy Administration Point (PAP) and Policy Decision Point (PDP). Figure 41 below presents the component architecture of the PDP that is our main interest.

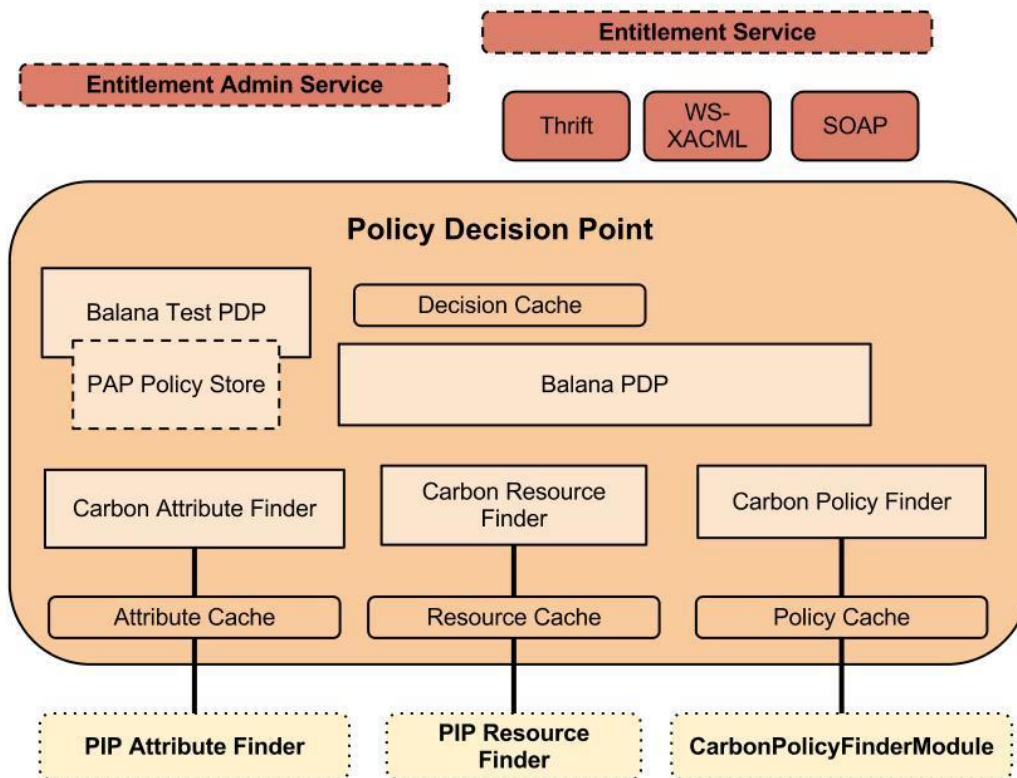


FIGURE 41 - BALANA PDP

More details on the components of in the PDP architecture are presented below.

**Entitlement Admin Service** provides an API that is used to expose all PDP configurations, such as:

- Invalidating caches
- Refreshing policy, attribute, resource finder modules
- Retrieving PDP configurations
- Testing the PDP

**Entitlement Service** provides XACML authorization API that supports the following three communication methods with PEP.



- SOAP-based Web service
- Apache Thrift binary protocol [24]
- WS-XACML

**Balana PDP** is the core of the engine of Balana

**Balana Test PDP** is a duplication of Balana PDP can be only used for testing policies.

**Carbon Policy Finder** is a module that finds policies from different policy stores to evaluate an XACML request. Figure 42 presents a high-level diagram of the usage of the carbon policy filter for the collection of the policies to be evaluated.

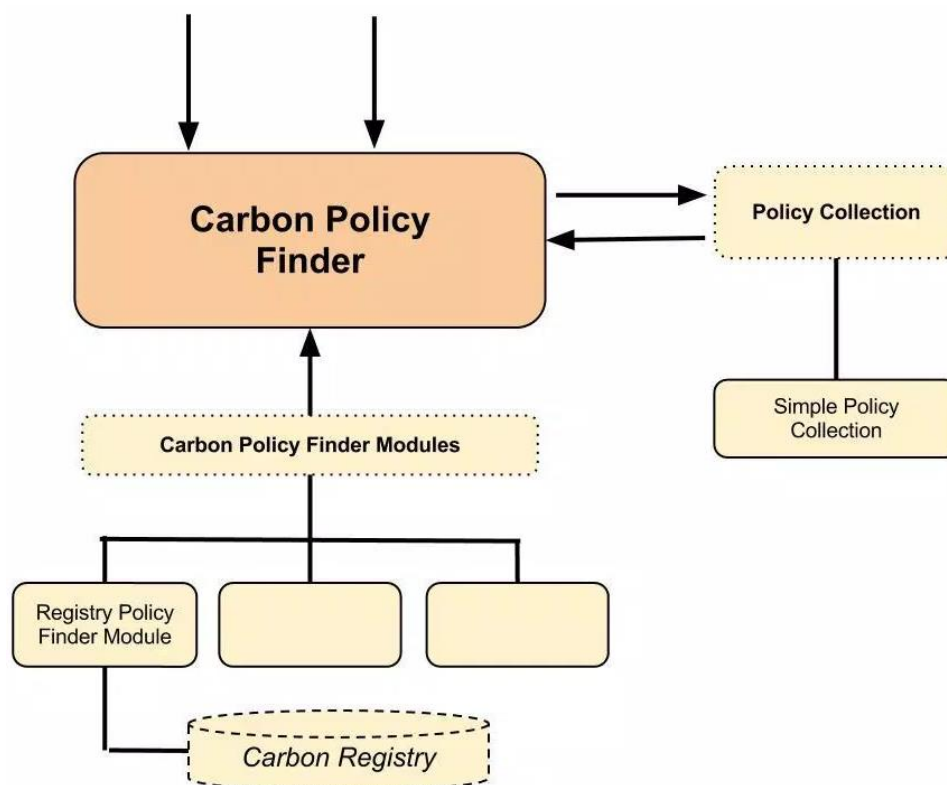


FIGURE 42 - CARBON POLICY FILTER

Policy finder modules implementing the CarbonPolicyFinderModule interface should be registered and plugged with the Carbon policy finder. WSO2 Identity Server provides by default a Carbon registry-based policy finder module that can retrieve policies from a registry collection. Carbon policy finder finds XACML requests and creates an effective policy. When an update in the policy store happens, Carbon policy finder can be re-initialized automatically by the module, or it can be re-initialized using the API of the Entitlement Admin Service.

**Carbon Attribute Finder** is a module that is responsible for finding missing attributes for a given XACML request, using the underlying PIP attribute finders. Figure 43 below provides a high-level diagram for both the Carbon attribute finder and resource finders.

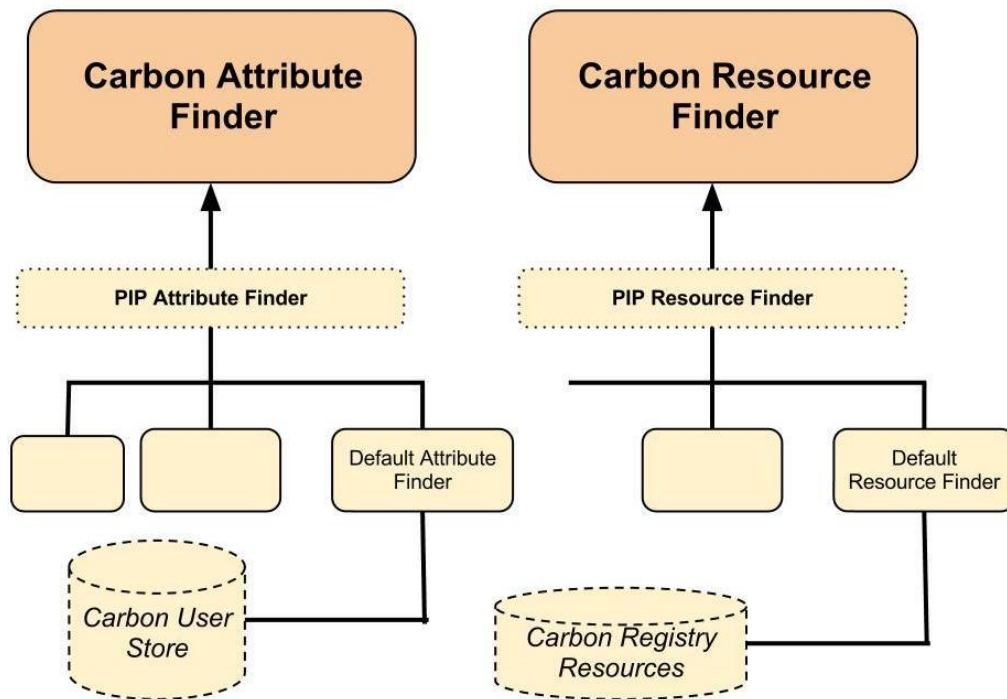


FIGURE 43 - CARBON ATTRIBUTE FINDER

A PIP attribute finder module should implement the PIPAttributeFinder interface and register it using the entitlement properties configuration file to the Carbon attribute finder. WSO2 Identity Server by default communicates with the underlying user store of the Identity Server that is built with ApacheDS [25].

On runtime, Carbon attribute finder checks for the attribute Id and hands it over to the proper module to handle, while caching mechanism (provided by Carbon attribute finder) is used for caching the findings when possible.

**Carbon Resource Finder** is used to retrieve children or descendant resources of a given root level resource value, used to fulfil requirements for a multiple decision profile. Similarly to the PIP attribute finder module, it has to implement the PIPAttributeFinder interface.

#### 4.5.2 Keycloak

Keycloak [15] is probably the most **powerful authentication proxy for micro-services** and legacy systems. As such, it **abstracts the functionality of identity extraction and identity verification** for different systems and for different protocols. In parallel, it is able to map users and roles from existing legacy systems in what it calls **authentication realms**. Through configured realms, Keycloak is able to centralize the login-process of various systems through the implementation of many protocols such as OAuth2.0 [26] and OpenIDConnect [27] (a.k.a. OIDC). The OpenIDConnect signalling is presented in Figure 44.

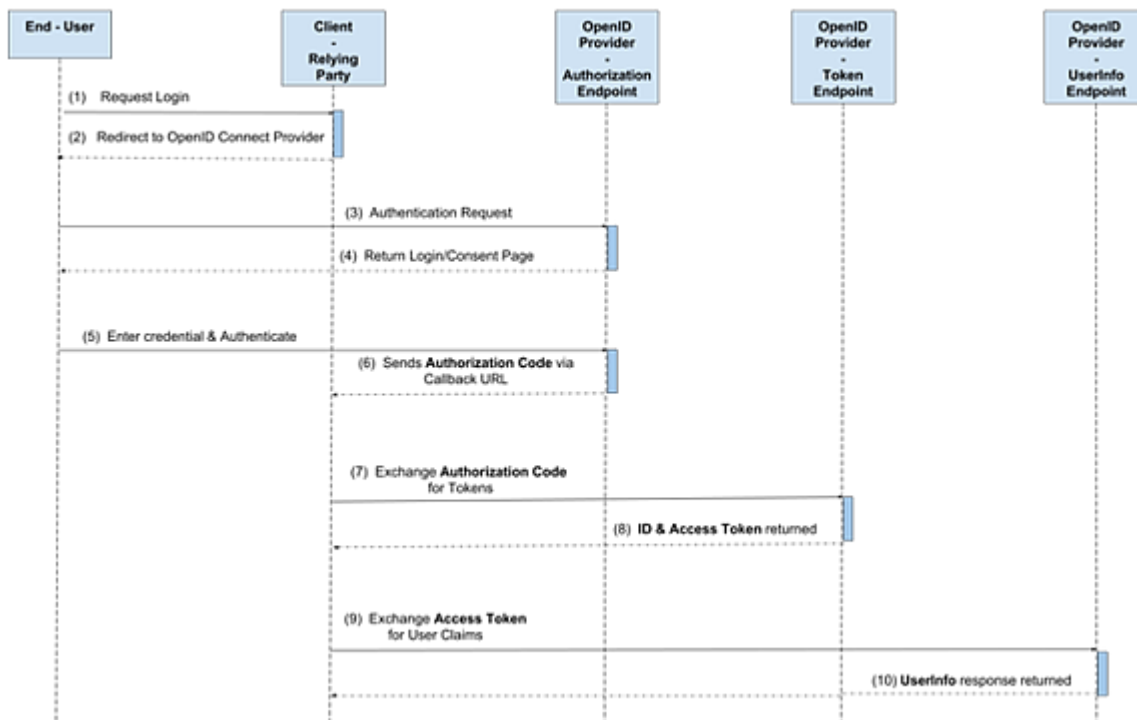


FIGURE 44 - OIDC SIGNALLING

According to the flow diagram, a user is attempting to connect to a service which supports OIDC. The health care service is redirecting the user to an OIDC provider that is configured to authenticate users based on **a service-id and a user-role mapping**. The **combination** of the service-id and the user-role-mapping is addressed as a **realm**.

The OIDC server is “challenging” the user to authenticate based on various methods (username/password, X509 certificate etc.). Our software prototype utilizes the username/password method and upon successful login a distinct set of claims are serialized as a token back to the user in order to use it in his/her interaction with the service. **These claims contain electronically signed attributes that can be used by the ABAC authorization engine.**

As a result, the OIDC signalling (and Keycloak in general) is extremely crucial for PolicyCLOUD ABAC even if it is not an ABAC engine per se. It acts as an **enabler of verifier attributes and attribute provider**.

## 4.6 Source Code

### Code Overview and Availability

The docker image for the ABAC Server as well as the image and the code for the ABAC Proxy can be found at [https://registry.grid.ece.ntua.gr/oikonomou/abac\\_proxy](https://registry.grid.ece.ntua.gr/oikonomou/abac_proxy).

Access is restricted to members of the project consortium.

## 4.7 Deployment Status

The Keycloak Server containing the realms, clients and users of the PolicyCLOUD ecosystem is deployed at <https://policycloud-auth.euprojects.net>. The ABAC Server has been deployed together with the Gateways it protects on a public VPS server in the project's cloud infrastructure that has been provided and is supported by RECAS-BARI and EGI.

## 5 Conclusions

In this document the progress in the technical work of the tasks T3.1, T3.3, T3.4, and T3.6 until M34 (October 2022) of the project was presented. EGI operates the cloud-based infrastructure for the project and monitors its performance on a regular basis so no new mention is made to the INDIGO-DataCloud PaaS Orchestrator. Members of the project continue to have access and deployment capabilities of virtual clusters on top of the IaaS resources through this component.

Furthermore, in this document the status of the cloud gateways component has been reported. These gateways are responsible for obtaining data from heterogeneous data sources; gateways for Maggioli, Aragon, London and Sofia use-cases have been provided through various services like the global terrorism database and Twitter. More microservices and APIs have been made available to satisfy the needs of the PolicyCLOUD pilots and the integration between Cloud Gateways & APIs component has been implemented for all the endpoints, while the integration with the user authorization mechanism has been completed, thus ensuring that all the required security standards are met.

Regarding Incentive Management, the software prototype presented here is the final one since no further development was deemed necessary. However, the necessary integration and deployment steps have been concluded and are presented.

Finally, in section 4, the components and technologies that are used to provide an ABAC based access control mechanism suitable for PolicyCLOUD were described. A project wide authentication mechanism, called Keycloak, that ensured a common user base and secure authentication in the PolicyCLOUD ecosystem is also presented in this Section and has been integrated with the Marketplace, the PDT and the cloud gateways. This final prototype contains eight key components that have been combined, along with a test client, in order to provide the required access control capabilities to the use cases. To further enhance this, custom access policies have been developed for all the gateway APIs to fine-tune access to each URL, based on the scenario. Finally, to boost the number of potential adopters of the prototype and increase the usability of it, an alternative way to authenticate to it through EGI Check-In has been implemented and a new XACML editor for policy development has been created.

This version of the deliverable describes the third and final version of the project cloud infrastructure incentives management and data governance software prototype.

## References

- [1] PolicyCLOUD D3.1, Cloud Infrastructure Incentives Management and Data Governance: Design and Open Specification 1, 2020
- [2] PolicyCLOUD D3.4, Cloud Infrastructure Incentives Management and Data Governance: Design and Open Specification 2, 2021
- [3] PolicyCLOUD D3.7, Cloud Infrastructure Incentives Management and Data Governance: Design and Open Specification 3, 2022
- [4] PolicyCLOUD D3.2, Cloud Infrastructure Incentives Management and Data Governance: Software Prototype 1, 2020
- [5] PolicyCLOUD D3.5, Cloud Infrastructure Incentives Management and Data Governance: Software Prototype 2, 2021
- [6] PolicyCLOUD D6.11, Use Case Scenarios Definition & Design, 2022
- [7] PolicyCLOUD D6.3, Use Case Scenarios Definition & Design, 2020
- [8] Twitter Search Tweets, <https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-recent>.
- [9] Twitter Filtered Stream, <https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction>.
- [10] Twitter Rate Limits, <https://developer.twitter.com/en/docs/twitter-api/rate-limits>
- [11] Swagger Stats, <https://www.npmjs.com/package/swagger-stats>.
- [12] Prometheus, <https://prometheus.io/docs/introduction/overview/>
- [13] Grafana, <https://grafana.com/>
- [14] SheetJS, <https://docs.sheetjs.com/>.
- [15] KeyCloak, <https://keycloak.org>
- [16] C. Flow, <https://auth0.com/docs/flows/authorization-code-flow>
- [17] Netflix ZUUL Proxy, <https://github.com/Netflix/zuul>

- [18] EGI Check-In, <https://www.egi.eu/services/check-in/>
- [19] eduGAIN, <https://edugain.org/>.
- [20] ORCID, <https://orcid.org/>
- [21] Keycloak Admin REST API, <https://www.keycloak.org/docs-api/5.0/rest-api/index.html>
- [22] Balana, <https://github.com/wso2/balana>
- [23] W. I. Server, <https://wso2.com/identity-and-access-management/>.
- [24] Thrift, <https://thrift.apache.org/>.
- [25] ApacheDS, <https://directory.apache.org/apacheds/>.
- [26] O. 2.0, <https://oauth.net/2/>.
- [27] OpenIDConnect, <https://openid.net/connect>