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

D3.4 CLOUD INFRASTRUCTURE INCENTIVES MANAGEMENT AND DATA GOVERNANCE: DESIGN AND OPEN SPECIFICATION 2

Dissemination Level	PU
Due Date of Deliverable	31/08/2021, Month 20
Actual Submission Date	31/08/2021
Work Package	WP3 Cloud Infrastructures Utilization & Data Governance
Task	T3.1, T3.2, T3.3, T3.4, T3.6
Type	Report
Approval Status	
Version	V1.0
Number of Pages	p.1 – p.44

Abstract: This report will provide the second version of the design of the cloud gateways and the cloud provisioning approaches to ensure utilization of cloud resources as required for the PolicyCLOUD environment. It will also provide the algorithms implementing the incentives management as well as the data governance model and the specification of the tools used to ensure compliance to the model across the complete data path.

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	25/05/2021	ToC	K. Oikonomou
0.2	22/06/2021	Section 3 content, first draft	V. Ardizzone
0.3	24/06/2021	Contribution on legal and ethical aspects	A. Bettiol M. Taborda Barata
0.4	13/07/2021	Section 4 content. First draft	George Manias
0.5	16/07/2021	Section 2 content	Giuseppe La Rocca
0.6	30/07/2021	Section 5 content	Maria Angeles Sanguino, Ana Luiza Pontual, Miquel Milà, Ricard Munné
0.7	05/08/2021	Section 6 ready, first draft. First complete version of the document	Giannis Ledakis, K.Oikonomou
0.8	24/08/2021	Unification of content. Minor updates in the content by all partners. Ready for review	Giannis Ledakis
0.9	27/08/2021	Wording fixes throughout the document, added references to D3.3 where relevant.	M. Taborda Barata
0.9.1	27/08/2021	Review by UPRC	Thanos Kiourtis
0.9.2	30/08/2021	Review by SARGA	Javier Sancho
0.9.3	31/08/2021	QC Review	Argyro Mavrogiorgou
1.0	31/08/2021	Ready for submission	Giannis Ledakis

Author List

Organisation	Name
ATOS	Maria Angeles Sanguino, Ana Luiza Pontual, Miquel Milà, Ricard Munné
EGI Foundation	Giuseppe La Rocca, Valeria Ardizzone
ICTLC	A. Bettiol, M. Taborda Barata
UBITECH	Giannis Ledakis, Konstantinos Oikonomou
UPRC	George Manias

Abbreviations and Acronyms

Abbreviation/Acronym	Definition
ABAC	Attribute-based access control
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CS	Crowdsourcing
DB	Database
DCs	Data Cooperatives
DPA	Data Processing Agreement
DSPs	Data Sharing Pools
EC	European Commission
EOSC	European Open Science Cloud
EU	European Union
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GDPR	Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)
GTD	Global Terrorism Database
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
IM	Incentives Management
JSON	JavaScript Object Notation
KPI	Key Performance Indicators
MIAB	Motive-Incentive-Activation-Behavior
NLP	Natural Language Processing
NoSQL	Non Structured Query Language
PaaS	Platform as a Service
PDS	Personal Data Sovereignty
PDT	Policy Development Toolkit
PDTs	Public Data Trusts
REST	Representational state transfer
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
XML	eXtensible Markup Language

Contents

List of Tables

List of Figures

Executive Summary

This document is part of WP3, and it provides an updated version of the design and specification of the tools described in deliverable "D3.1 Cloud Infrastructure Incentives Management and Data Governance: Design and Open Specification 1". As this document provides information about the work performed in tasks T3.1, T3.2, T3.3, T3.4, and T3.6 between August 2020 (M8) and August 2021 (M20) of the project, it covers information regarding a) cloud provisioning and utilization of cloud resources for PolicyCLOUD, b) the design of the cloud gateways and their APIs, c) the incentives management and identification in the scope of PolicyCLOUD, and d) the data privacy and governance mechanisms for PolicyCLOUD. This document is the second iteration of this report, with one more iteration to be provided in August 2022 (M32). For an easier understanding of the updates, we provide the updates from D3.1 in each section, and also a summary of the updates in the introduction of the document.

1 Introduction

1.1 Purpose of the document

This document is the second iteration of "Cloud Infrastructure Incentives Management and Data Governance: Design and Open Specification", covering tasks T3.1, T3.2, T3.3, T3.4, and T3.6. In the scope of *T3.1 - Cloud Provisioning of the PolicyCLOUD Infrastructure*, the focus was on the monitoring and control of the provisioned cloud resources, while T3.2 - *Registration of the Cloud-based Environment for Data-driven Policy Management in EOSC* set out a plan for onboarding the envisioned services to the EOSC Portal. In the scope of *T3.3 - Cloud Gateways & APIs for Efficient Data Utilization* and *T3.4 - Incentives Management*, the design of cloud getaways and Incentive management mechanisms, respectively, was started. Finally, for *T3.6 - Data Governance Model, Protection and Privacy Enforcement,* the goal is the creation of both the model and the mechanism used for privacy enforcement and the protection of data. For all tasks, this document describes the work performed until M20, while also providing, for each section, a summary of the updates from D3.1 (which was delivered in August 2020, M8 of the project).

1.2 Structure of the document

The document is structured as follows: Section 2 covers the provisioning process for the cloud infrastructure of PolicyCLOUD, while Section 3 provides an overview of the process for onboarding in the EOSC Portal. Section 4 presents the Cloud Gateway components. Section 5 describes the identification and management of incentives. Section 6 presents the background and the technologies to be used for the creation of the data privacy mechanism. Finally, Section 7 provides the conclusion of the document, followed by the document's references.

1.3 Summary of Changes

As mentioned already, this document extends the work presented in D3.1. Therefore, to assist the reader of the document, we collect here an overview of the content presented and changes regarding each section of the document.

- Section 2 presents the latest updates regarding the PolicyCloud infrastructure. Readers can find more information about the EGI framework, the requirements collection and the call for providers process in D3.1.
- Section 3 – Registration of the Cloud-based Environment for Data-driven Policy Management in EOSC – is a new section, which was not present in D3.1, as the task was not active when D3.1 was written.
- Section 4 presents the updates on the design and the implementation aspects of Gateways & APIs for Efficient Data Utilization.

- Section 5 extends greatly the content provided in D3.1, as it provides the relevant analysis, the functionalities and the design of the incentives management components.
- Section 6 presents the updated analysis regarding data governance, its relevance to the legal/regulatory requirements that have to be covered by the platform, and the updates on the design of the privacy enforcement mechanism.

In addition, at the beginning of each section, more details are provided.

2 Cloud Provisioning of the PolicyCLOUD Infrastructure

As reported in D3.2 - Cloud Infrastructure Incentives Management and Data Governance Software [1], in October 2020, the project signed a pay-for-use agreement[1] with a cloud resource provider (RECAS-BARI), the full capacity of which was allocated to the project.

Since October 2020, EGI.eu has been monitoring the performance of the resource provider in order to make sure that the resources and the services requested by the technical partners are being delivered according to the service targets described in the agreement. From a technical perspective, to enable the monitoring of the performance of the resource provider supporting the PolicyCLOUD Infrastructure, EGI created a specific dashboard[2] in ARGO, a lightweight service for service level monitoring designed for medium and large-sized research infrastructures.

As of the date of this document, EGI.eu has produced a first service performance report showing the monthly availability and the monthly reliability of the cloud provider from August 2020 to January 2021. As reported in Section 4 of the pay-for-use agreement, the following are the service level targets defined within the Operational Level Agreement entered into with the provider:

Monthly Availability: Defined as the ability of a service to fulfil its intended function over a calendar month.

- Minimum (as an average percentage per month):
  - Cloud Compute: 90%
    - RECAS-BARI: 90%
  - Online Storage: 90%
    - RECAS-BARI: 90%

Monthly Reliability: Defined as the ability of a service to fulfil its intended function over a calendar month, excluding scheduled maintenance periods.

- Minimum (as an average percentage per month):
  - Cloud Compute: 95%
    - RECAS-BARI: 95%

[1] https://documents.egi.eu/document/3667
[2] https://argo.egi.eu/egi/dashboard/SLA/EGI_POLICYCLOUD_SLA

- Online Storage: 95%
  - RECAS-BARI: 95%

To complement the monitoring of the cloud-based infrastructure, EGI.eu has also conducted customer satisfaction reviews on a regular basis, in order to collect feedback from the technical partners about the status of the agreement, identify suggestions for improvement and additional requests to scale-up the initial resources allocated by the provider.

2.1 Updates Since D3.1

In addition to the aforementioned updates about the monitoring of the infrastructure and the signing of the agreement, the reader can find more information about the EGI framework, the requirements collection and the call for providers process in D3.1.

2.2 Next Steps

In the coming months, EGI.eu will:

- Continue monitoring the provisioning of the resource capacity allocated to operate the PolicyCLOUD infrastructure;
- Organize customer satisfaction interviews to review the whole agreement, identify possible improvements and scale-up the pool of resources allocated by the end of the year 2021, if necessary;
- Provide technical support in order to integrate the PolicyCLOUD ABAC service, based on KeyCloak, with the EGI AAI Check-in service;
- Enhance the EGI.eu Service Level and Operational Level Agreement templates in order to include a Data Processing Agreement (DPA) to regulate the personal data processing to be carried out by the provider (acting as a processor).

3 Registration of the Cloud-based Environment for Data-driven Policy Management in EOSC

Registration of PolicyCLOUD services in EOSC portal https://eosc-portal.eu/ is a goal which requires preparation and assessment work, on the basis of a clear alignment on the approach to these services. It is a good practice to consider and include the perspective of users/consumers in the way in which the information is presented, to empower the offering and the understanding of the new solutions contained within these services.

The onboarding and registration process in the EOSC Portal follows a very straightforward procedure, based on the maturity of the services and the early registration of the providers.

EGI will approach this activity within the Consortium through the following phases:

1. Analysis of providers who are part of the Consortium and preparation for onboarding;
2. Onboarding of providers;
3. Analysis of PolicyCLOUD service maturity and gap analysis;
4. Provision of support to onboard the services, based on the following requirements:
    - The service is accessible to users outside its original community;
    - The service is described through a common template focused on value proposition and functional capabilities;
    - At least one service instance is running in a production environment available to the user community;
    - Published research data is Findable, Accessible, Interoperable and Reusable [2];
    - Release notes and sufficient documentation are available;
    - Helpdesk channels are available for support, bug reporting and requirements gathering;
5. Technical integration;
6. Assessment of any gaps.

The EOSC Portal is continually improved and optimized based on the feedback of thousands of users and providers. Its wide capability to reach the scientific community and target users can be easily verified using various statistics (see Figure 1), which are available and easily accessible on the Portal.

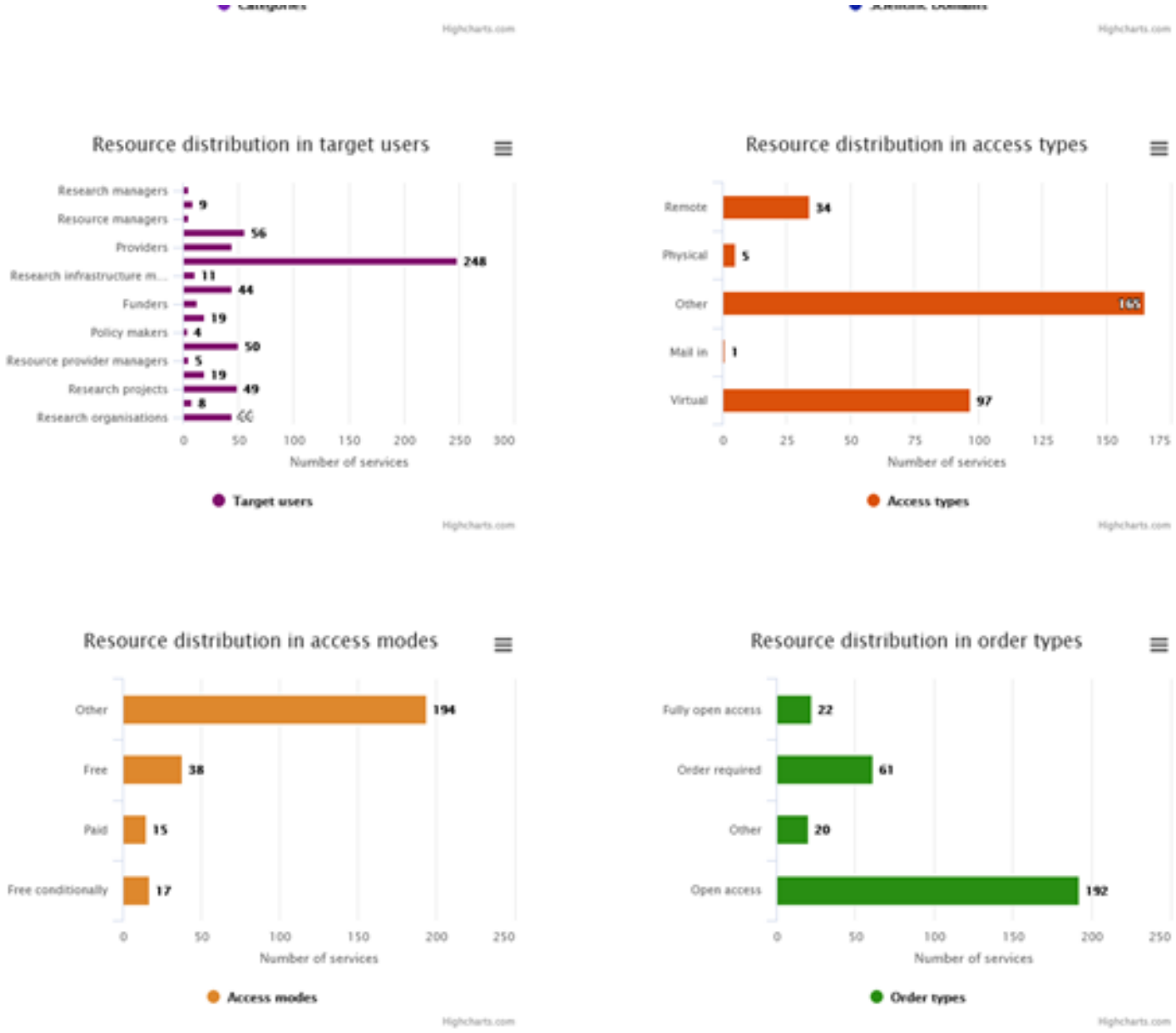

FIGURE 1 - EOSC PORTAL STATISTICS

Below we describe some examples of statistics gathered by the EOSC Portal:

- Resource (Service) distribution in target users
  (it's possible to see the distribution of a single service requested by target users)
- Resource (Service) distribution in the access type
  (it depends on the technical service access offered)
- Resource (Service) distribution in access model
  (it's related to a portal statistic on the order model (open, paid, etc.))

- Resource (Service) distribution in order types model

  (It's possible to verify, according to the Portal order types (order, fully open, open, etc.), the status of the distribution of the services)

More statistics are available at the following link: https://providers.eosc-portal.eu/stats/providers, giving helpful information depending on the type of the service offered and technical implementation as Lifecycle, Areas of Activity, Network (to which Scientific area or community the services are linked to), etc.

Years of work and the dedication of many people have resulted in the creation of a very powerful tool, with a consolidated way of working, acting as a playground for the scientific community and any users looking for outputs of research and EU projects, with a currently huge service offering from big and small providers.

Leveraging this experience and the wide capability of the EOSC Portal is a key factor for the success of PolicyCLOUD's new services and providers.

FIGURE 2 - EOSC PORTAL CAPABILITY

In terms of communication, the possibility to publish promotional articles (one per service) and to leverage EOSC communication and events offers a very large audience, e.g. 4,171 Visitors of EOSC Portal + Marketplace average/Month, to be reached quickly and with no additional effort.

In Figure 2, a graphical summary of the EOSC Portal capability is shown to remind what adding services as a Provider to EOSC can offer to:

- Choose to advertise them on the EOSC Portal and promote their adoption outside your traditional user groups, reaching a wider user base.
- Get statistics about access requests and customer feedback.
- Get a free online platform where you can manage service requests, interact with users and provide support to them, and agree the most suitable service levels.
- Allow users to authenticate with their own credentials to access your services and resources and get support to enable this.
- Contribute to the definition and maintenance of EOSC service provisioning policies and the portfolio roadmap.
- Join the group of providers that meet EOSC quality standards.

3.1 Next Steps

As summarized in the phases above, here added for simplification,

1. Analysis of providers who are part of the Consortium and preparation for onboarding;
2. Onboarding of providers;
3. Analysis of PolicyCLOUD service maturity and gap analysis;
4. Provision of support to onboard the services, based on the following requirements:
    - The service is accessible to users outside its original community;
    - The service is described through a common template focused on value proposition and functional capabilities;
    - At least one service instance is running in a production environment available to the user community;
    - Published research data is Findable, Accessible, Interoperable and Reusable [2];
    - Release notes and sufficient documentation are available;
    - Helpdesk channels are available for support, bug reporting and requirements gathering;
5. Technical integration;
6. Assessment of any gaps.

The most important activity in the next period is to perform a one-to-one analysis, of each potential Provider, and specific gap analysis with Municipalities, to understand their readiness to be onboarded in accordance with EOSC templates. This activity has already started.

EGI will support Providers and Municipalities to fill any gap and to be ready for the onboarding with proper registration.

According to the maturity of the services, each service will be registered in a second step.

4 Cloud Gateways & APIs for Efficient Data Utilization

4.1 Updates Since D3.1

As reported in D3.2 - Cloud Infrastructure Incentives Management and Data Governance Software Prototype 1[1], the first version of the Cloud Gateway & APIs component has been implemented. Moreover, the overall architecture was designed, as also depicted in the below Figure 3, and the exact technologies, tools and frameworks which were further utilized at the implementation phase were identified. In this context, two main microservices (Twitter component, GTD component) were implemented to obtain data from external data sources, along with filtering and mapping mechanisms according to the suggested schemas of the project. Furthermore, the first version of the API gateway component was implemented, providing a REST API, an Open API specification and documentation of the endpoints through the installation and utilization of Swagger UI[3]. This allows the provision of a graphical interface to interact with the API. Finally, a data encoding mechanism was implemented by using the Avro Schema[4] to feed data to PolicyCLOUD's Kafka message broker.

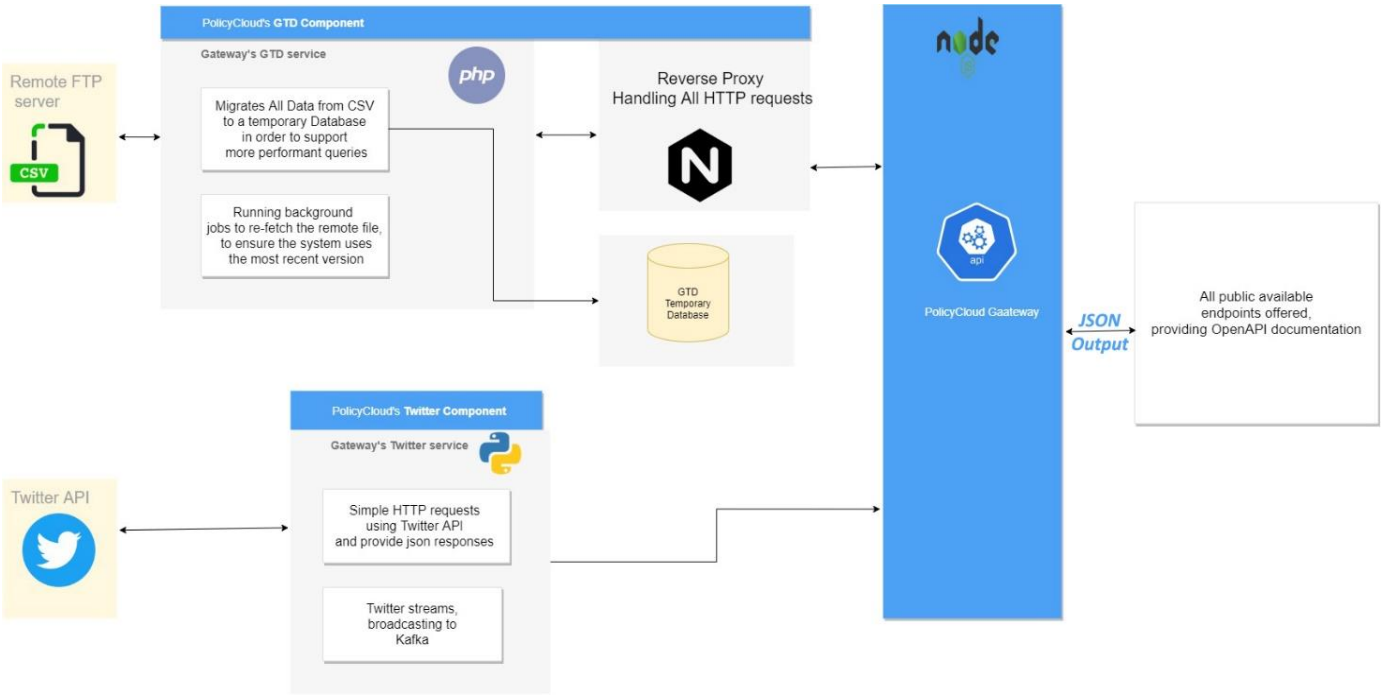

FIGURE 3 – CLOUD GATEWAYS ARCHITECTURE

[3] https://swagger.io/
[4] https://avro.apache.org/docs/current/spec.html

4.2 Cloud Gateway Capabilities

The Cloud Gateway component is the single point of connection between the cloud and the outside world. It works just as an enhanced reverse proxy with more advanced capabilities, including orchestration, security, monitoring, etc. Internally, it provides a simple and effective way to route incoming requests. Following the Gateway Pattern [3], it exposes a set of microservices to users through a unified API, instead of providing an API for each service that it includes. A few well-known API gateway implementations include Neftlix Zuul [4], the Cross-Cloud API Management Platform (Apigee) [5], Amazon Cloud Gateway [6] and the Spring Cloud Gateway [7].

4.2.1 Fetching external Resources

The Cloud Gateway will be a component that combines and integrates various microservices that will obtain data from heterogeneous data sources, such as:

- External APIs (Twitter, Facebook, Reddit, etc.);
- Files (Excel, CSV);
- Data streams from IoT devices and sensors;
- Databases (SQL, NoSQL).

As noted also within deliverable D3.3 [16], it is important that each of the above services be carefully configured in order to comply with the terms of usage (e.g., terms and conditions) applicable to each source, to ensure lawful usage of those sources, and to avoid interruptions to the data stream due to rules violation (e.g., overloading an external API despite the rate limit per request established in the applicable terms and conditions for that API).

To this end, and under the scope of D3.2 - Cloud Infrastructure Incentives Management and Data Governance Software Prototype 1, two specific microservices were implemented: the GTD Microservice and the Twitter Microservice. These two microservices are presented below.

GTD Microservice

The GTD (Global Terrorism Database) microservice bases its functionality on Scenario A of Use Case 1, as described in D6.3 [8]. In this Scenario, data sourced from the Global Terrorism Database (GTD) will be used in order to produce a heatmap which shows the frequency of occurrence of radicalization incidents in the geographic proximity of a region. The main goal of this microservice is to obtain data from the CSV file of the GTD and to update PolicyCLOUD's data stores. To this end, the GTD microservice reads the CSV file from an FTP server (used for development purposes - this will be replaced with a cloud storage solution in the future) and applies certain transformations according to schema requirements. The data are temporarily stored inside the component, and when the reading process finishes, the microservice is able to serve requests. Batch techniques ensure that large CSV files can be ingested without affecting performance.

The data refresh procedure of the component can be triggered in two possible ways: either by using the offered REST endpoint, or through a periodic scheduler that ensures the latest data are sent to PolicyCLOUD's internal database. The GTD microservice was implemented using a Laravel framework in PHP, in order to create a robust and flexible API service. The service is also running on an NGNIX server that can scale very well, as it can support thousands of connections per worker process and achieve great performance.

Twitter Microservice

The Twitter microservice bases its functionality on Scenario C of Use Case 1, as described in D6.3 [8]. In This Scenario, data sourced from social media (and especially from the Twitter platform) will be used in order to produce a bar chart which shows the main trends linked to radicalization. Its main goal is to retrieve data from Twitter and, after applying certain transformations according to the suggested schema, send those data to internal components of the PolicyCLOUD platform for further processing, cleaning and analysis. This specific microservice uses the Twitter API to provide tweet results for a given key that is encoded and sent to the Kafka broker. One of the major limitations of data collection using the Twitter API is that the client machine must always keep up with the data stream – if it is unable to do so, the stream disconnects. To overcome this limitation, the initial design of this microservice is also integrated with a Kafka[5] event streaming platform. One of the main benefits of using Kafka with a Twitter stream is fault tolerance. Instead of having a single Python module that collects, processes, and saves everything into a JSON file, two modules are provided for this. The first module, called the "Producer", collects data from the Twitter stream and saves the data as logs into the Kafka queue, without doing any processing. In the next step, another module, called the "Consumer", reads the logs and processes the data, essentially creating a decoupled process. To this end, the microservice enables data streaming by obtaining real time tweets for a pre-selected period of time, and sending them to the Kafka broker.

The Twitter microservice has been implemented through Python 3.7. On top of this, the Flask framework is being used to create a robust and flexible API service. Moreover, a Kafka event streaming platform is utilized, so that raw Twitter data can be processed without the risk of about the stream being disconnected.

4.2.2 Authentication and Security

The Gateway authentication and security mechanisms ensure that only authorized requests can get access to services and data.

[5] https://kafka.apache.org/

Integration with Keycloak, PolicyCLOUD's federated identity mechanism, was implemented in the Cloud Gateway component. To obtain access to protected resources, users have to send a token obtained from Keycloak that ensures their identity and role along with every request.

4.2.3 Data filtering

The Cloud Gateway component will provide mechanisms responsible for checking the reliability of the provided data, by filtering the obtained datasets before pushing them to the internal PolicyCLOUD services. To this end, inaccurate records or corrupt data must be removed from the datasets and incomplete data must be identified before storage inside the cloud infrastructure. Moreover, as noted also within deliverable D3.3 [16], retrieved data must be evaluated according to applicable legal requirements, to ensure the lawfulness of its further processing. In particular, sensitive and private data (such as personal data) must not be stored where this is not strictly necessary for lawful purposes for which the data source is to be used via the PolicyCLOUD services, in order to be compliant with applicable privacy and data protection laws and regulations, such as the GDPR.

4.2.4 Monitoring

Monitoring of the system by tracking meaningful statistics allows for an accurate view of the system's performance, availability, and overall health. To this end, metrics, events and metadata are collected in order to generate insights via graphical interfaces, reports, and alerts.

4.3 Offering a well-structured and well-documented API

An essential part of the Cloud Gateway component will be the API functionality, which will allow interaction between the Gateway's functions and procedures. The need to have a programming interface, which will allow the effective interaction between the services inside the PolicyCLOUD platform and the Gateway, will be met through the utilization of the API functionality. Using a REST API is considered a good practice and is a very common way for web service communication. Therefore, the design of the REST API has a great impact on the security, performance, and ease of use for API consumers. A well maintained, futureproof design for the REST API is surely required to implement a reliable and fault-tolerant Gateway component. Some of the best practices that will be utilized during the development of REST API are included in [9]:

JSON for HTTP requests and responses: Using a REST API means both request payload and responses should be in JSON format, as this is the standard for transferring data. Although XML format is a possible alternative selection, it is not widely supported by all frameworks. Furthermore, data in JSON format can easily be manipulated, especially in NodeJS frameworks, without the need to transform data before the start of processing. Moreover, a wide range of NoSQL databases, like MongoDB, save data in this format.

Proper naming conventions: It is very important for a REST API to have a strong and consistent naming convention strategy. To this end, the usage and understanding of a well and properly named API is easier. In the opposite case, a poor naming strategy can lead to confusion and misusage of the API by its own users.

Filtering and Pagination: Datasets obtained from external sources are expected to be very large. In order not to slow down the overall system, data processed in the REST API must be paginated. Filtering and pagination can increase performance and reduce resource usage for longer time periods.

Caching: Caching can be applied to data, so that it can be retrieved from local memory instead of repeating the same queries over and over. Caching in different levels of the Gateway (e.g., at the application level, using Redis) can help to significantly reduce resource usage and allow for the Gateway component to work faster and more efficiently. On top of this, proper configuration of caching mechanisms is important for PolicyCLOUD's Gateway component, especially in occasions where there may be a demand of the most updated data results, like data streams, etc. On the other hand, improper caching configuration can cause other services to retrieve false data as input, leading to inaccurate results.

Versioning: Following a versioning system is important for the system's maintainability and for users. The most popular versioning system is Semantic Versioning [10], following the MAJOR.MINOR.PATCH pattern.

1. MAJOR version for backwards-incompatible changes;
2. MINOR version for newly-added backwards-compatible functionality to the API;
3. PATCH version for backwards-compatible patches and bug fixes.

Through the use of semantic versioning, the Cloud Gateway component is able to handle what in programming world is referred to as "dependency hell", meaning the inability to further extend the system because the application uses many shared libraries, and these libraries may depend on other libraries, causing compatibility problems for the whole system, which adds complexity and may cause frustration for users.

Error Handling: When a system is operational, errors occur. To eliminate the possibility for an error to bring the whole system down, errors must be handled gracefully, and well-defined responses that indicate the kind of error that occurred should be returned. This allows maintainers to better understand and fix possible bugs. Common error HTTP status codes are:

- 400 Bad Request;
- 401 Unauthorized Request – Only authorized users have access to this resource;
- 403 Forbidden – Access to that resource is forbidden;
- 404 Not Found – Resource not found;
- 500 Internal server error – Generic error indicated some problem with the server;
- 502 Bad Gateway – Invalid server response;
- 503 Service Unavailable – Error occurred in server.

Security Practices: Security should be an important part of the API's development. In monolithic applications, security is easily centralized, with one interceptor that intercepts between calls; however, in a microservices environment, this functionality can be very challenging. Moreover, since a RESTful API is stateless, there is the need to implement a way of authentication/authorization that does not depend on techniques like sessions or cookies. The JSON Web Token (JWT) [11] is a compact and self-contained way to securely exchange information between parties in JSON format. JWTs can be encrypted to provide a proper layer of secrecy in communication. Furthermore, JWT offers signed tokens which verify the integrity of contained data. A basic request implementing JWT consists of 3 basic parts:

- **Header:** Contains the token and the hashing algorithm;
- **Payload:** Consists of information about the authenticated entity along with metadata;
- **Signature:** Used to check the authenticity of the JWT – this is generated by hashing Header and Payload together, along with a secret key.

During the design of the security practices, performance and scalability must also be considered. As JWT includes a cryptographic operation to validate the token, caching could help significantly reducing the time and resource impact of repetitive token validations.

The first version of the Cloud Gateway REST API, following Open API specification, is available and has been deployed. The Open API approach was followed in order to make it easier for users to understand 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, using Swagger UI, so that we can provide a graphical interface for interacting with the API. The documentation for all

available endpoints, short descriptions, parameters, request bodies and sample responses can be found in the Swagger UI page.

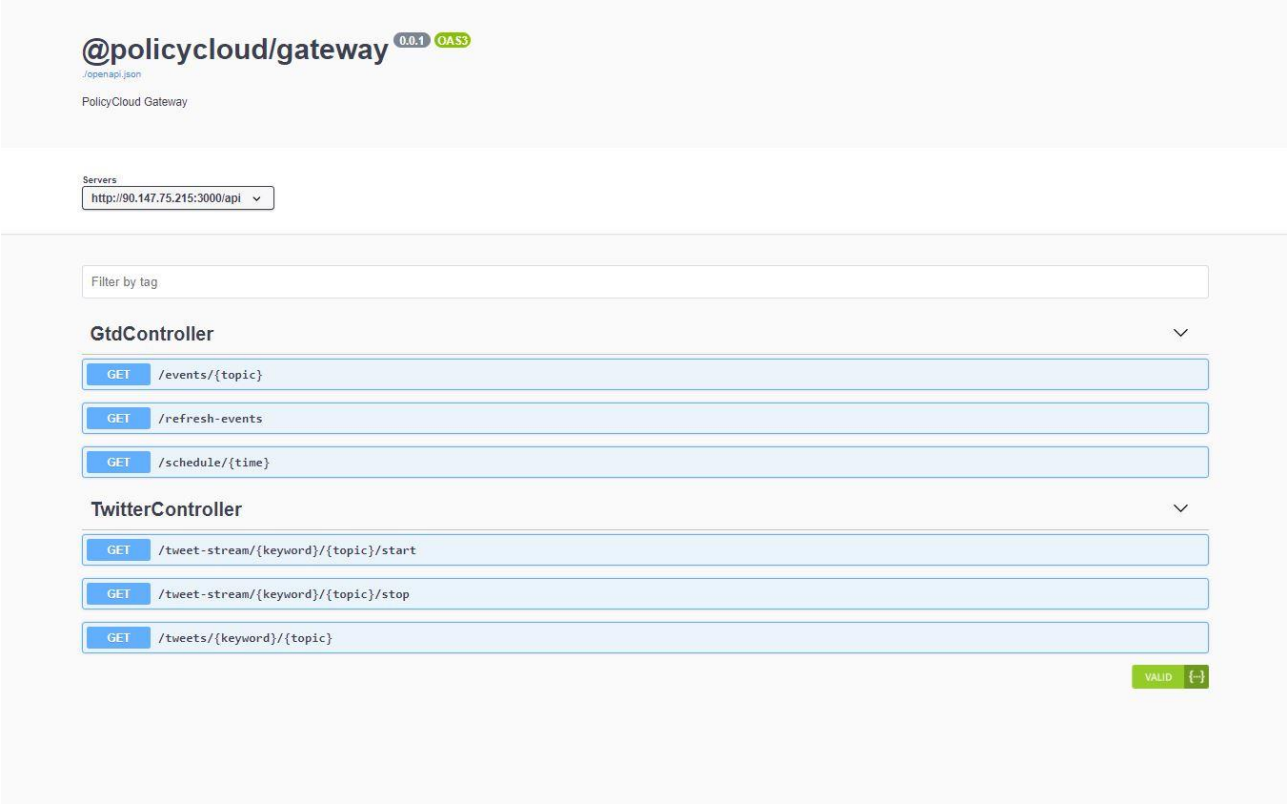

FIGURE 4 – GATEWAY OPEN API DOCUMENTATION

4.4 Technologies & Methodologies

The PolicyCLOUD Gateway will be designed and implemented using a microservices architecture methodology. Every part of the Gateway will be wrapped in its own service, executing its own process. This architectural methodology allows the construction of highly maintainable systems that are independent and loosely coupled, easily testable and deployable.

4.4.1 Services

PolicyCLOUD Gateway's microservices will utilize MoleculerJS [12], a framework built on top of NodeJS that is oriented in microservices development. NodeJS has been developed based on a V8 runtime engine and can be very efficient for input-output (IO) heavy tasks, in contrast with CPU-bound tasks such as calculations, which have a high demand on resources. To this end, NodeJS is an excellent choice for IO-bound tasks, such as those included in the Gateway's main functionalities. On top of this, along with a very large community, many innovative enterprises add NodeJS to their main technology stack, as it increases productivity and offers high performance at a much lower cost.

Although MoleculerJS offers many out-of-the box features and it is very performant, it has many features and a large learning curve, which resulted in some drawbacks and bottlenecks regarding time for implementation. As such, the first version of the Gateway component was implemented using the LoopBack4[6] framework. Moreover, this framework facilitated the focus on the business logic and the providing of the required data from external sources to PolicyCLOUD, in order to be further processed and analyzed by other components. Both above mentioned frameworks, MoleculerJS and LoopBack4, are Node JS frameworks, meaning that most business logic parts can be transferred and mitigated to each other without major changes and time drawbacks.

4.4.2 Dynamic Service Discovery

In a microservices environment, the running instances of services dynamically changes location inside networks. In order for clients or services to make requests to a service, they must use a service-discovery mechanism. MoleculerJS has a built-in module to handle local service discovery, but also supports integration with in-memory data stores, like Redis, for both local and remote service discovery. Using Redis-based discovery, a Redis server is used to keep track of available nodes. MoleculerJS nodes (running services) periodically publish their status (heartbeat) and fetch information from Redis to update their internal service registry. The Redis key expiration mechanism removes nodes that do not publish heartbeat packets for a specified period. This keeps the service registry updated and allows MoleculerJS nodes to detect disconnected nodes.

4.4.3 Load Balancing

To ensure that the Gateway will not fail in production, multiple nodes of the same services must be deployed when needed. When a service is called, the request must be routed to the instance that has the lowest load at that moment. MoleculerJS has built-in strategies for load balancing. A round-robin algorithm is a very simple, yet very effective algorithm, which finds the most appropriate node to which to send traffic, by using an appropriate statistical model. More straightforward strategies can be used based on CPU-usage or latency of every node.

4.4.4 Fault Tolerance

Microservices need to be designed so that they are fault tolerant, meaning that if an error occurs, it should be handled gracefully, and the service should return an error message. To this end, there are a few patterns that ensure service resiliency. Using the Circuit Breaker pattern, faults which require a large amount of recovery time can be handled. Failing to fetch external APIs can be a time- and resource-consuming task. The Circuit Breaker pattern prevents repeated attempts to execute operations which have a high probability to fail. It also allows to detect whether the fault has been resolved and to try to

[6] https://loopback.io/

invoke the previously faulty operation. Moreover, it is a good practice to set a timeout for service calling. On top of this, the utilization of MoleculerJS to build services inside the Gateway and to configure different fault tolerance strategies is very straightforward and easy to implement.

4.4.5 Caching

Through the usage of the MoleculerJS framework for the implementation of various microservices, out-of-the-box caching solutions are offered. Redis caching is utilized using the ioredis NodeJS library. In case multiple nodes are needed for one service, the Redis distributed cache module will allow sharing cached data across these instances. Moreover, this service, coupled with options to use serializers, like MsgPack, allows the faster storage of cached data faster, requiring less memory than a simple JSON serializer.

4.4.6 Messaging and Message Brokers

Services need to communicate with each other, and often to collaborate to handle requests. Synchronous communication requires both services participating in the request to be available for the duration of the request, usually over HTTP or REST. For inter-service communication, an asynchronous approach that allow exchanging messaging over specific channels is needed. MoleculerJS allows for the nodes to communicate with each other by using a large variety of protocols and message brokers like NATS, Redis, MQTT, AMQP and Kafka. Kafka is suitable especially for occasions of large data streams and is remarkably fault tolerant by supporting "replay" scenarios.

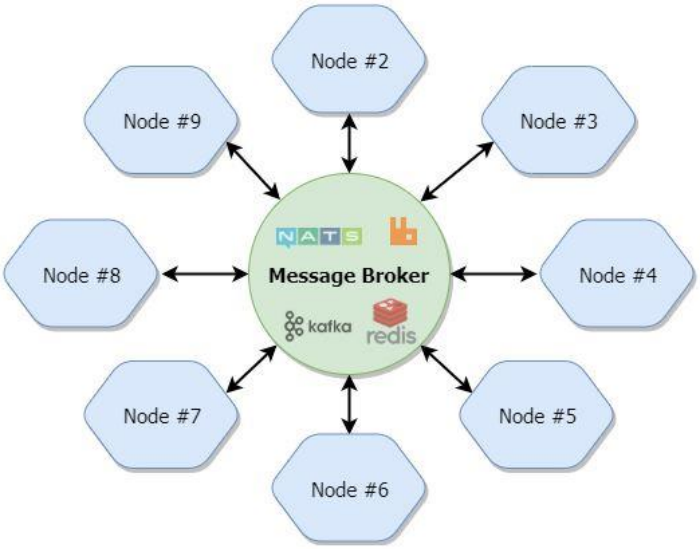

FIGURE 5 – SUPPORTED TRANSPORTERS CONNECT TO A CENTRAL MESSAGE BROKER [13]

4.4.7 API Gateway

Services can be easily published as RESTful APIs using the moleculer-web, the official API gateway for the MoleculerJS framework, allowing route whitelisting, throttling, mapping, authorization/authentication and many other features a modern API should be able to support.

4.4.8 Data Transformation and Database Adapters

MoleculerJS allows connectivity with both SQL and NoSQL databases. More specifically, it includes official adapters for MongoDB, PostgreSQL, SQLite, MySQL, MSSQL. Data manipulation is also possible by using action hooks before sending data to storage services inside PolicyCLOUD, to remove sensitive or unnecessary information (including data which should not, for legal/ethical purposes, be further stored, in line with the requirements identified in deliverable D3.3 [16]).

Data ingested from external sources are encoded in Avro Schema before being sent to the PolicyCLOUD Message broker mechanism, which is based on a Kafka tool.

4.5 Next Steps

With regards to the next steps that will be followed, at first a more detailed description of system microservices and operational constraints, such as how the system will be utilized, will be provided. The external microservices that will be used for information gathering and the requested output from the Gateway's services will be further developed.

On top of this, a detailed architecture for all the provided and exposed microservices, that will be implemented, will be delivered. Moreover, communication between microservices, including dataflow diagrams that allow the visualization of data flows inside the Gateway, will also be designed and described.

The Cloud Gateways component is designed following a microservices architecture. Hence, every microservice is running in individual nodes and the communication between nodes is enabled via a TCP transporter.

Moreover, in the next steps, an OpenAPI specification for the Gateway's REST API will be delivered, describing the entire API functionality, including:

- Endpoints;
- Authentication methods;
- Parameters;
- License, terms of use of information.

The basic API structure and specifications provided can be found in the Swagger UI that is available. Furthermore, next steps include updating these specifications with the latest endpoints.

Moreover, special attention will be given according to the development and implementation of all microservices and the communication between them in a containerized environment following the detailed README file which is being provided.

All microservices are being containerized using Docker, in order to ensure fast deployment, better service management and easier maintenance. More components for ingesting data from other external sources will be added in the future according to the needs and specifications of project's pilot use cases.

Finally, the final evaluation of the overall provided and implemented system will be held, by checking requirements validation, system integration, security, performance, stress, usability, and user acceptance. This step also includes bug fixes that may be necessary.

The first version of the Cloud Gateway component and all of the microservices are deployed on a VPS provided by EGI and Recas-Bari partner, through the project's cloud infrastructure.

5 Incentives Management

5.1 Updates Since D3.1

In this new version of the Design and Open Specification for WP3, a full detailed specification of the Incentives Management component is provided. New subsections detailing the state of the art and objectives of the Incentives Management tool, the functionalities description, architecture, data model, integration, and interface are included. Updated next steps are also provided.

5.2 Incentives Management in the scope of PolicyCLOUD

The Incentives Management component is based on the idea that including citizens as participants in the policy development and management process enhances the acceptability of, and confidence in, created policies. In this regard, PolicyCLOUD introduces a component that allows policy makers to manage incentives to boost the participation of citizens, allowing them to be part of an evidence-based policy making process.

In this context, the Incentives Management component is a tool to identify, declare and manage incentives for citizens' engagement. The types of incentives may vary to best match the goals of the incentives and the audience to whom they are addressed.

In particular, the types of incentives may be, e.g., social, cultural, political, etc. Participants may be classified as closed groups (i.e., communities evaluating proposed policies) or individually engaged citizens. Participants may propose social requirements which, once analysed, could be turned into policy requirements.

It is necessary to manage motivations, incentives and corresponding actions. Compensations can be useful to motivate the "crowd" to participate. In this regard, and based on the paper from Katmada et al. [14], and according to Figure 6 below, user motives are represented in the middle inner circle, each one with a different colour, while suitable incentives for each motive are in the outer circles, using the same colour as the corresponding motive.

FIGURE 6 – INCENTIVES MANAGEMENT MOTIVES AND INCENTIVES, EXTRACTED FORM KATMADA ET ALL [14]

The scope of the Incentives Management tool is limited to tracking the different incentives, motivations and actions managed by the policy maker. The specific incentives and motivations should be managed externally, using ad-hoc crowdsourcing platforms that best suit the needs in each environment (type of policy, group to which is addressed and other constraints).

5.3 Functionalities Description

After logging into the PolicyCLOUD GUI (Graphical User Interface) platform, using their own project credentials, policy makers will be able to manage all items of the Incentives Management Tool part of the platform. The main items related with the tool are the following:

- Incentive:
  This section allows policy makers to manage incentives: here, new incentives can be created, and existing incentives can be edited/deleted. An incentive is a way of causing the activation of a concrete motive in an individual, or group of individuals, and aims to result in an action on their part. In this sense, incentives can help in citizens' engagement and participation in the policy creation/modification process.
- Motive:
  This section allows policy makers to manage motives: here, new motives can be created, and existing motives can be edited/deleted. Motives are possible reasons that motivate citizens'

engagement and participation in the policy creation/modification process. Each incentive has a motive that drives it, and each motive can be activated by one or more incentives.

- Action:
  This section allows policy makers to manage actions: here, new actions can be created, and existing actions can be edited/deleted. Each action is related to an incentive, and represents the action needed for the incentive to be carried out.
- Producer Typology:
  This section allows policy makers to manage types of producers: here, new producers can be created, and existing producers can be edited/deleted. Producer types are different groups of co-producers or user participant types, related with the actions.
- Report:
  This section allows policy makers to manage reports, produced by external CS (Crowdsourcing) tools.

The following image in Figure 7, where a diagram with the entire process can be seen, depicts the dependencies between these items:

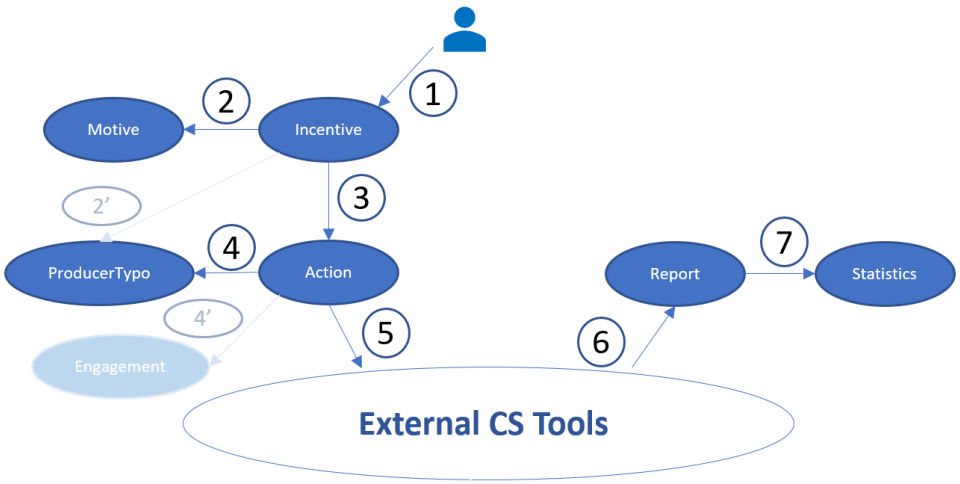

FIGURE 7 – INCENTIVES MANAGEMENT FUNCTIONALITY WORKFLOW

The process can be detailed as:

1 – A policy maker creates a new incentive through the Incentives Management Tool in the PolicyCLOUD GUI Platform;

2 – The new incentive activates (or can be associated to) a motive, which could be a new motive or an existing motive. The policy maker, through the Incentives Management Tool, links the new incentive with the relevant motive;

3 – The policy maker, through the Incentives Management Tool, creates a new action, related to the new incentive, and links both. The idea is to analyse which compensations would match specific crowdsourcing tasks for the purposes of motivating the crowd;

4 – In the creation process of the new action, the policy maker must select a Producer Typology associated with the action, between the existing ones;

5 – Actions are then executed through external CS Tools;

6- The Incentives Management Tool stores and manages external generated reports. These stored reports, generated following a specific pattern, can be used to provide insights, useful for the generation or modification of policies;

7- These reports can also provide statistics, that are also a helpful tool to provide insights to policy makers.

5.4 Architecture

The Incentives Management (IM) component is composed of three main parts:

- The "Incentives Portal", which is responsible for providing a GUI to the policy maker, with the aim of providing a friendly and intuitive interface to the tool, facilitating the policy maker's work when managing everything related to incentives;
- The "Access to policies" component, which allows policy makers to see results from their policies, providing them with insights for the creation of new incentives;
- The "Incentives Management Back End", which is responsible for managing the data stored in the IM database.

The communication between the "Incentives Portal" and the "Incentives Management Back End" has been designed following the CRUD (Create, Read, Update, Delete) approach, in order to create and manage the models used within. Thus, this approach can be seen in each component of the platform landscape.

The "Incentives Portal" component has three main layers:

- Service: contains the business logic of the application that interacts with the back end component;
- Component: manages the data from the service layer to the presentation layer;
- Presentation: displays the information in a human-readable format, to be used by the policy maker.

The main components of the back end can be split in following parts:

- Components designed to manage the database content and expose APIs, in order to use the CRUD approach to work with them. These components are:
  - "Incentive Manager": Used to manage incentives;
  - "Action Manager": Used to manage actions;
  - "Producer Typology Manager": Used to manage producer typologies;
  - "Report Manager": Used to manage reports.

- Components that provide help to the users:
  - "Exploitation Results": Provides some analytics to the policy maker using the reports that have been uploaded onto the platform.
- For user management, which is needed in any application to be able to manage users' access to the platform, the Incentives Management Tool uses the common user management component of the project in order to facilitate both the tool's integration into the PolicyCLOUD platform and its usability by policy makers. This integration is detailed in section 5.6 of this document.

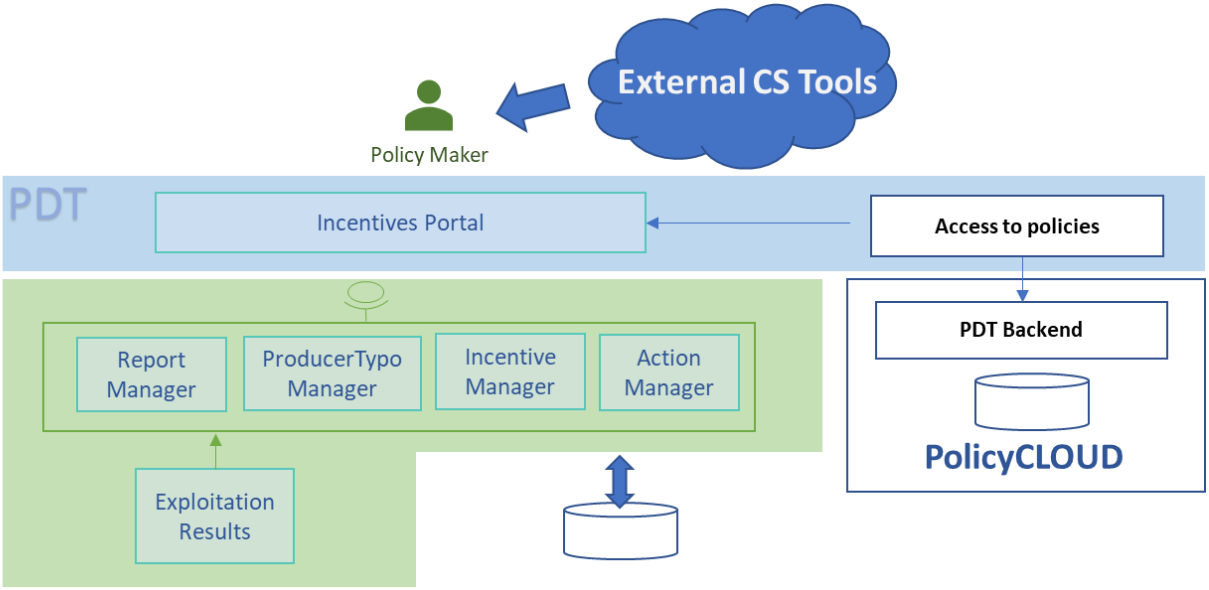

FIGURE 8 – BLOCK DIAGRAM OF THE INCENTIVES MANAGEMENT ARCHITECTURE

5.5 Data Model

The first version of the Incentives Management Tool's data model has been designed taking into account the Motive-Incentive-Activation-Behaviour Model (MIAB), explained in [14] and adapted for PolicyCLOUD's purposes. This model has been validated by the project Use Case of Sofia (Use Case 3, as described in D6.3 [8]).

The first version of this diagram is depicted in the following UML (Unified Modelling Language) diagram:

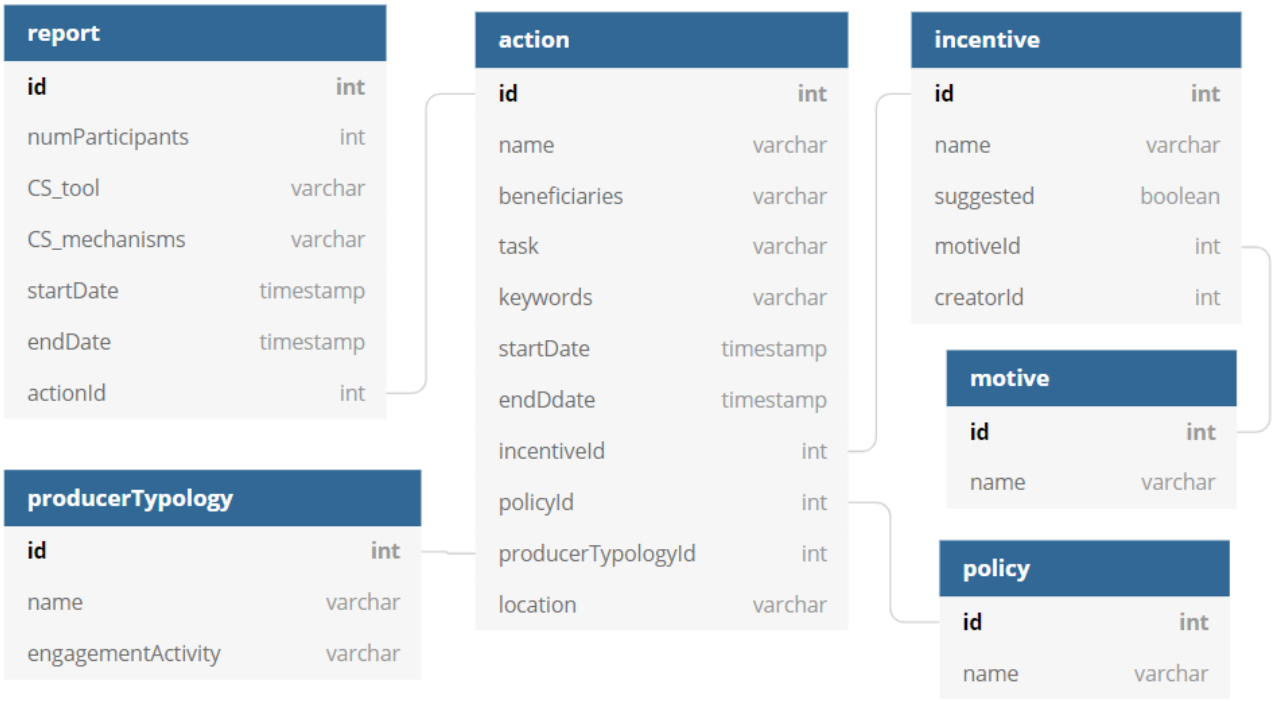

FIGURE 9 – THE INCENTIVES MANAGEMENT DATAMODEL

This data model, as a first approach, is not closed yet. It is an open first version that could be updated according to project needs and evolution in the next steps.

5.6 Integration

The integration of the Incentives Management Tool has been designed to make it easily accessible and intuitive, thus facilitating its use by policy makers. Following this thought, the idea to integrate it inside the PDT turned out to be the most appropriate, in such a way that policy makers can have a single user interface from which they can manage the PDT functionalities itself, the Project Model Editor and the IM.

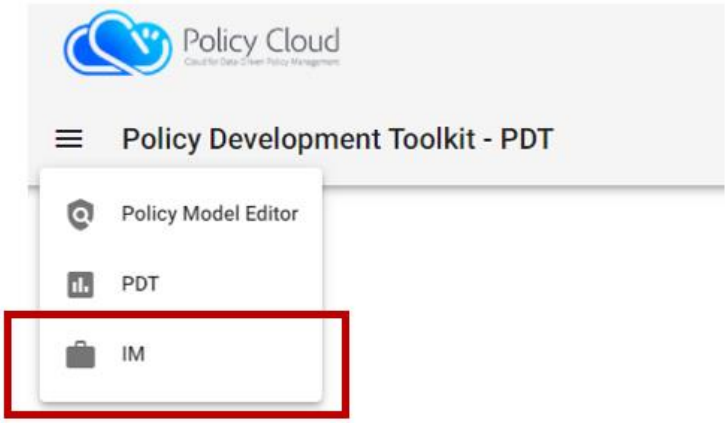

FIGURE 10 – THE INCENTIVES MANAGEMENT TOOL ACCESS FROM PDT

However, as seen in Section 5.4 Architecture, the fact that the front end part of the Incentives Management Tool is integrated into the PDT does not affect its back end or storage components. These components have been designed so as to be independent, with different technologies.

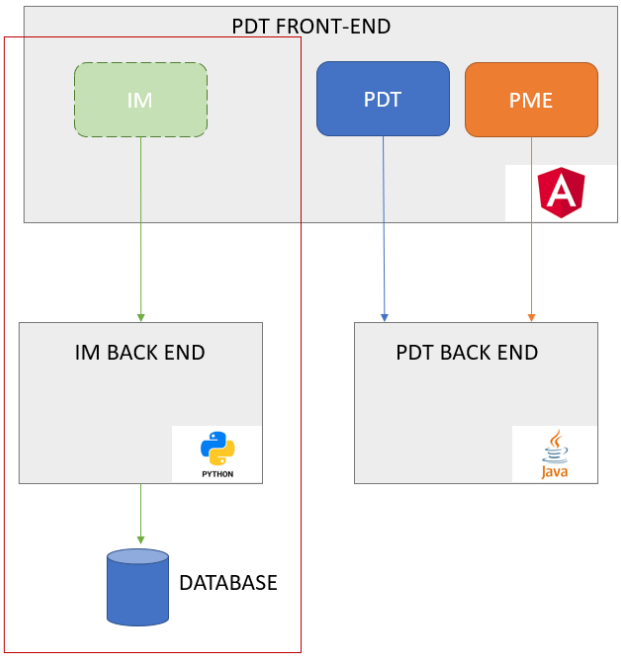

FIGURE 11 – THE INCENTIVES MANAGEMENT TOOL ARCHITECTURE

5.7 Interface

As the Incentives Management Tool front end is integrated in the PDT, the technology selected to facilitate this integration was the same used for the PDT front end, therefore Angular has been used.

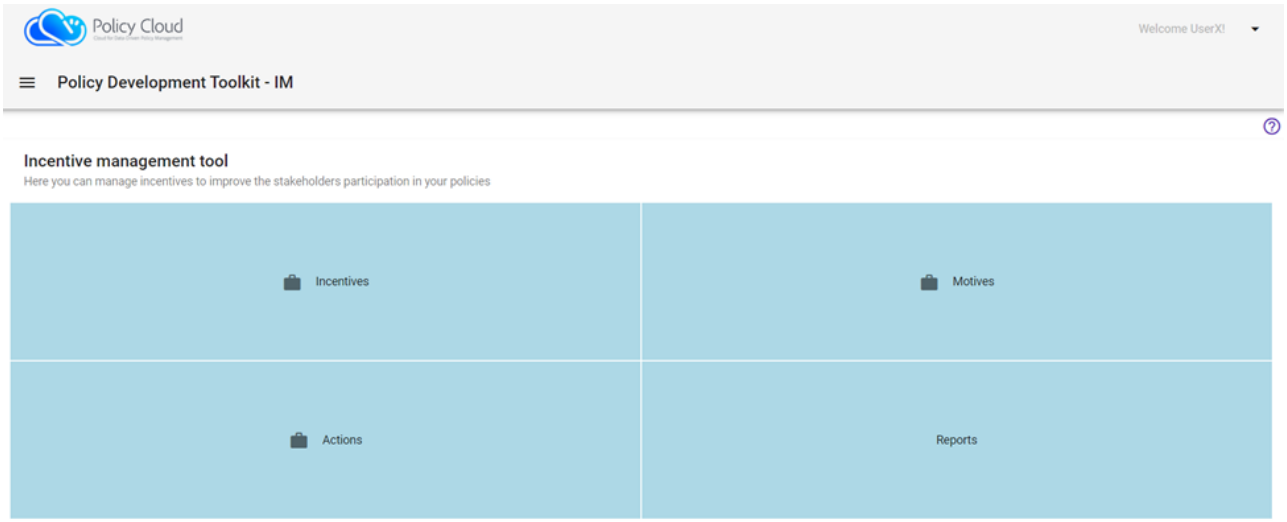

FIGURE 12 – HE INCENTIVES MANAGEMENT TOLL FRONT END

From the first screen of the Incentives Management Tool front end, as depicted in Figure 12 above, the policy maker is able to access all the well-defined options of the tool:

- **Incentives**: Gives access to all the actions related with incentives: create new incentives, update and/or delete existing incentives;
- **Motives**: Gives access to all the actions related with motives: create new motives, update and/or delete existing motives;
- **Actions**: Gives access to all the actions related with actions: create new ones, update and/or delete existing actions;.
- **Reports**: Gives access to the reports that can be consulted by policy makers. These reports will give information about the policies created by the policy makers.

For the back end components of the Incentives Management Tool, which is independent of the PDT's back end, the technology chosen for its development was Python.

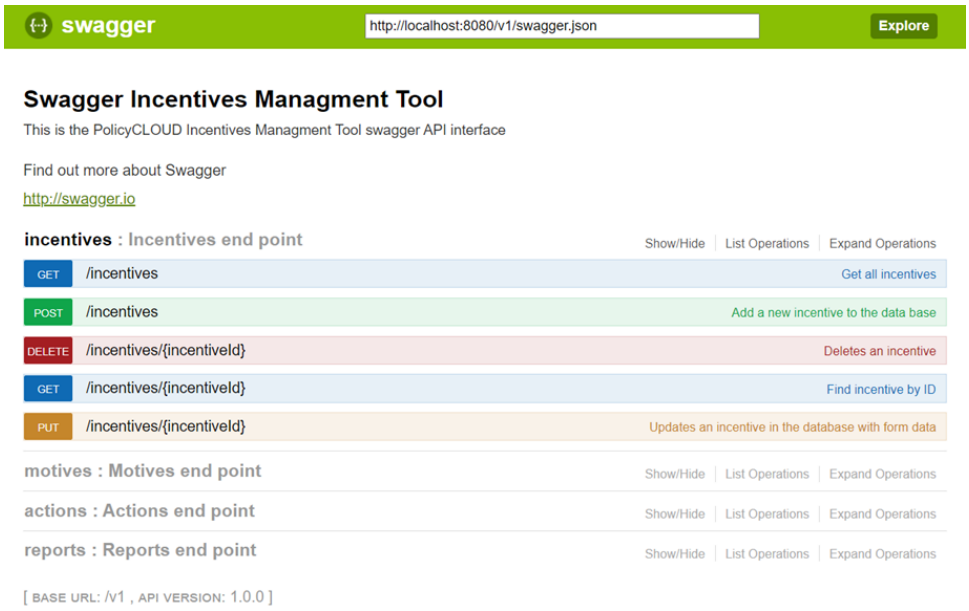

FIGURE 13 – THE INCENTIVES MANAGEMENT TOOL BACK END

This back end provides all the end points referring to all the actions following the CRUD method that can be performed from the front end, related to incentives, motives, actions and reports. The previous screenshot is a mock-up of the Swagger user interface of this end point.

5.8 Next Steps

The next steps are to develop and deploy the component in one of the pilots, to assess its suitability and to check if this approach matches the needs in this environment. The data model will also be updated according to project needs and evolution.

6 Data Governance Model, Protection and Privacy Enforcement

6.1 Updates Since D3.1

In addition to what has been presented in D3.1 [15], in this document we examine data governance as part of PolicyCLOUD, in order to prepare the model and the accompanying tools that will allow the secure access and sharing of data. For this, in Section 6.2, we provide an overview related to privacy enforcement and related basic concepts, and in Section 6.3, we present the specifications for the protection and privacy enforcement mechanism of PolicyCLOUD, which ensures that data used across the PolicyCLOUD lifecycle follows specific guidelines and legal/ethical requirements. Finally, in Section 6.4, we examine possible models used for data governance, and how these can be supported by PolicyCLOUD.

6.2 Data Protection and Access Control in PolicyCLOUD

One of the core aspects of protection and privacy mechanisms is to provide logical access control to generated data. This has also been identified as a core legal/regulatory requirement within deliverable D3.3 [16]

- "Platform users shall be limited from both a technical and from a contractual point of view in how they can process personal data which are collected and managed through the PolicyCLOUD platform";
- "Internal policies shall be implemented to make users aware of what they can and cannot do with the personal data collected for the different use cases and more in general for the execution of the Project".

Towards this, in D3.1, we presented an overview of different authorization methods and how attribute-based access control (ABAC) and the XACML-based implementation provided by Balana Library seem the most appropriate for PolicyCLOUD's needs. We explain basic concepts here – readers can find a more detailed analysis of the various frameworks in D3.1.

6.2.1.1 Authorization & Access Control Mechanisms

First, it is important to clarify the difference between *authorization* and *authentication*. Authentication deals with the problem of proving the identity of a natural person or a system entity that aims to interact with a resource, while authorization deals with the problem of providing logical access control to these resources. Logical access control is always bound to specific actions that the natural person/system entity (hereinafter "subject") wishes to perform, e.g., discovering, reading, creating, editing, deleting, or executing.

The "problem statement" of an authorization request entails some concepts that are common to all approaches. These include the **subject** (i.e., requestor), the **object** (i.e., resource), the **action** (to be applied to the resource), the environmental context, the defined policy (or policies) and the policy-evaluation business logic (i.e., Policy Enforcement).

Access Control Mechanisms are mechanisms realizing various logical access control models that provide the framework and a set of boundary conditions upon which the objects, subjects, operations, and rules may be combined to generate and enforce an access control decision.

6.2.1.2 ABAC

ABAC uses attributes and policies that express boolean rule sets that can evaluate many different attributes before allowing access. ABAC, therefore, avoids the need for capabilities (operation/object pairs) to be directly assigned to subject requesters or to their roles or groups before the request is made. IBAC and RBAC can be seen as special cases of ABAC, with IBAC using the attribute of "identity" and RBAC using the attribute of "role".

Any ABAC system should implement the conceptual flow that is depicted in Figure 14. According to this flow, any subject can perform an access request for a specific operation regarding a specific object (step 1).

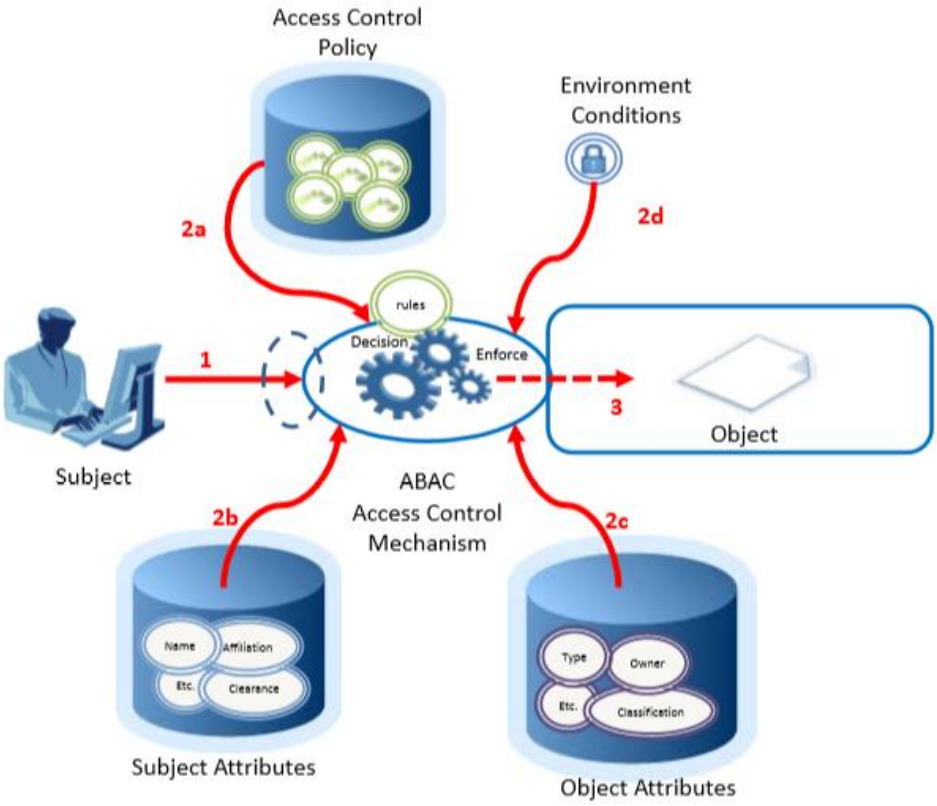

FIGURE 14 - ABAC INDICATIVE INFORMATION FLOW

The access request is handled by the ABAC reference implementation engine, which consults a policy repository (step 2a) to derive the set of attributes that must be examined to reach a decision of "allow" or "deny". The attribute examination phase checks the subject's attributes (step 2b), the object's attributes (step 2c) and environmental attributes (step 2d) to perform the actual assessment (step 3).

It should be clarified that ABAC is a theoretical framework and not a standard. In the next section, we will elaborate on the XACML standard that is used for the PolicyCLOUD Data Governance and Privacy Enforcement mechanism.

6.2.1.3 XACML

XACML [17] is an OASIS standard that describes both a policy language and an access control decision request/response language. It defines five main components (see Figure 15) that handle access decisions, namely Policy Enforcement Point (PEP), Policy Administration Point (PAP), Policy Decision Point (PDP), Policy Information Point (PIP), and a Context Handler, as depicted in Figure 15.

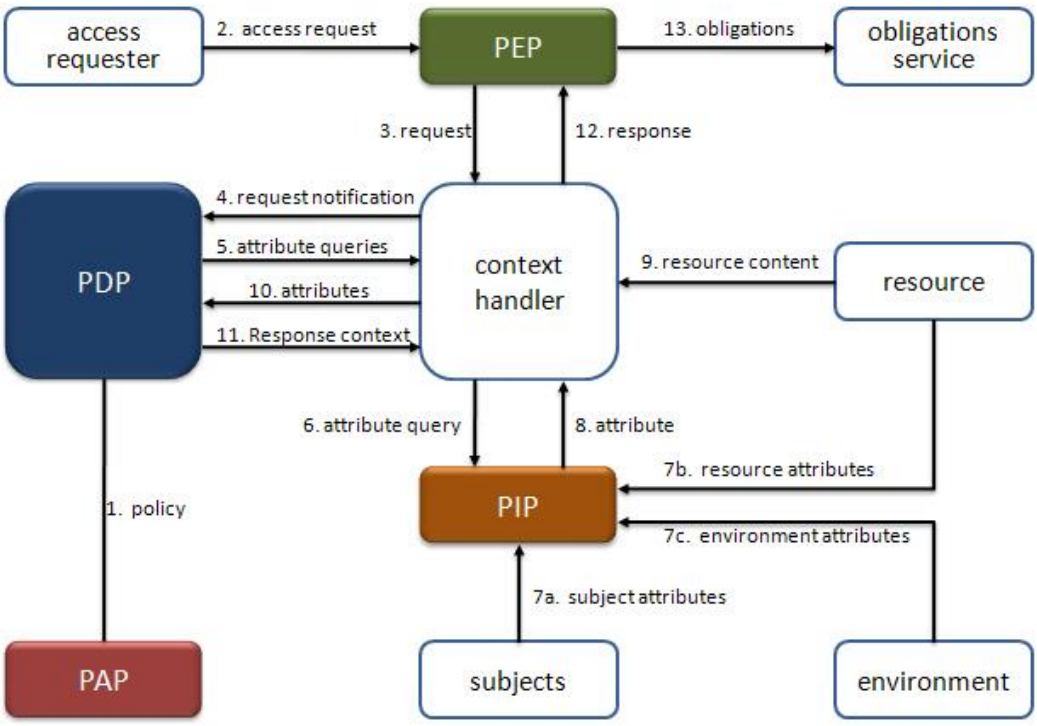

FIGURE 15 - XACML FLOW & ARCHITECTURAL COMPONENTS

The functional purposes of the main components are as follows:

- The Policy Administration Point (PAP) is the repository for the policies, and provides policies to the Policy Decision Point (PDP);
- The Policy Enforcement Point (PEP) is the interface of the whole environment with the outside world. It receives access requests and evaluates them with the help of the other actors, and permits or denies access to resources;

- The Policy Decision Point (PDP) is the main decision point for access requests. It collects all the necessary information from the other actors and calculates a decision;
- The Policy Information Point (PIP) is the point where the necessary attributes for the policy evaluation are retrieved from several external or internal actors. The attributes can be retrieved from the resource to be accessed, environment (e.g., time), subjects, and so on.

Having provided the basic background needed for the understanding of access control through ABAC & XACML, we provide below more details about the actual mechanism provided in PolicyCLOUD.

6.3 PolicyCLOUD Data Governance and Privacy Enforcement mechanism

The Data Governance and Privacy Enforcement mechanism includes three different parts: a) the access policy editor, b) the model and model editor, and c) the ABAC authorization engine. The access policy editor will provide the user with the ability to define and store policies based on the ABAC scheme according to the XACML standard. The data model of PolicyCLOUD will be used for the definition of these policies, and also for the actual enforcement of the policies through the authorization engine. The authorization engine will be able to evaluate policies and attributes, thus enforcing protection and privacy-preserving policies.

6.3.1 PolicyCLOUD Policy Enforcement

For the implementation of the policy enforcement engine of PolicyCLOUD, Balana has been used. Balana was the first open-source reference implementation of the XACML protocol and is a widely adopted solution. Balana has two major components: the Policy Administration Point (PAP) and Policy Decision Point (PDP).

A Policy Enforcement Point (PEP) has been implemented as a filter that can be included in a web client, to trigger the necessary interception to the ABAC Authorization Engine. It depends on the ABAC client and acts as a filter for access to a specific API or website, restricting access until it receives authorization from the ABAC engine. More information regarding the introduction of the PEP in an application, and also the creation of policies, will be provided in the prototype deliverable D3.5.

Regarding the Policy Information Point (PIP), this is a module that is responsible for finding missing attributes for a given request, using the underlying PIP attribute finders provided by Balana. Technical details for this engine can be found in D3.2 [1] and will be provided in the upcoming deliverables D3.5 and D3.8.

The final part of the mechanism is the actual usage of the policies. The XACML-based policies are applied to a PEP and are used for the decision to allow or deny each request. A very basic sample policy is provided below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xacml3:Policy xmlns:xacml3="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 PolicyId="MyPolicy_0"
 RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
 Version="1.0">
        <xacml3:Description>Permit ALL Policy</xacml3:Description>
        <xacml3:Target>
        </xacml3:Target>
        <xacml3:Rule Effect="Permit" RuleId="Permit-Rule"/>
</xacml3:Policy>
```

6.4 Data Governance in the scope of PolicyCLOUD

The term "data governance" is extensively used, but its meaning is still ambiguous [18]; similarly, it is also seemingly used with various scopes. For PolicyCLOUD, we used the definition provided by [19] that "data governance" reflects the power relations between all the actors affected by, or having an effect on, the way data is accessed, controlled, shared and used, the various socio-technical arrangements set in place to generate value from data, and how such value is redistributed between actors [19]. Therefore, for PolicyCLOUD, we have to provide an appropriate mechanism for accessing, controlling and sharing data.

6.4.1 Data Governance Models

As our goal was to identify other models of data governance that could be potentially supported by PolicyCLOUD, we based our work on the analysis of [19]. While there is the possibility to implement a general, dominant governance model, based on a single entity doing all the governance, we also examined other models that also allow other actors, such as small businesses, public bodies or even the data subjects themselves, to take part in data governance. We will then present how PolicyCLOUD can support these different data governance models.

The identified data governance models are presented below:

- Single Entity Data Governance Model

Accessing and sharing data is usually done through this model, where a single platform is collecting and economically exploiting massive amounts of data.

- Data Sharing Pools (DSPs)

Different actors join DSPs to analyse each other's data, and help fill knowledge gaps while minimizing duplicative efforts [20]. By creating these partnerships, they ease the economic need for exclusive rights and obtain limited co-ownership stakes in the resulting data pool. This means that the goal of this model is to share data with the minimum effort, while various owners of data are able to manage and share their own data.

- Data Cooperatives (DCs)

Data cooperatives present a de-centralised data governance approach, in which data subjects voluntarily pool their data together, to create a common pool for mutual benefits [21]. This means that the management and governance of the data is performed in a distributed way and by different actors, guided by different goals, which allows for higher involvement of data subjects.

- Public Data Trusts (PDTs)

Public Data Trusts are a data governance model in which a public actor accesses, aggregates and uses data about its citizens, including data held by commercial entities, with which it establishes a relationship of trust [22]. Several stakeholders might be involved in this model, including city administrators, managers of public institutions, platform companies, trusted data intermediaries, research institutions, start-ups, and SMEs.

- Personal Data Sovereignty (PDS)

The PDS model is characterized by the promise of data subjects having greater control over their data, both in terms of privacy management and data portability, compared to existing other models. A limitation of this model lies in its dependence on personal data spaces, as these are currently adopted by only a niche of users and often fail to scale beyond pilots [23].

6.4.2 Data Governance models supported by PolicyCLOUD

PolicyCLOUD provides a mechanism to support organizations that is compatible with most governance models. By utilizing ABAC, PolicyCLOUD gains greater flexibility through an attribute-based access control mechanism that is able to support complex scenarios of allowing or denying access to data. In addition, the usage of PEPs that allow the assignment of policies in real-time and by different actors allows the mechanism to be used by different stakeholders/actors to achieve specific governance goals, and thus cover various data governance models.

The following Table 1 presents how each of the models is supported by the developed mechanism:

Model	How it is supported by PolicyCLOUD
Single Entity Data Governance Model	The entity adds a PEP and then creates appropriate XACML policies that control the access to data by external stakeholders.
Data Sharing Pools (DSPs)	Each entity of the DSP provides the data to the sharing pools, and the PEPs are added. Each entity then is able to create appropriate XACML policies that control the access to each DSPs' data.
Data Coooperatives (DCs)	Each entity of the DC can add PEP and creates appropriate XACML policies that control the access to each DC's data.
Public Data Trusts (PDTs)	The PDT collects all data, then adds a PEP for the control of access to the data. Then, based on the PDT scenario needs appropriate stakeholders are entitled to create the XACML

	policies that control the access to data by external stakeholders.
Personal Data Sovereignty (PDS)	Each owner of a personal data space is able to add a PEP to the data space, and then create XACML policies. For unification reasons, a common model for the XACML policies has to be used as well as a verified PIP to provide attributes in a secure and transparent way.

TABLE 1 - MODELS SUPPORT FROM POLICYCLOUD MECHANISM

6.5 Next Steps

In the next period, the second version of the software prototype will be delivered and documented in D3.5. This means that the policy enforcement mechanism will be updated according to the updated specifications, and more detailed access control scenarios and their corresponding policies will be provided. For the final release of the platform, we will also work on improving the mechanism with the inclusion of a user interface.

In addition, a detailed architecture for all services that will be implemented will be delivered in the next iteration of this deliverable.

Regarding the deployment, the first version of the governance mechanism is already deployed on the project's cloud infrastructures provided by EGI and Recas-Bari. The updated version of the mechanism for the second prototype will also be deployed on the same infrastructure.

Finally, starting from the deployment of the second prototype, we will focus on resolving issues and also ensuring system validity by checking requirements, proper integration and the security of the provided mechanism.

7 Conclusion and Next Steps

In this document, we present the progress of tasks T3.1, T3.2, T3.3, T3.4, and T3.6, with a focus on the period between August 2020 and August 2021. At first, we described the monitoring process regarding the needed computing and storage that is used in PolicyCLOUD in order to analyse a plethora of datasets from different data sources, and facilitate policy making. We also provided the process that will be followed for the registration of PolicyCloud Services in the EOSC Portal https://eosc-portal.eu/.

In the next period, both tasks T3.1 and T3.2 will be active and continue their role in the provisioning and monitoring of the PolicyCLOUD infrastructure and the actual registration of the developed services to the EOSC portal. In addition, a DPA will be created for the proper regulation of the personal data processing on integration, while the EGI AAI Check-in service will be integrated into the platform.

In this document, we reported the capabilities and mechanisms to be used by the components called "cloud gateways", which are responsible for obtaining data from heterogeneous data sources, and also provided the foundations for the incentives management activity in PolicyCLOUD. PolicyCLOUD provides a set of tools to policy makers to identify and manage incentives, so as to raise citizen consciousness and allow citizens to be part of an evidence-based policy making process. Finally, we examined different models and technologies and presented how data protection and privacy enforcement can be offered by an ABAC-based access control mechanism.

For all these tasks (T3.3, T3.4, and T3.6) that are producing software artefacts, we have the short term goal of creating the updated software prototypes and then proceed with the validation of the artefacts by the use cases and perform the relevant improvements and updates.

The second version of the software prototype of the services mentioned in this document will be provided by M22 (October 2021), while the final iteration of this report will be provided through document D3.7 (due in August 2022, M32).

References

[1] PolicyCLOUD, D3.2 - Cloud Infrastructure Incentives Management and Data Governance Software Prototype 1, 2020.

[2] Fair Principles, https://www.go-fair.org/fair-principles/.

[3] API Gateway pattern: The API gateway pattern versus the Direct client-to-microservice communication (https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-com), 2019.

[4] Netflix Zuul: Documentation, https://github.com/Netflix/zuul.

[5] Apigi, https://cloud.google.com/apigee..

[6] Amazon API Gateway, https://aws.amazon.com/api-gateway/.

[7] Spring Cloud Gateway, https://spring.io/projects/spring-cloud-gateway.

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

[9] J. Au-Yeung, Best practices for REST API design, https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/, 2020.

[10] Semantic Versioning, https://semver.org/.

[11] JWT, https://jwt.io/introduction/.

[12] MoleculerJS, https://moleculer.services/docs/0.14/.

[13] MoleculerJS Networking, https://moleculer.services/docs/0.14/networking.

[14] A. &. S. A. &. K. I. Katmada, Incentive Mechanisms for Crowdsourcing Platforms., 2016.

[15] PolicyCLOUD D3.1 - Cloud Infrastructure Incentives Management And Data Governance: Design And Open Specification 1, 2020.

[16] PolicyCLOUD, D3.3 – PolicyCLOUD's Societal and Ethical Requirements & Guidelines., 2020.

[17] W3C XML Schema Definition Language (XSD), https://www.w3.org/TR/xmlschema11-1/.

[18] Colebatch, HK, Making sense of governance. Policy and Society 33(4): 307–316., 2014.

[19] Micheli M, Ponti M, Craglia M, Berti Suman A. Emerging models of data governance in the age of datafication. Big Data & Society, 2020.

[20] Shkabatur, J, The global commons of data. Stanford Technology Law Review 22: 1–46., 2019.

[21] Ho, C, Chuangt, T, Governance of communal data sharing. In: Daly, A, Devitt, K, Mann, M (eds) Good Data. Amsterdam: Institute of Network Cultures, pp. 202–213., 2019.

[22] Delacroix, S, Lawrence, ND, Bottom-up data trusts: Disturbing the 'one size fits all' approach to data governance. International Data Privacy Law 9(4): 236–252., 2019.

[23] Ilves, LK, Osimo, D, A roadmap for a fair data economy. Policy Brief, Sitra and the Lisbon Council. Helsinki, Finland, 2019.