CLOUD FOR DATA-DRIVEN POLICY MANAGEMENT

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

D3.2 CLOUD INFRASTRUCTURE INCENTIVES MANAGEMENT AND DATA GOVERNANCE SOFTWARE PROTOTYPE 1

Dissemination Level	PU
Due Date of Deliverable	31/10/2020, Month 10
Actual Submission Date	30/11/2020
Work Package	WP3 Cloud Infrastructures Utilization & Data Governance
Task	T3.1, T3.3, T3.4, T3.6
Type	Demonstrator
Approval Status	Final
Version	V1.0
Number of Pages	p.1 – p.38

Abstract: This document is an accompanying report providing information about the demonstrator of the first 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 specifications.

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	09/10/2020	TOC and assignments released	Giannis Ledakis, Konstantinos Theodosiou
0.2	23/10/2020	EGI contribution for section 2	Giuseppe La Rocca
0.3	06/11/2020	ATOS contribution for section 4	Maria Angeles Sanguino, Jorge Montero, Ana Luiza Pontual, Miquel Milà, Tomas Pariente and Ricard Munné
0.4	10/11/2020	UBI contribution for section 5, first merged version	Giannis Ledakis, Konstantinos Oikonomou
0.5	12/11/2020	Updates on section 2	Giuseppe La Rocca
0.6	13/11/2020	UPRC contribution for section 3	Ilias Maglogiannis, Thanos Kiourtis, Argyro Mavrogiorgou, George Manias
0.7	17/11/2020	All content in place, document template updated, ready for review	Giannis Ledakis, Konstantinos Oikonomou
0.8	22/11/2020	Internal Review by UPRC	Thanos Kiourtis, Argyro Mavrogiorgou
0.9	23/11/2020	Changes addressed after Internal Review	Giannis Ledakis, Konstantinos Oikonomou
0.10	25/11/2020	Comments for Section 4 addressed by ATOS	Maria Angeles Sanguino
0.11	27/11/2020	Comments for Section 2 addressed by EGI	Giuseppe La Rocca
0.12	27/11/2020	Final Version, Ready for QA check and submission	Giannis Ledakis, Konstantinos Oikonomou
0.13	28/11/2020	Quality Check	Argyro Mavrogiorgou
1.0	30/11/2020	Final version ready for submission	Giannis Ledakis

Author List

Organisation	Name
ATOS	Maria Angeles Sanguino, Jorge Montero, Ana Luiza Pontual, Miquel Milà, Tomas Pariente, Ricard Munné
EGI	Giuseppe La Rocca
UBITECH	Giannis Ledakis, Konstantinos Theodosiou, Konstantinos Oikonomou
UPRC	Ilias Maglogiannis, Thanos Kiourtis, Argyro Mavrogiorgou, George Manias, Fotis Karagiannis

Abbreviations and Acronyms

Abbreviation/Acronym	Definition
ABAC	Attribute Based Access Control
API	Application Programming Interface
CSV	Comma Separated Values
EC	European Commission
EOSC	European Open Science Cloud
FTP	File transfer Protocol
GTD	Global Terrorism Database
HPC	High Performance Computing
OID	OpenId Connect
OLA	Operational Level Agreement
ORM	Object Relational Mapping
PAP	Policy Administration Point
PDP	Policy Decision Point
SLA	Service Level Agreement
SQL	Structured Query Language
TRL	Technology Readiness Level
UI	User Interface
XACML	eXtensible Access Control Markup Language

Contents

List of Tables

List of Figures

Executive Summary

The first 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] specifications. The prototype's cloud infrastructure will be supported by RECAS-BARI and will be utilized by EGI through cloud gateways. These gateways allow the prototype to gather data from heterogenous data sources, such as from twitter and the global terrorism database.

This first version of the prototype also includes an ABAC based access control mechanism suitable for PolicyCLOUD. This is broken down to 3 key components that can be combined, along with a test client, in order to demonstrate the access control capabilities to the use cases and to proceed with the definition of an accurate data model, based on the provided feedback. An updated version of this prototype will be delivered and documented at a later stage for deliverables D3.4 and D3.7.

1 Introduction

This document is an accompanying report for the first iteration of "Cloud Infrastructure Incentives Management and Data Governance: Software Prototype", and is the second 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], all task participants collaborated towards the implementation of their corresponding outcomes to be utilized or integrated in the platform in later stages, and this first prototype is a snapshot of this effort. In the scope of *T3.1 - Cloud Provisioning of the PolicyCLOUD Infrastructure*, focus was provided for delivering the actual infrastructure to describe the tools that will be used. In the scope of *T3.3 - Cloud Gateways & APIs for Efficient Data Utilization* and *T3.4 - Incentives Management*, the first prototypes of cloud getaways and Incentive management have been described respectively. Finally, for *T3.6 - Data Governance Model, Protection and Privacy Enforcement* the first prototype of the mechanism that is used for privacy enforcement and the protection of data is described. For all these tasks, this document provides the preliminary work performed until M10.

1.1 Structure of the document

The rest of the document is structured as follows. Section 2 covers the provisioning process for the cloud infrastructure of PolicyCLOUD, while Section 0 presents the Cloud Gateway components of the first prototype. Section 4 describes the components for the identification and management of incentives, as provided for the first prototype. Section 0 presents the first prototype of the data privacy mechanism. Finally, Section 6 provides the conclusion of the document, while Section 7 adds the document references.

2 Cloud Provisioning of the PolicyCLOUD Infrastructure

The PolicyCLOUD infrastructure is supported by the RECAS-BARI [2], one cloud provider of the EGI Federation [3] selected through an open call sent by EGI.eu after the 1st General Assembly meeting. To provision the cloud resources during the project lifetime and guarantee a certain level of performance, a dedicated Service Level Agreement (SLA) and an Operational Level Agreement (OLA) have been agreed with the customer (represented by ATOS) and the cloud provider respectively. The SLA/OLA approval date is October, 13rd 2020. Through the present agreement the cloud provider will allocate the following resources from August 2020 to December 2022:

Cloud Compute	
Resource Centre	**RECAS-BARI**
Category:	Cloud Compute
Number of virtual CPU cores:	68
Memory per core (GB):	304
Public IP addresses:	Yes. Access to the VPN is also provided.
Allocation type:	Pledged
Other technical requirements:	The INDIGO-DataCloud PaaS orchestrator [4] will be available on the same resources, without an additional cost.
Payment mode offer:	Pay-for-use[1]
Duration:	01/08/2020 - 31/12/2022

Online Storage	
Resource Centre	**RECAS-BARI**
Category	Online Storage
Guaranteed storage capacity [TB]:	2TB
Opportunistic storage capacity [TB]:	N/A
Standard interfaces supported:	POSIX/Object Storage
Storage technology:	N/A
Other technical requirements:	Additional 50-100GB of OpenStack Swift Storage will be made available, without an additional cost.
Duration:	01/08/2020 - 31/12/2022
Payment ode offer:	Pay-for-use
Allocation type:	Pledged

Virtual Organisation	
Supported VOs:	vo.policycloud.eu
VO ID card:	https://operations-portal.egi.eu/vo/view/voname/vo.policycloud.eu
VO-wide list:	https://appdb.egi.eu/store/vo/vo.policycloud.eu

[1] See service offer for specifications (e.g. pricing, administration)

Virtual Organisation	
Provider AUP link	https://documents.egi.eu/document/2623

Service Offer/Cost [€]	
Compute	20,000€
Storage	Free (included in the compute costs)
Technical support	5,000€
Total	25,000€[2]

TABLE 1 - RESOURCES INCLUDED IN THE EGI SERVICE LEVEL AGREEMENT

With this agreement the cloud provider will also guarantee the availability of the platform for a maximum of 90 days from the end of the agreement, or until the final PolicyCLOUD review takes place (with no additional costs). In order to review the terms and conditions for a possible additional extension, a check-point will take place 3 months before the end of this agreement (September-October 2022).

Details about the agreements can be found in the EGI Document Repository [5].

In the coming months EGI.eu will:

- Monitor the provisioning of resources from the cloud provider,
- Produce Service Performance Reports on regular basis in order to measure the fulfilment of the agreed service level targets, and
- Perform Customer Satisfaction Review process to review the whole agreement and identify possible improvements for the agreement and services.

2.1 The INDIGO-DataCloud PaaS Orchestrator

To facilitate the deployment and the operation of a distributed Kubernetes cluster to support the project needs, an orchestrator based on the INDIGO-DataCloud [6] PaaS will also be available on the allocated resources without any additional costs. The INDIGO-DataCloud PaaS Orchestrator is the core component of the INDIGO PaaS layer.

More specifically, the PaaS Orchestrator service allows coordinating the provisioning of virtualized compute and storage resources on distributed cloud infrastructures and the deployment of dockerized services and jobs on Mesos clusters. The service comes with a set of application/service topologies ready-to-use that can be deployed through a user-friendly web interface. Users can deploy complex virtual infrastructures or docker containers in an automated way, through a user-friendly web interface. They can monitor the deployment state and get the relevant endpoints to access the deployed services, once the deployment is complete. The high-level architecture of the INDIGO-DataCloud PaaS Orchestrator is shown in Figure 1.

[2] Excluding VAT (reverse charging)

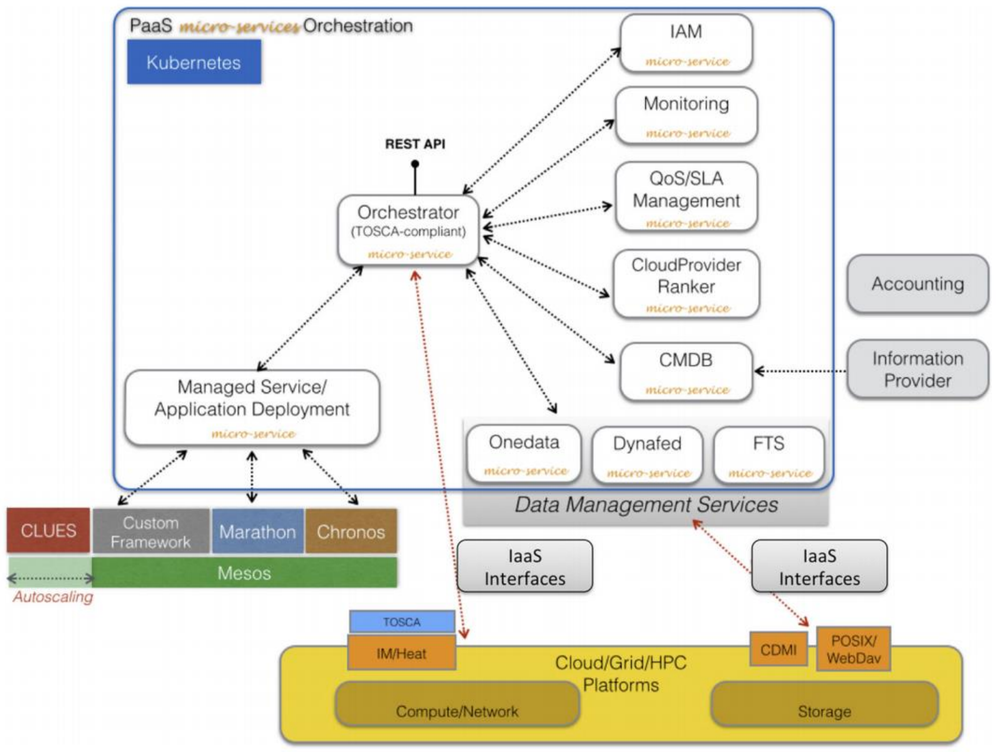


FIGURE 1 - ARCHITECTURE OF THE INDIGO-DATACLOUD PAAS ORCHESTRATOR

The Orchestrator receives the user deployment request in the form of a TOSCA template. Then it collects all the information needed to select the best provider where the virtual infrastructure (i.e. the cluster of VMs) or the dockerized service/job will be deployed. To this purpose, the Orchestrator relies on a set of auxiliary services:

- The SLAM Service that provides the prioritized list of SLAs per user/group;
- The Configuration Management DB (CMDB), that publishes the capabilities of the underlying IaaS service platforms;
- The Data Management Services that provide the status of the data files and storage resources needed by the user service/application;
- The Monitoring Service, that provides information about the availability of the IaaS services and their metrics;
- The CloudProviderRanker Service (Rule Engine), that is in charge of computing the ranking of the providers, based on some criteria, like the SLAs targets and the monitoring metrics.

This complex workflow is managed by the PaaS Orchestrator that implements the architecture described in Section 2.2. Instructions to access the INDIGO-DataCloud PaaS Orchestrator have been provided as Appendix in D3.1 - Cloud Infrastructure Incentives Management and Data Governance: Design and Open Specification 1 [1].

2.1.1 Main Features of the INDIGO-DataCloud PaaS Orchestrator

The main features of the INDIGO-DataCloud PaaS orchestrator are described below:

- Support the provisioning and automated configuration of VMs on different Cloud Management Frameworks: private clouds (e.g. Openstack, OpenNebula) and public clouds (e.g. Amazon, Azure).
- Support the deployment of containers (both long running services and batch-like jobs) on Mesos clusters.

- Support the exploitation of specialized hardware (like GPUs and Infiniband): for example, the user can request to deploy a VM or a docker container with one or more GPUs: the Orchestrator will automatically discover the providers where these hw resources are available and will schedule the deployment there.
- Support the submission of jobs on HPC facilities that expose a QCG Gateway [7].
- Support for secrets management to store sensitive information safely:
    - the Orchestrator and its web dashboard support the integration with Hashicorp Vault, a Secret Manager, in order to store sensitive data, e.g. credentials for public clouds or ssh keys.
- Support the data-aware scheduling which allows the processing of jobs close to the data.
- Support the integration with Rucio [8], a Data Management Policy engine (QoS and Data Life Cycle) that allows users to submit and manage data replication rules.
- Support workflows for data pre-processing at ingestion:
    - the Orchestrator can be instructed to trigger a processing job as soon as new data is available.

2.2 Baseline technologies and tools

During the INDIGO-DataCloud project, this component has been extended and enhanced to support the specific microservices building the INDIGO PaaS Layer. It delegates the actual deployment of resources to IM, OpenStack Heat or Mesos frameworks based on templates written in TOSCA YAML Simple Profile v1.0 [9] and the ranked list of sites (see Figure 2).

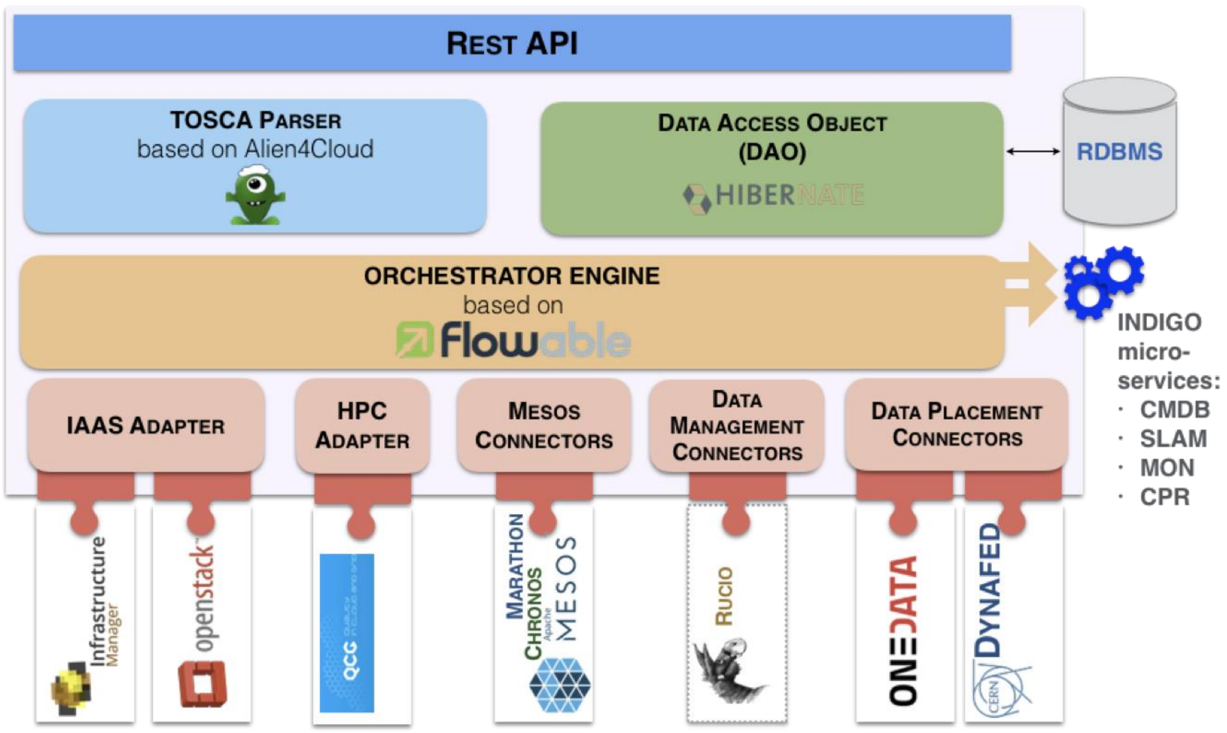


FIGURE 2 - ARCHITECTURE OF THE ORCHESTRATOR WITHIN THE PAAS LAYER

Thanks to this important achievement, using the PaaS Orchestrator and the TOSCA templates, the end user can exploit computational resources without knowledge about the IaaS details: indeed the TOSCA standard language ensures that the same template can be used to describe a virtual cluster on different cloud sites; then the Infrastructure Manager implements the TOSCA runtime for contacting the different cloud sites through their native APIs. The provisioning and configuration of the IaaS resources is therefore accomplished in a completely transparent way for the end user. The same approach is used also for submitting dockerized applications and services to a Mesos cluster (and its frameworks Marathon and Chronos): the user can describe his request in a TOSCA template and the Orchestrator provides the TOSCA runtime for contacting the Mesos master node, submitting the request and monitoring its status on behalf of the user as detailed in the next section.

2.3 Source Code

The INDIGO-DataCloud PaaS Orchestrator is distributed under the Apache License 2.0 and its open source code is available on GitHub [10]. The level of maturity of the service is TRL 8 [11] , where the Technology Readiness Level (TRL) is the scale adopted by the EC. More specifically, the scale spans over nine levels as follows:

- TRL 1 – basic principles observed
- TRL 2 – technology concept formulated
- TRL 3 – experimental proof of concept
- TRL 4 – technology validated in lab
- TRL 5 – technology validated in relevant environment (industrially relevant environment in the case of key enabling technologies)
- TRL 6 – technology demonstrated in relevant environment (industrially relevant environment in the case of key enabling technologies)
- TRL 7 – system prototype demonstration in operational environment
- TRL 8 – system complete and qualified
- TRL 9 – actual system proven in operational environment (competitive manufacturing in the case of key enabling technologies; or in space)

2.4 Deployment Status

The development of the PaaS Orchestrator has been co-funded by several H2020 projects starting from 2015 with the INDIGO-DataCloud project [6]. From 2015 onwards, the solution has been further developed in the context of the eXtreme-DataCloud [12] and DEEP- HybridDataCloud [13] projects to support the needs of the target communities. From 2018 the PaaS layer is one of the EOSC- hub [14] common services and it's registered in the EOSC portal [15].

3 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, which support both SQL and NoSQL data stores and public and private data. Based on the specifications provided in D3.1 Cloud Infrastructure Incentives Management and Data Governance Design and Open Specification 1 [1] 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. The provided Gateway API will allow building scalable and robust APIs [16], while simplifying the interaction and data collection from various sources and providers. On top of this, the main goal of this component is to handle a request by invoking multiple microservices and aggregating the results [17]. Hence, it will enhance the design of resources and structure, add dynamic routing parameters, and develop custom authorizations logic. PolicyCLOUD's Cloud Gateway and API component will support scalability, high availability, fault tolerance, and shared state without compromising performance.

3.1 Prototype Overview

Under the scope of this Deliverable, the 1st Software Prototype of the Cloud Gateways & APIs component consists of two main sub-components/microservices. The initial design of these two specific sub-components, 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 two sub-components base their functionality on two different Use Cases according to D6.3 Use Case Scenarios Definition & Design [18] and are being described into the below subsections. Currently these sub-components do not integrate with an authentication mechanism to control access to them. The integration between Cloud Gateways & APIs component and User Authorization mechanism is one of the next steps that will be implemented during the next Deliverables and Software Prototypes, in order to ensure that all the required security standards are being met.

3.2 Main components of the prototype

3.2.1 Global Terrorism Database component (GTD Component)

The first sub-component that is being described in this Deliverable bases its functionality on Use Case 2.1.3 Scenario A as described in D6.3 Use Case Scenarios Definition & Design [18], where data coming from the Global Terrorism Database (GTD) will be used in order to visualize a heatmap that shows the frequency of occurrence of radicalization incidents in the geographic proximity of a region. The main goal of this sub-component is to obtain data from the CSV file of the GTD and to update the PolicyCLOUD's data stores. The GTD component is invoked either by asynchronous workers as background processes, or by user demand after making request to the appropriate API endpoint. The file database, which is hosted in a remote location, has a total size of 165 MB for its initial state and it is in CSV format. After the database file has been successfully been fetched via an FTP connection, it is downloaded to a local directory inside the sub-component's filesystem. Afterwards, the file is been effectively parsed, in order to maintain the overall good performance. On top of this, data are loaded in a local temporary database by using batch techniques in order to be stored in a more efficient structure that enable the much easier interaction with the database. Furthermore, the component provides a RESTful API that every user or external service can interact with the GTD component.

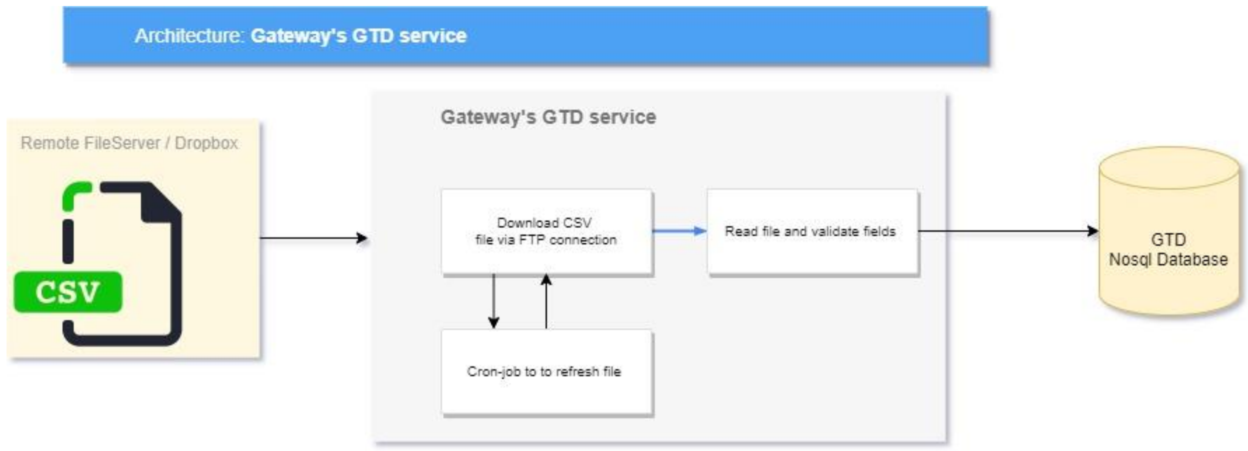

Architecture: Gateway's GTD service
Remote FileServer / Dropbox
CSV
Gateway's GTD service
Download CSV file via FTP connection
Read file and validate fields
Cron-job to to refresh file
GTD Nosql Database

FIGURE 3 - ARCHITECTURE: GATEWAY'S GTD SUB-COMPONENT

3.2.2 Twitter Connector Component

The provided sub-component bases its functionality on Use Case 2.1.3 Scenario C as described in D6.3 Use Case Scenarios Definition & Design [18], where data coming from the social media and especially from Twitter platform will be used in order to visualize a bar chart that shows the main trends linked to radicalization. The main goal of the Twitter Connector sub-component is to fetch data from the Twitter platform based on specific hashtags. In this specific use case, the corresponding sub-component filters twitter data based on the high-interest hashtag "jihad". On top of this, this sub-component relies its functionality on the Twitter API for data collection. One of the major limitations of data collection using the Twitter API is that the client machine always must keep up with the data stream and if it could not then, the stream just disconnects. Hence, in order to overcome this limitation, the initial design of this sub-component integrates also with Kafka [19] event streaming platform. One of the main benefits of using Kafka with Twitter Stream is fault tolerance. To this end, instead of having a single Python module that collects, processes, and saves everything into a JSON file, two modules are being provided for these functionalities. The first module, also called as the "Producer", collects data from the twitter stream and saves them as logs into the Kafka queue without doing any processing. In the next step another module, also called as the "Consumer", reads the logs, and processes the data, essentially creating a decoupled process. A complete pipeline and architecture of the Twitter Connector sub-component is shown in Figure 4.

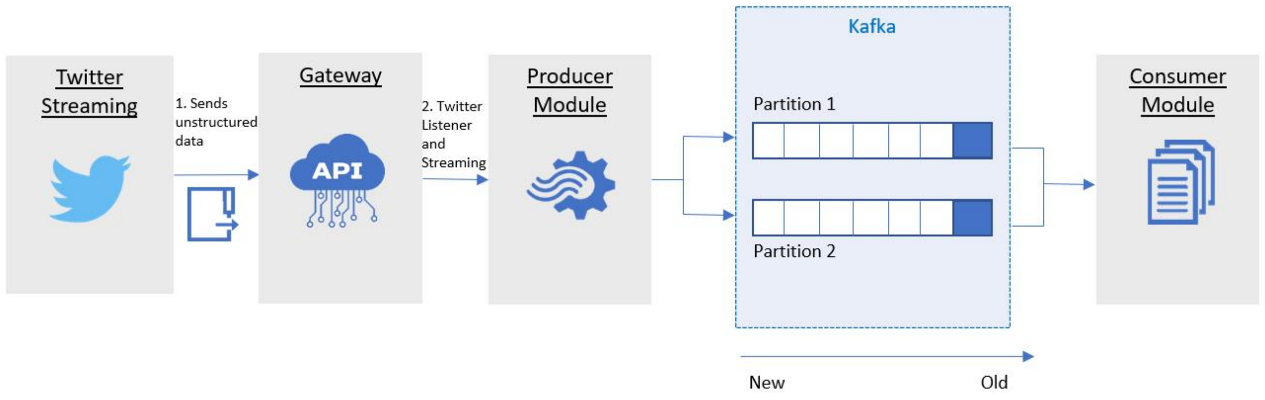


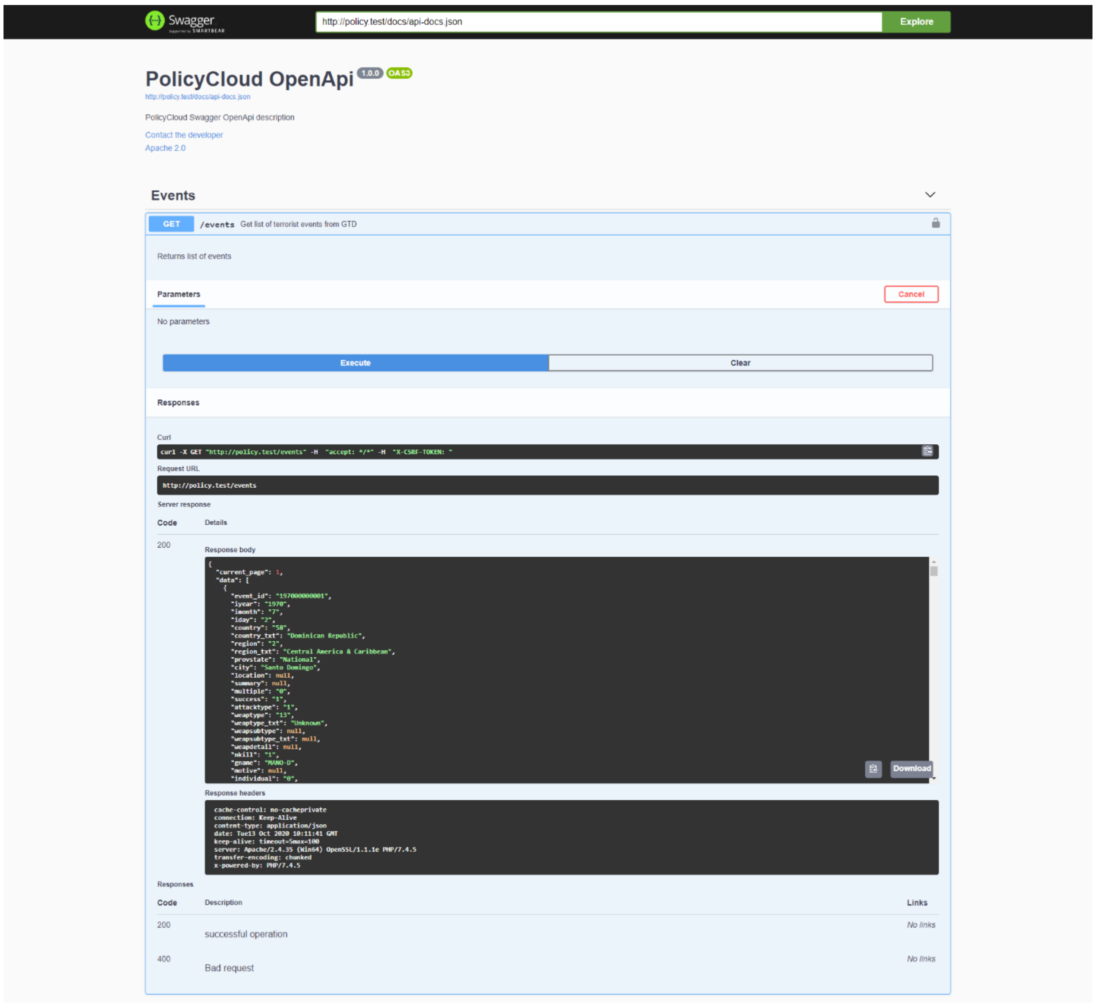
FIGURE 4 - ARCHITECTURE: GATEWAY'S TWITTER CONNECTOR SUB-COMPONENT

To this end, a decoupled twitter stream has been developed, where the initial Twitter stream is divided into two different modules. Thus, one module is responsible for retrieving data from the Twitter API and feeding them into the Kafka Cluster, the Producer Module, and one module for reading the data from the Kafka Cluster and processing the data separately, the Consumer Module. Hence, raw data from Twitter are being processed without worrying about the stream getting disconnected.

3.3 Interfaces

3.3.1 GTD component Application Programming Interface

The GTD 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 we can provide a graphical interface for interacting with the API. This makes it easier for the developer to explore all available requests and responses, which are listed including the required parameters, without the need of setting up a client on his own.



```
curl -X GET "http://policy.test/events" -H  "accept: */*" -H  "X-CSRF-TOKEN: "
```

```
http://policy.test/events
```

```
{
  "current_page": 1,
  "data": [
    {
      "event_id": "197000000001",
      "iyear": "1970",
      "imonth": "7",
      "iday": "2",
      "country": "58",
      "country_txt": "Dominican Republic",
      "region": "2",
      "region_txt": "Central America & Caribbean",
      "provstate": "National",
      "city": "Santo Domingo",
      "location": null,
      "summary": null,
      "multiple": "0",
      "success": "1",
      "attacktype": "1",
      "weaptype": "13",
      "weaptype_txt": "Unknown",
      "weapsubtype": null,
      "weapsubtype_txt": null,
      "weapdetail": null,
      "nkill": "1",
      "gname": "MANO-D",
      "motive": null,
      "individual": "0",
```

```
cache-control: no-cacheprivate
connection: Keep-Alive
content-type: application/json
date: Tue13 Oct 2020 10:11:41 GMT
keep-alive: timeout=5max=100
server: Apache/2.4.35 (Win64) OpenSSL/1.1.1e PHP/7.4.5
transfer-encoding: chunked
x-powered-by: PHP/7.4.5
```

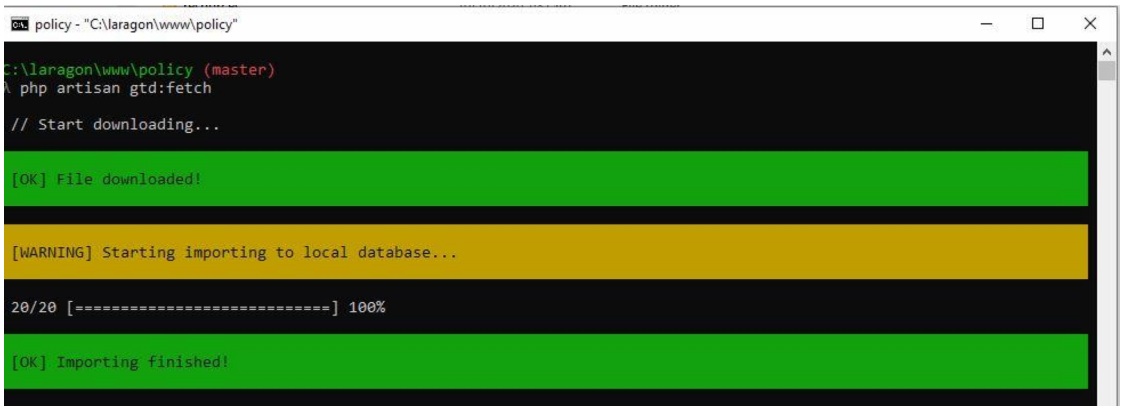
Responses

Code	Description	Links
200	successful operation	No links
400	Bad request	No links

FIGURE 5 - GTD'S SWAGGER OPENAPI INTERFACE

3.3.2 GTD component Command Line Interface

The GTD sub-component also provides a command line interface for admin users who have access to the components shell and with which it enables them to run commands in order to easily interact with different services.



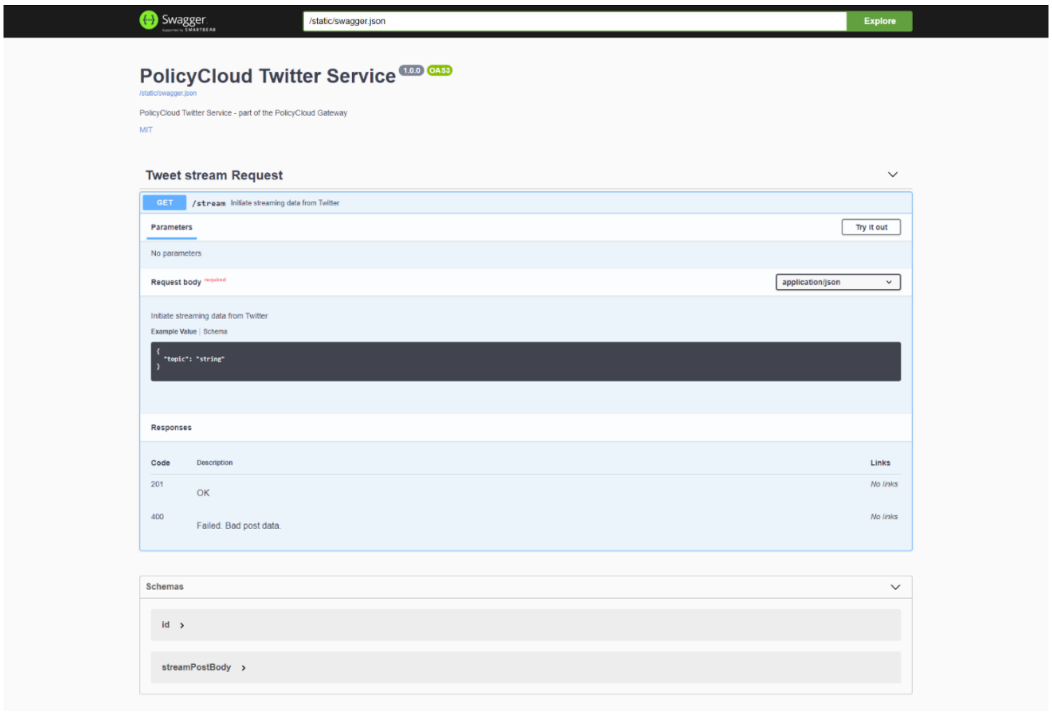
policy - "C:\laragon\www\policy"

```
:\laragon\www\policy (master)
\ php artisan gtd:fetch

// Start downloading...

[OK] File downloaded!

[WARNING] Starting importing to local database...

20/20 [============================] 100%

[OK] Importing finished!
```

FIGURE 6 - GTD'S COMMAND LINE INTERFACE

3.3.3 Twitter Connector component Application Programming Interface

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. Furthermore, the component includes an API documentation page, by using Swagger UI, so that we can provide a graphical interface for interacting with the API. This makes it easier for the developer to explore all available requests and responses, which are listed including the required parameters, without the need of setting up a client on his own. In next Deliverables a unified and concrete REST application interface will be provided for all the corresponding sub-components of the Cloud Gateways & APIs component.


Swagger /static/swagger.json Explore

PolicyCloud Twitter Service 1.0.0 OAS3
/static/swagger.json
PolicyCloud Twitter Service - part of the PolicyCloud Gateway
MIT

Tweet stream Request

GET /stream Initiate streaming data from Twitter

Parameters Try it out

No parameters

Request body required application/json

Initiate streaming data from Twitter
Example Value | Schema

```
{
  "topic": "string"
}
```

Responses

Code	Description	Links
201	OK	No links
400	Failed. Bad post data.	No links

Schemas
id
streamPostBody

FIGURE 7 - TWITTER CONNECTOR SWAGGER OPENAPI INTERFACE

3.4 Baseline technologies and tools

3.4.1 GTD component

The main component's core functionalities are written in PHP v7.4. Moreover, in order to create a robust and flexible API service, Laravel framework is being utilized. The service is running on a NGNIX server that can scale very well as it can support thousands of connections per worker process and achieve great performance. For the component's temporary database, we used MySQL 5.7 because it is working out of the box with the Laravel's Eloquent ORM, but a migration to MongoDB has already started to be implemented, considering the flexibility, scalability and performance the MongoDB provides.

3.4.2 Twitter Connector component

The implementation of the core functionalities of this component are written in Python 3.7. On top of this, Flask framework is being used in order to create a robust and flexible API service. Moreover, Kafka event streaming platform is utilized, so raw Twitter data can be processed without worrying about the stream getting disconnected.

3.5 Source code

3.5.1 GTD component Code Overview and Availability

Source code and an installation manual for the GTD component is available on this repository [20]. The installation instructions and project requirements are firmly described inside the components README.md file, but they, also, are presented and described particularly into the below subsections.

3.5.1.1 Preview

PolicyCloud GTD sub-component will be running on three separate service containers:

- The app service running PHP7.4-FPM.
- A DB service running MySQL 5.7 (Will be replaced with MongoDB in future software prototypes).
- A Nginx service that uses the app service to parse PHP code before serving the PolicyCloud GTD service to the final user.

3.5.1.2 Prerequisites

- Access to an Ubuntu 18.04 local machine or development server as a non-root user with sudo privileges.
- Docker installed on the server.
- Docker Compose installed on the server.
- A remote FTP server to upload the GTD database CSV file.

3.5.1.3 Installation Using Docker

Installation instructions based on the corresponding docker installation are being followed in this subsection.

1. Download the PolicyCloud GTD project: git clone http://snf-877903.vm.okeanos.grnet.gr/george/policycloud-gtd-service.git
2. Build the image locally: $ docker-compose build app
3. Copy the .env.example to the new .env file, run: $ cp .env.example .env
4. Open the new .env file and replace
   DB_DATABASE=your_db_name /
   DB_USERNAME=your_username /
   DB_PASSWORD=your_password .
5. In the end of the file add your ftp server credentials
   FTP_HOSTNAME=your_hostname /
   FTP_USERNAME=your_username /
   FTP_PASSWORD=your_password /
6. In the end of the .env file also include the name of the file which has uploaded to the server: GTD_FILENAME=your_filename
7. Run the environment in background: $ docker-compose up -d
8. To show information about the state of the active services, run: $ docker-compose ps
9. If all 3 services are running install composer, run: $ docker-compose exec app composer install
10. If problems occur, please check the logs: $ docker-compose logs nginx
11. To generate unique application key, run: $ docker-compose exec app php artisan key:generate
12. To run all migrations for the DB, run $ php artisan migrate

3.5.1.4 Usage

1. Open the browser and access server's domain name or IP address on port 8000: http://server_domain_or_IP:8000
2. To view all available REST endpoints in Swagger UI - OpenApi, please visit the http://server_domain_or_IP:8000

3.5.1.5 CLI TOOL

The usage of the CLI tool can be achieved either through a SSH connection into the Nginx service or anyone can run:

$ docker-compose exec app php artisan gtd:fetch , in order to refresh the database with the newest version of the Database file.

3.5.2 Twitter Connector Code Overview and Availability

Source code and an installation manual is available on this repository [21]. The installation instructions and project requirements are firmly described inside the project's README.md file, but they, also, presented and described particularly into the below subsections.

3.5.2.1 PREVIEW

PolicyCLOUD's Twitter connector sub-component will be running on two separate service containers:

- The app service running Python 3.7
- A Kafka producer container
- A Kafka consumer container

3.5.2.2 PREREQUISITES

- Access to an Ubuntu 18.04 local machine or development server as a non-root user with sudo privileges.
- Install docker-compose [22]
- Modify the `KAFKA_ADVERTISED_HOST_NAME` in docker-compose.yml to match your docker host IP (Note: Do not use localhost or 127.0.0.1 as the host ip if you want to run multiple brokers.)
- Twitter API access tokens and keys provided from https://developer.twitter.com/en

3.5.2.3 USAGE

- Inside flask-app/.env directory add the provided Twitter API credentials
- docker-compose build
- docker-compose up -d
- docker ps dirfferent conainers must be up and running
  - twitter-kafka
  - twitter-zookeeper
  - flask_dev
- Connect to flask container to start producer:
  - docker exec -it /bin/bash
- Navigate to app directory:
  - cd app
- Start Kafka producer in python:
  - Open python twitter_connector_1.py file with an editor
  - Change "bootstrap_servers='kafka'" in line 13 twitter_connector_1.py with the IP of the kafka container
  - RUN python twitter_connector_1.py
- Start Kafka consumer in python:
  - Change "bootstrap_servers='kafka'" in line 10 python cosnumer.py with the IP of the kafka container
  - RUN python consumer.py

3.6 Deployment Status

Currently the provided sub-components of the Cloud Gateways and APIs component are not deployed in a public server. These sub-components are provided through docker installations. By using Docker in local machine during the development process, different provided services of the sub-component can easily be deployed by running commands in the Dockerfile. The latter ensures also that each component will run flawlessly later in any production environment as far as Docker platform runs on this environment.

4 Incentives Management

Based on the specification provided in D3.1 [1] the effort related to incentives management will be focused on providing the following two main functionalities:

- Incentives Identification, allowing the policy maker to identify successful incentives able to attract and engage citizens who will participate in the policy making life cycle.
- Incentives Management, allowing the policy maker to manage the identified incentives and the crowdsourced data resulting from the participative external activity.

As the effort is still in the functional definition, there is not currently contribution in the form of demonstrator for this component, that will be provided in the next iterations.

4.1 Main Capabilities

4.1.1 Incentives Identification

For the time being the work performed in the context of Incentives Identification has been done towards a manual exploration of possible participants motivation through a set of surveys shared between the use cases. As main results, Table 2 summarizes a first attempt to identify incentives which would motivate participants to take part of the process.

Use Case	Motivations	HOW TO
MAG	Voluntarism Shaping a better future for the society Lobbying	Improvement in the public services Improvement in policy implemented Increased awareness in the society
SARGA	Greater commitment and participation of users with the policies Voluntarism Shaping a better future for the society Lobbying	Facilitate Government of Aragon and wine producers the decision-making process regarding agri-food promotion campaigns Involvement of producers and end users in the policies of the government of Aragon
SOFIA	Established partnerships with the municipality for collaboration Voluntarism Shaping a better future for the society Lobbying	Improvement in the public services or policies implemented related to urban environment/ clean air/ transport / etc..

TABLE 2 - INCENTIVES PER USE CASE

4.1.2 Incentives Management

Regarding the management of incentives the following functionality has been extracted from the use cases:

- Declaration of Incentives. The policy maker should be able to register new incentives with a set of attributes (incentives type, affected group, affected policies, actions type, participants result). Where:

    - Actions Type:
        - evaluate existing policies
        - provide feedback on existing priorities and KPIs
        - propose new policies to be considered
        - propose new focus areas that might be interesting to be investigated
- Crowdsource Data Ingestion and Summary. The policy maker should be able to visualize the repercussion of the applied incentive by means of a summary statistic from the crowdsourced data. For example, the policy maker would explore information such as the number of participants, the group to which the participants belong, period data of participation, type of performed actions,…)

The participation task itself will be performed outside the PolicyCLOUD system, since the provision of a crowdsourcing tool is out of the scope of the task.

As future improvements, a set of analyses performed over the crowdsourced data could be provided to the policymaker. These new statistics will be evaluated for future versions. I.e: the opinion mining component, from task 4.4, would be used to detect main topics in the citizen signals which would correspond to the main problems founds in the urban environment and which might help policy maker in the creation of new policies.

4.2 Next Steps

As part of the upcoming prototype, the basic functionality for incentives management will be provided. As part of this, a deepen research in the SOFIA use case will take place in order check the feasibility of coming across with a text summarization solution based on text analysis which might be helpful to the policymaker in the management of the incentives.

5 Data Governance Model and Privacy Enforcement mechanism

5.1 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 [23] 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 is not available for this version of the prototype and will not be further mentioned in this deliverable as it needs to be agreed with the various pilots. Our main focus was to create a first prototype that can be used to display the proper function of the mechanism and then integrate the refined Data Model with this prototype.

5.2 Main components of the prototype

5.2.1 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

- mvn clean install

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

- keytool -genkey -alias pdp-server -keyalg RSA -keysize 2048 -storetype PKCS12 -storepass <YOUR_KEYSTORE_PASSWORD> -dname "CN=pdp-server,OU=Information Management Unit (IMU),O=Institute of Communication and Computer Systems (ICCS),L=Athens,ST=Attika,C=GR" -ext "SAN=dns:pdp-server,dns:localhost,ip:127.0.0.1" -keystore pdp-server-keystore.p12 -startdate -1d -validity 3650
- 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:

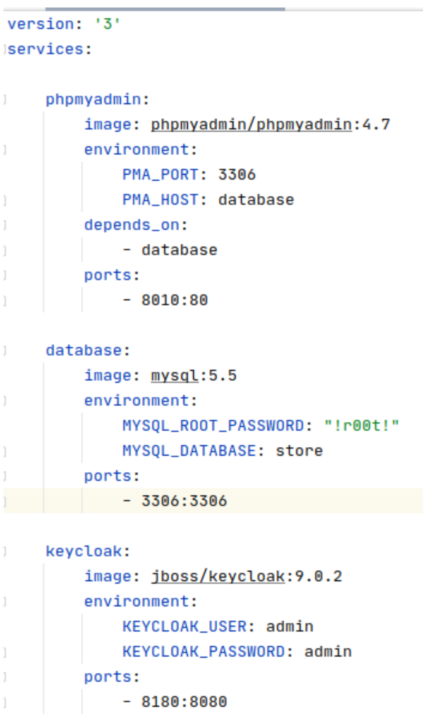
- export CONFIG_DIR=src/main/resources/config

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

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

5.2.2 KeyCloak

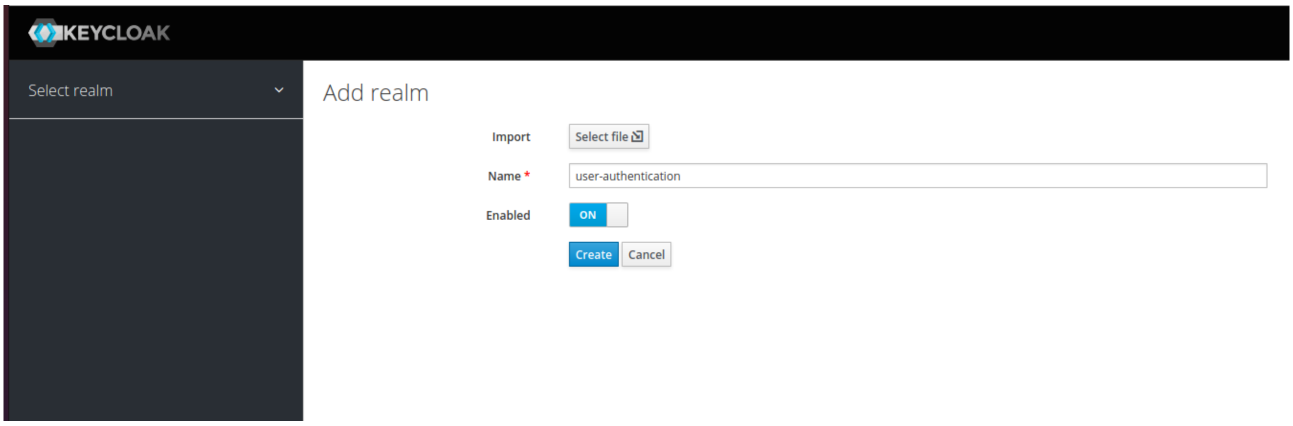
A dockerised version of Keycloak is used as the Attribute Provider and User Authenticator. The image selected is jboss/keycloak and the admin username and password, as well as the ports of the docker container are selected in the sample docker-compose.yaml provided below:



```
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 8 - DOCKER-COMPOSE.YAML FILE

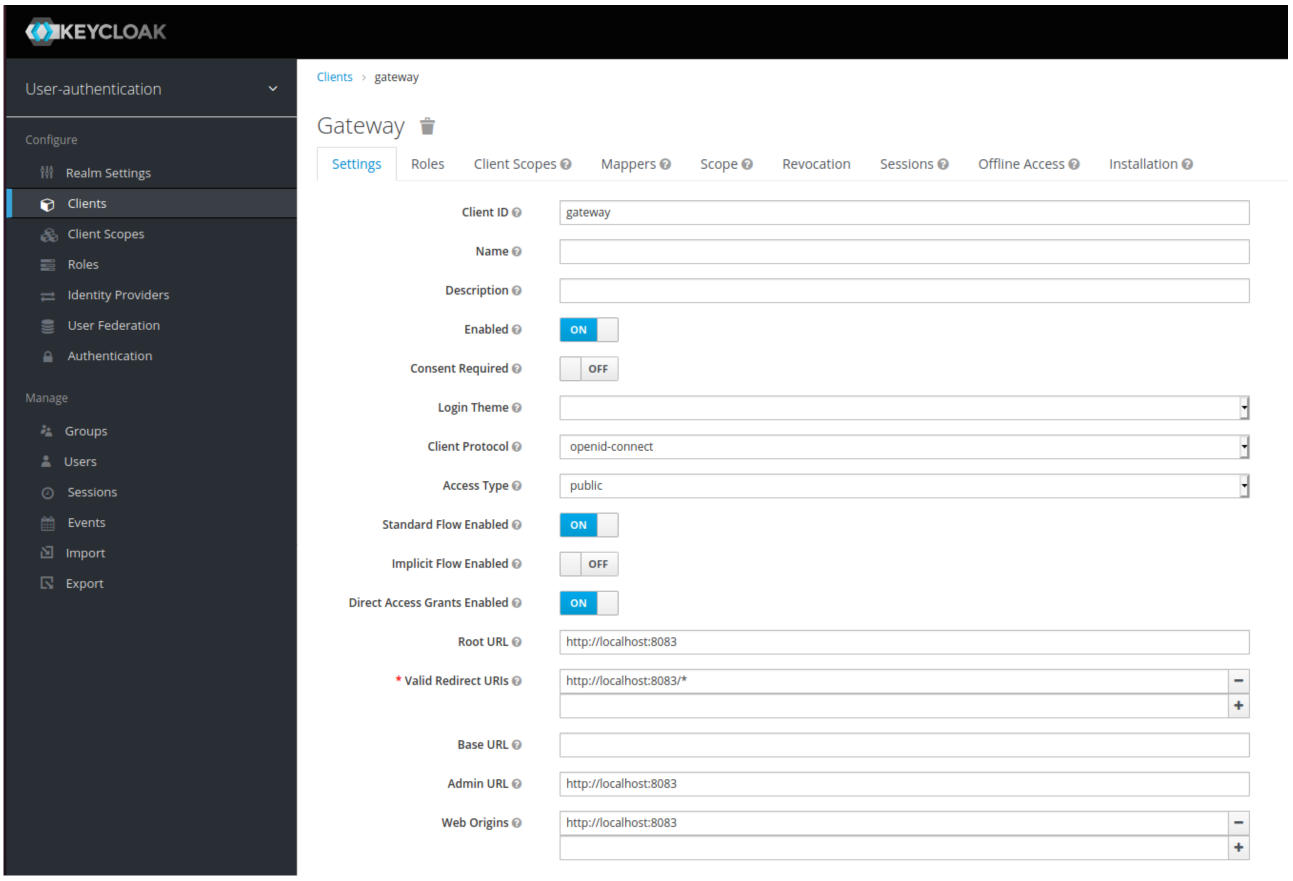
Along with Keycloak, a simple mysql database and mhpmyadmin are installed with basic settings. The Keycloak Admin Console can be accessed by visiting localhost:8180 and providing the login credentials specified in Figure 8. The necessary steps for setting up Keycloak to act as a User Authentication provider involve firstly creating the relevant realm. 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.



Import	Select file
Name *	user-authentication
Enabled	ON
	Create Cancel

FIGURE 9 - KEYCLOAK REALM CREATION

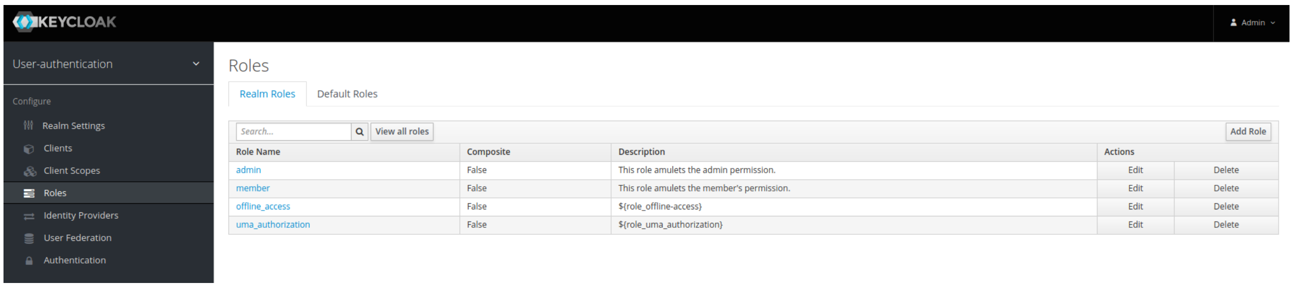
The next step is the creation of a Client inside our newly created Realm. As mentioned before, it is also important here that the new Client is named appropriately compared to the "resource" attribute in the application.yml of the service we are trying to secure.



Client ID	gateway
Name	
Description	
Enabled	ON
Consent Required	OFF
Login Theme	
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/*
Base URL	
Admin URL	http://localhost:8083
Web Origins	http://localhost:8083

FIGURE 10 - KEYCLOAK CLIENT CREATION

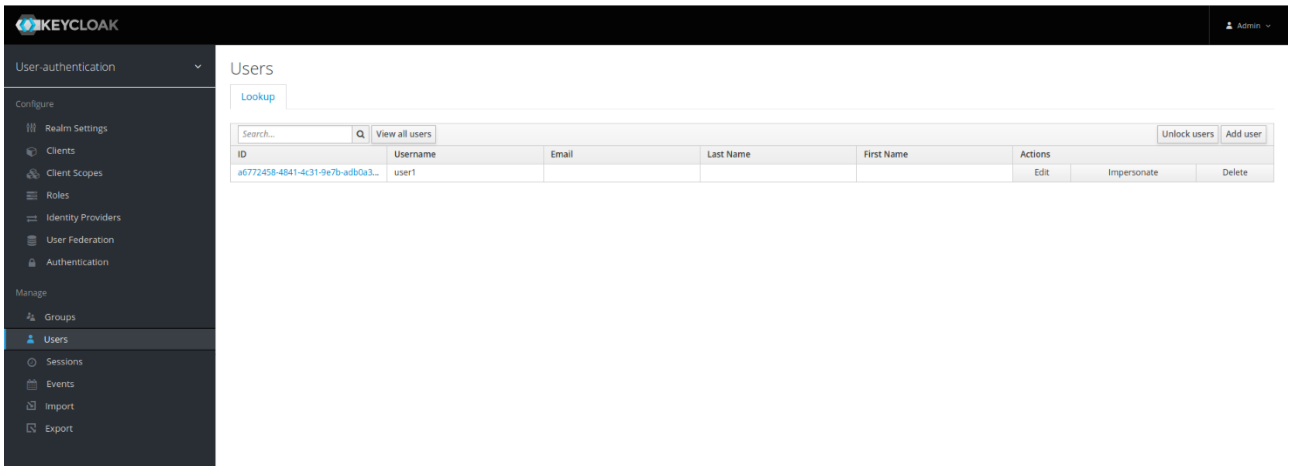
It is important to ensure that "Standard Flow Enabled" is selected as this enables standard OpenID Connect redirect-based authentication with authorization code. More specifically, in terms of OpenID Connect or OAuth2 specifications, this enables support of "Authorization Code Flow" [24] for this client. 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 11.



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 11 - KEYCLOAK REALM ROLES

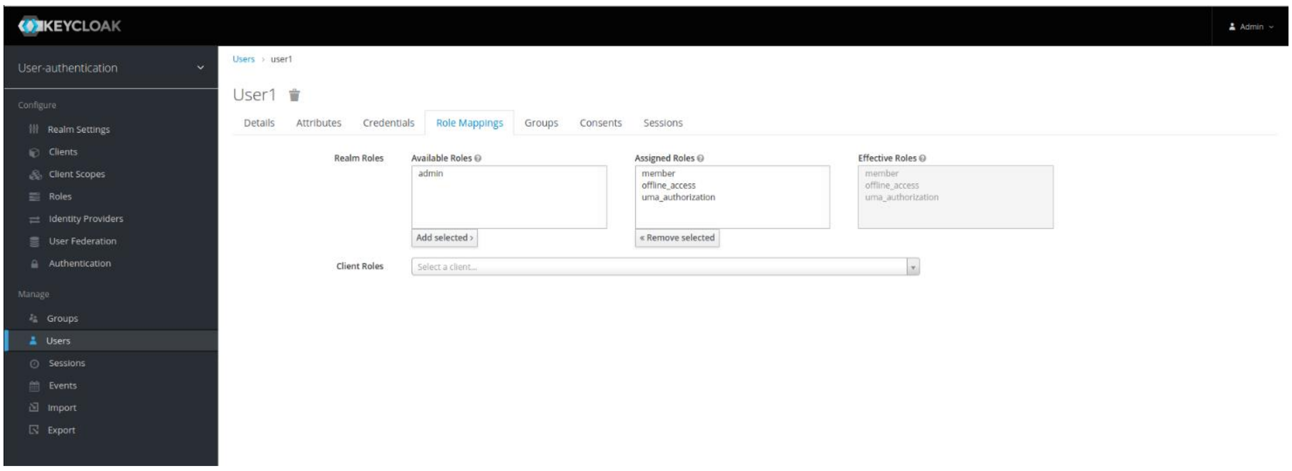
User creation can also be handled by the Admin Console and for the purpose of this prototype a sample user with username "user1" is created.



ID	Username	Email	Last Name	First Name	Actions		
a6772458-4841-4c31-9e7b-adb0a3...	user1				Edit	Impersonate	Delete

FIGURE 12 - KEYCLOAK USERS

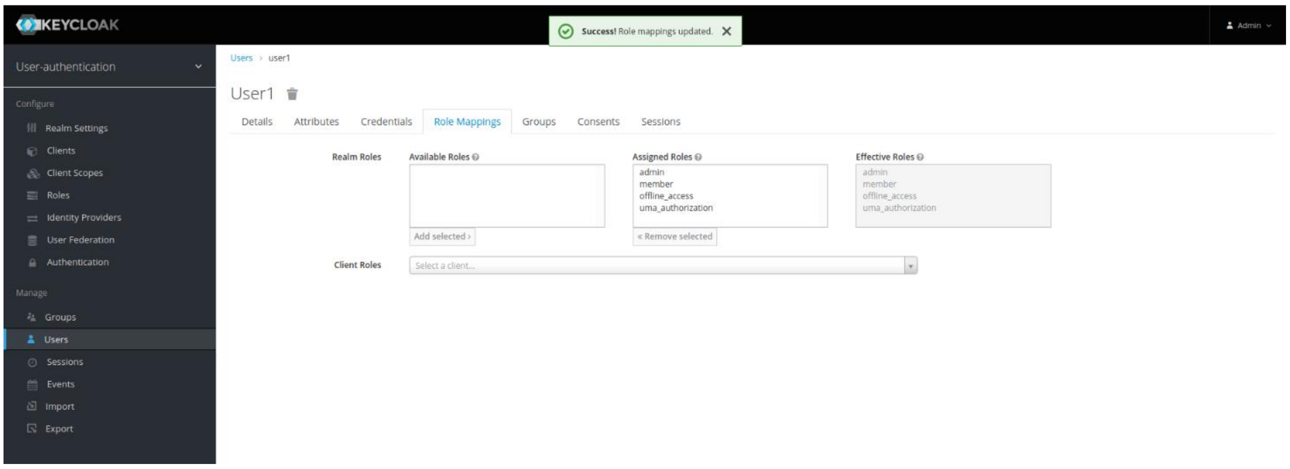
At first the newly created user is not assigned the admin role but is given the member, offline_access and uma_authorization roles. This default behaviour can be changed through Keycloak settings accordingly for newly created or registered users.



	Available Roles	Assigned Roles	Effective Roles
Realm Roles	admin	member offline_access uma_authorization	member offline_access uma_authorization
Client Roles	Select a client...		

FIGURE 13 - INITIAL ROLE MAPPINGS

As shown in Figure 13, 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.



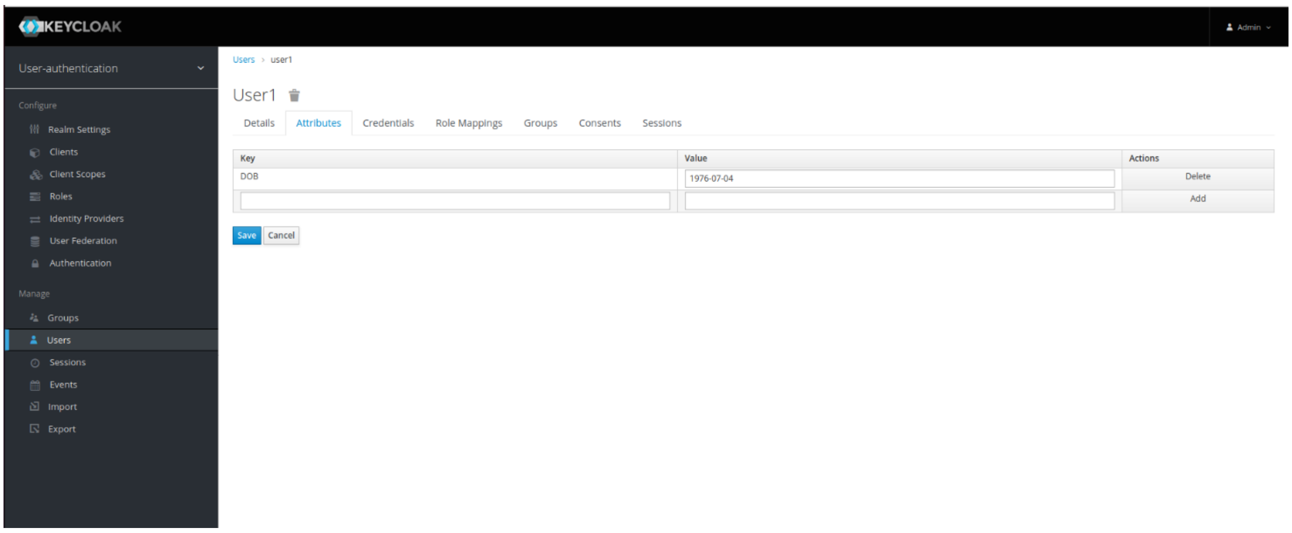
Success! Role mappings updated.

Realm Roles	Available Roles	Assigned Roles	Effective Roles
		admin member offline_access uma_authorization	admin member offline_access uma_authorization

Client Roles: Select a client...

FIGURE 14 – ADMIN ROLE GRANT

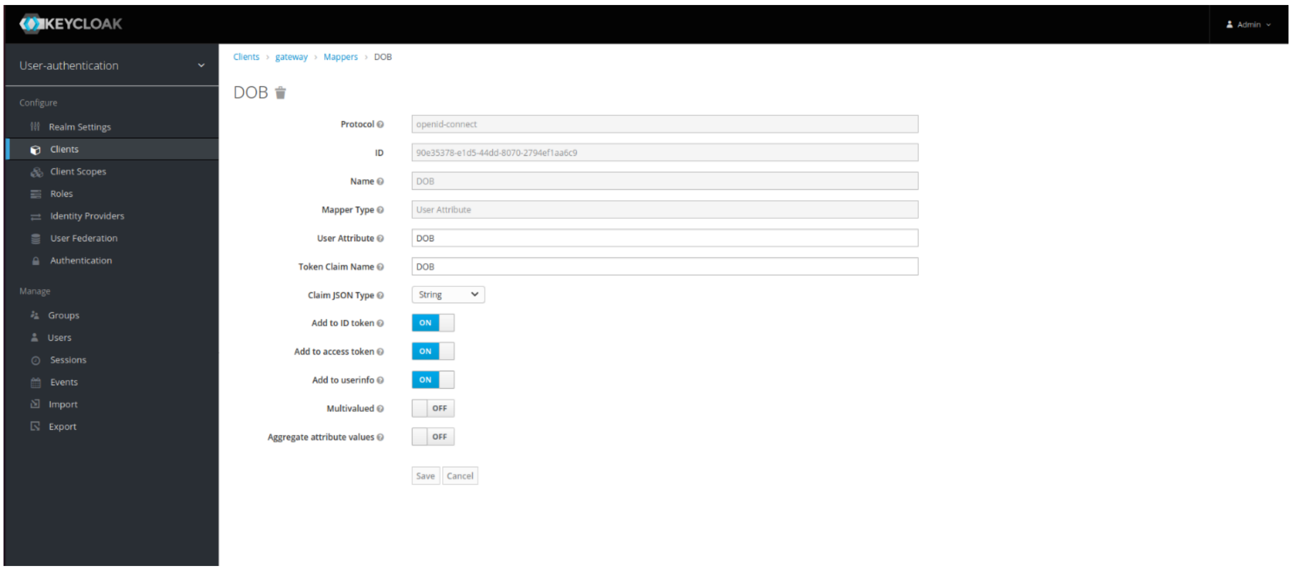
Figure 14 presents an example of the admin Role assignment to our test user. Information about users in Keycloak is not restricted to Roles and Credentials, as each User can contain custom Attributes to further define their right in the PolicyCLOUD ecosystem. Figure 15 shows such an example with the Date of Birth of the user added as a custom Attribute from the relevant tab.



Key	Value	Actions
DOB	1976-07-04	Delete
		Add

FIGURE 15 - 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 16.



DOB

Protocol	openid-connect
ID	90e35378-e1d5-44dd-8070-2794ef1aa6c9
Name	DOB
Mapper Type	User Attribute
User Attribute	DOB
Token Claim Name	DOB
Claim JSON Type	String
Add to ID token	ON
Add to access token	ON
Add to userinfo	ON
Multivalued	OFF
Aggregate attribute values	OFF

FIGURE 16 - 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 15, 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.

5.2.3 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 a filter for the access to a specific API or website and restricts access until it receives authorization by 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=asclepios
AZ_SERVER_ENDPOINTS=https://localhost:7071/checkJsonAccessRequest
AZ_SERVER_ACCESS_KEY=7235687126587231675321756752657236156321765723
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:

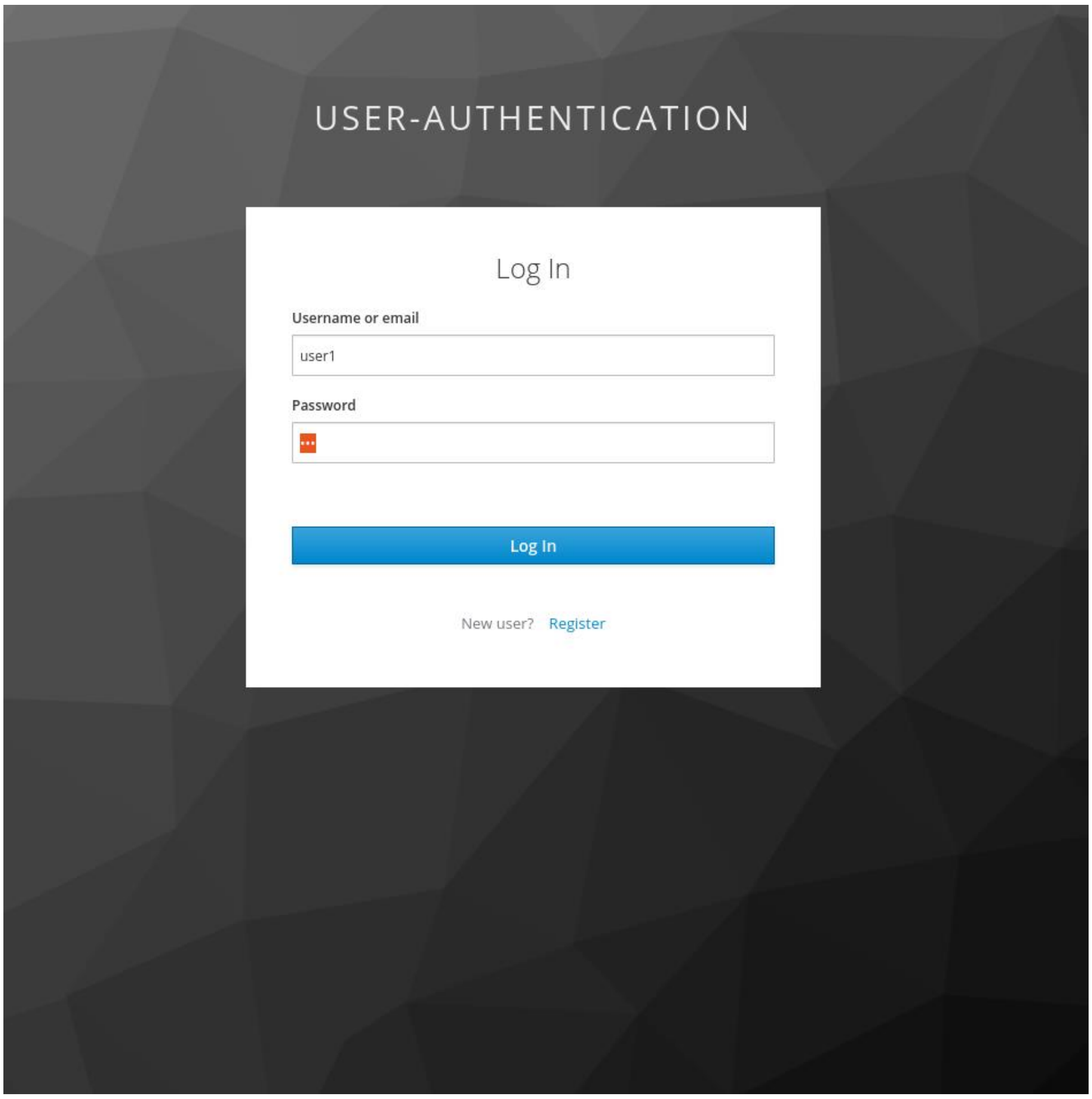
Create a long, difficult to guess (random) string. We suggest more than 32 characters long, including any combination of capital and plain letters, numbers and symbols. Avoid using phrases or values that can be guessed or extracted from context.

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.

5.2.4 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 to intercept any access to the secured API's and instead request a Keycloak login. 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 on Figure 17.


USER-AUTHENTICATION

Log In

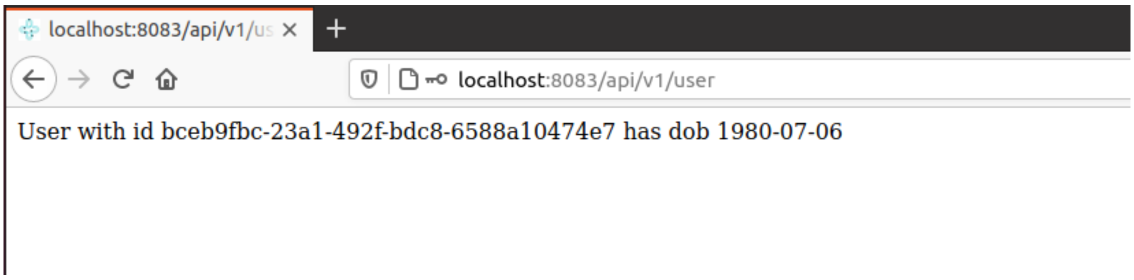
Username or email

user1

Password

FIGURE 17 - 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 on Figure 18.



```
User with id bceb9fbc-23a1-492f-bdc8-6588a10474e7 has dob 1980-07-06
```

FIGURE 18 - SUCCESSFUL ATTRIBUTES RETRIEVAL

5.3 Baseline technologies and tools

5.3.1 Balana

Balana [25] was the first open-source reference implementation of the XACML protocol and is a widely adopted solution. It supports the entire lifecycle of authorization processing. It is tightly integrated into the WSO2 Identity Server [26]. Balana, as XACML engine of the WSO2 Identity Server has two major components, the Policy Administration Point (PAP) and Policy Decision Point (PDP). Figure 19 - Balana PDP presented the component architecture of the PDP that is our main interest.

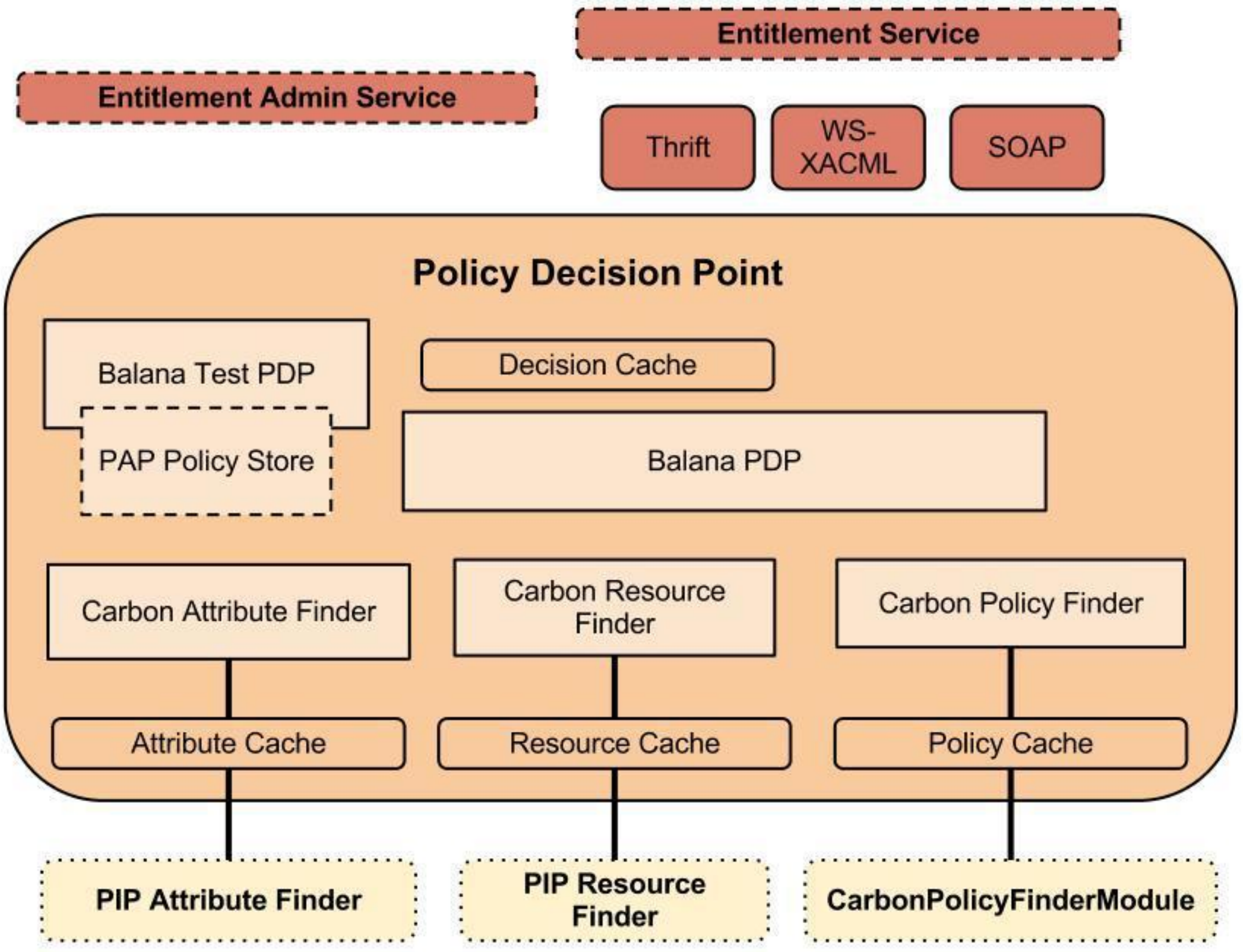


FIGURE 19 - BALANA PDP

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 [27]
- 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 20 presents a high-level diagram of the usage of the carbon policy filter for the collection of the policies to be evaluated.

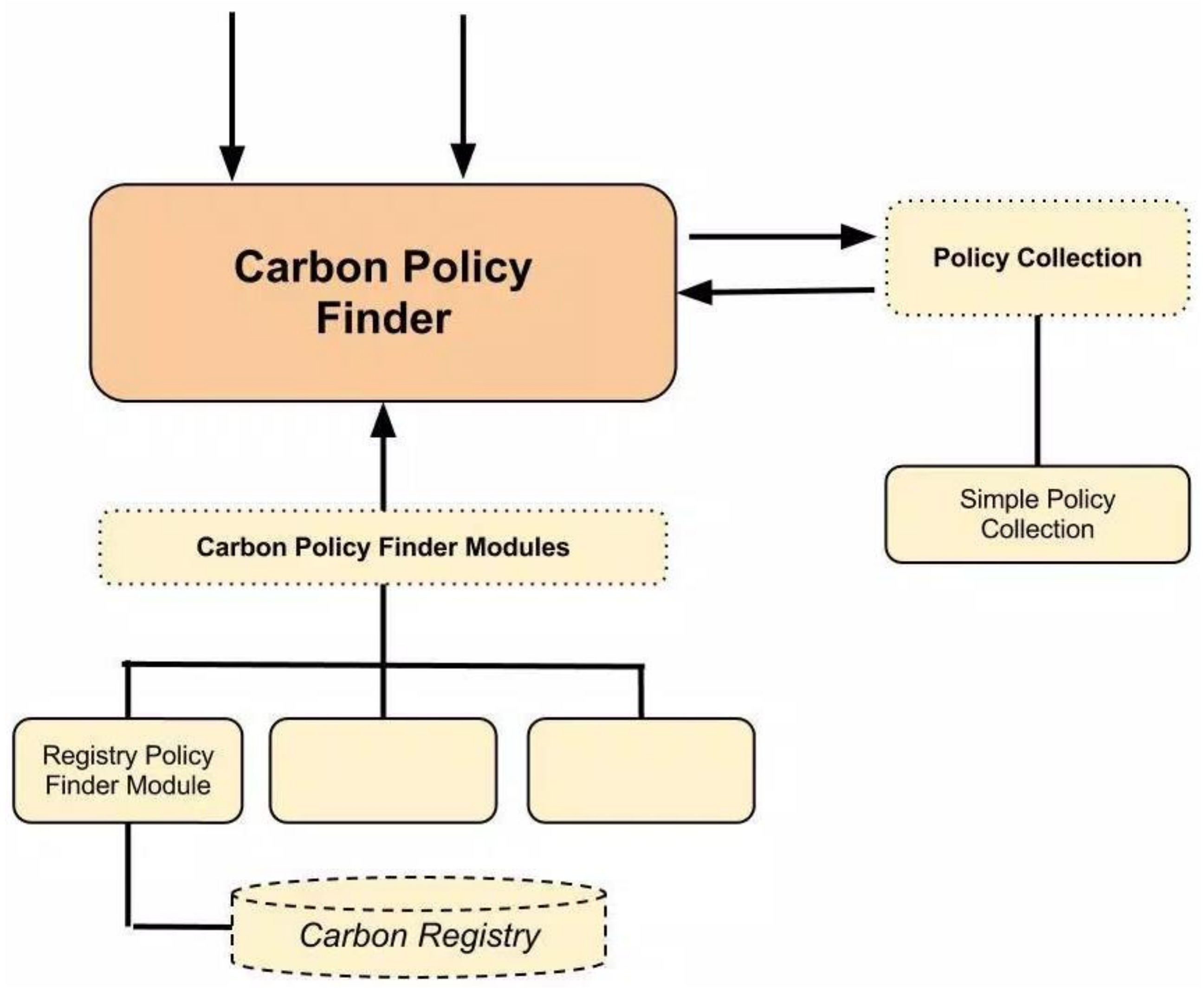


FIGURE 20 - CARBON POLICY FILTER

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 the 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 21 provides a high-level diagram for both the Carbon attribute finder and resource finders.

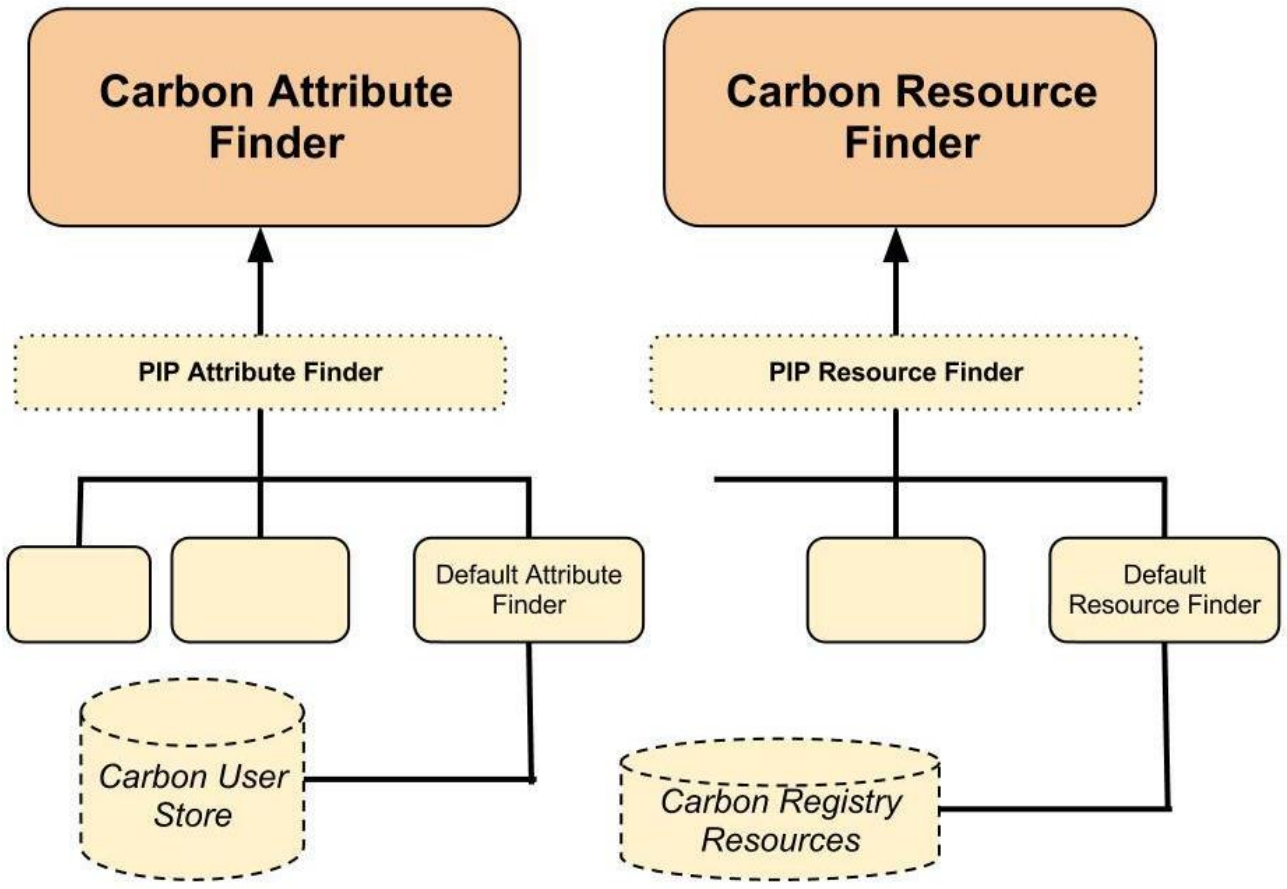


FIGURE 21 - CARBON ATTRIBUTE FINDER

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 [28].

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.

In general, we consider Balana a highly extensive open source solution, suitable for the needs of PolicyCLOUD.

5.3.2 Keycloak

Keycloak [23] 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 [29] and OpenIDConnect [30](a.k.a. OIDC). The OpenIDConnect signalling is presented in Figure 22.

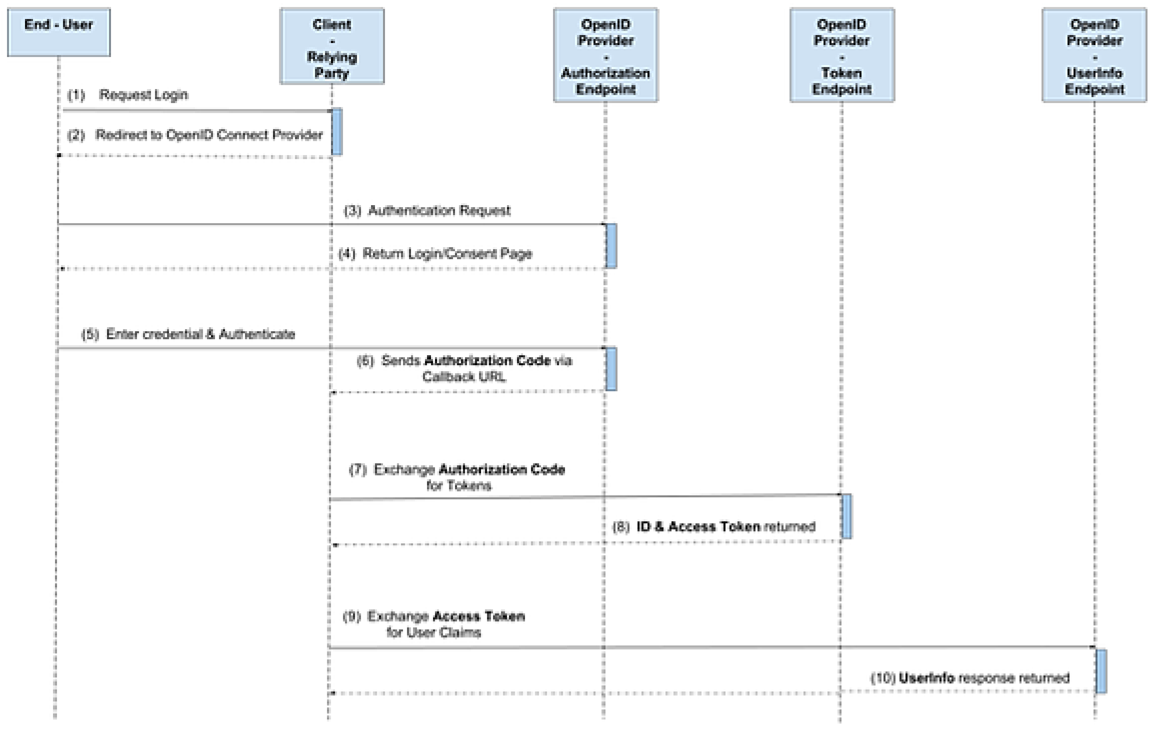

End - User
Client - Relying Party
OpenID Provider - Authorization Endpoint
OpenID Provider - Token Endpoint
OpenID Provider - UserInfo Endpoint
(1) Request Login
(2) Redirect to OpenID Connect Provider
(3) Authentication Request
(4) Return Login/Consent Page
(5) Enter credential & Authenticate
(6) Sends Authorization Code via Callback URL
(7) Exchange Authorization Code for Tokens
(8) ID & Access Token returned
(9) Exchange Access Token for User Claims
(10) UserInfo response returned

FIGURE 22 - 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.**

5.4 Deployment Status

Currently the ABAC Server and Client are deployed locally. A sample Keycloak Server is hosted in Openstack cloud provided by Ubitech. In the coming months, a dockerised version of all the ABAC components will be provided.

6 Conclusion

In this document the progress in the technical work of the tasks T3.1, T3.3, T3.4, and T3.6 until M10 of the project was presented. At first, we described the process of provisioning the PolicyCLOUD infrastructure that will be supported by the RECAS-BARI. EGI is managing the whole process and has defined the process to be used to utilize this cloud through the INDIGO-DataCloud PaaS Orchestrator. In the coming months the work related to the cloud infrastructure will focus on monitoring the cloud resources provisioning, the implementation of customer satisfaction review process to identify possible improvements, and on the creation of Service Performance Reports on regular basis in order to measure the fulfilment of the agreed service level targets.

On this document it was also reported the status of the components called cloud gateways and are responsible to obtain data from heterogenous data sources; gateways for twitter and the global terrorism database have been provided. The integration between Cloud Gateways & APIs component and the user authorization mechanism is one of the next steps that will be implemented during the next deliverables and Software Prototypes, in order to ensure that all the required security standards are being met.

Regarding the incentives identification and management, as the effort is still in the functional definition, there is not currently contribution in the form of demonstrator for this component, that will be provided in the next iterations of this document (D3.5 and D3.8). However, an analysis of the components and preliminary input by the use cases has been collected and documented. As part of the upcoming prototype, the basic functionality for incentives management will be provided, and a deepen research in the SOFIA use case will take place.

Finally, in section 5, it was provided the components and technologies that are used to provide an ABAC based access control mechanism suitable for PolicyCLOUD. This first prototype will be used to present the capabilities of an access control to the use cases, collect feedback and proceed with the definition of a proper model to be used. The development of these mechanisms is also tightly connected to the actual integration of the platform and the authentication mechanism that will be used in it. These updates will be mostly reflected in the final version of the software prototype.

The current version of the deliverable was the first version of the software prototype; the prototypes will be further updated, and the updates will be documented in the upcoming deliverables (D3.4 and D3.7).

7 References

[1] PolicyCLOUD, D3.1 Cloud Infrastructure Incentives Management and Data Governance Design and Open Specification 1, 2020.

[2] RECAS-BARI, https://www.recas-bari.it/index.php/en/.

[3] E. Federation, https://www.egi.eu/federation/.

[4] I. P. Orchestrator, https://indigo-dc.gitbook.io/indigo-paas-orchestrator/.

[5] E. V. SLA, https://documents.egi.eu/public/ShowDocument?docid=3667.

[6] I. D. Project, https://cordis.europa.eu/project/id/653549.

[7] Q. Gateway, https://apps.man.poznan.pl/trac/qcg-computing.

[8] Rucio, https://rucio.cern.ch/.

[9] T. Y. S. Profile, http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html.

[10] I. O. Repository, https://github.com/indigo-dc/orchestrator.

[11] T. R. Level, https://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2018-2020/annexes/h2020-wp1820-annex-g-trl_en.pdf.

[12] e.-D. project, http://www.extreme-datacloud.eu/.

[13] D.-H. project, https://deep-hybrid-datacloud.eu/.

[14] E.-h. project, https://eosc-hub.eu/.

[15] E. P. PaaS Orchestrator, https://marketplace.eosc-portal.eu/services/paas-orchestrator.

[16] C. e. a. Rodríguez, REST APIs: a large-scale analysis of compliance with principles and best practices., International conference on web engineering. Springer, Cham, 2016.

[17] H. a. H. K. Chawla, Implementing Microservices. Building Microservices Applications on Microsoft Azure., Apress, Berkeley, CA, 21-41, 2019.

[18] PolicyCLOUD, D6.3 Use Case Scenarios Definition & Design, 2020.

[19] A. Kafka, https://kafka.apache.org/.

[20] G. Component, http://snf-877903.vm.okeanos.grnet.gr/george/policycloud-gtd-service.git.

[21] T. Connector, http://snf-877903.vm.okeanos.grnet.gr/george/policycloud-twitter-service.

[22] D.-c. install, https://docs.docker.com/compose/install/.

[23] KeyCloak, https://keycloak.org.

[24] A. C. Flow, https://auth0.com/docs/flows/authorization-code-flow.

[25] Balana, https://github.com/wso2/balana.

[26] W. I. Server, https://wso2.com/identity-and-access-management/.

[27] A. Thrift, https://thrift.apache.org/.

[28] ApacheDS, https://directory.apache.org/apacheds/.

[29] O. 2.0, https://oauth.net/2/.

[30] OpenIDConnect, https://openid.net/connect/.

[31] J. Doe, "The standard of standards," *Standardised quotes from standards,* pp. 10-14, 2018.

[32] F. Dean, How to Write Bibliographies, Brussels: Adventure Works Press, 2006.