SPECIAL

Scalable Policy-awarE Linked Data arChitecture for prIvacy, trAnsparency and compLiance

Deliverable D3.2

Policy & events release

Document version: V1.0

SPECIAL DELIVERABLE

Name, title and organisation of the scientific representative of the project's coordinator:

Ms Jessica Michel t: +33 4 92 38 50 89 f: +33 4 92 38 78 22 e: jessica.michel@ercim.eu

GEIE ERCIM, 2004, route des Lucioles, Sophia Antipolis, 06410 Biot, France

Project website address: http://www.specialprivacy.eu/

Project	
Grant Agreement number	731601
Project acronym:	SPECIAL
Project title:	Scalable Policy-awarE Linked Data arChitecture for prIvacy, trAnsparency and compLiance
Funding Scheme:	Research & Innovation Action (RIA)
Date of latest version of DoW against which the assessment will be made:	17/10/2016
Document	
Period covered:	M01-M36
Deliverable number:	D3.2
Deliverable title	Policy & events release
Contractual Date of Delivery:	30-04-2018
Actual Date of Delivery:	30-04-2018
Editor (s):	Sabrina Kirrane, Javier D. Fernández, Axel Polleres (WU), Rigo Wenning (ERCIM), Rudy Jacob (PROXIMUS)
Author (s):	Wouter Dullaert, Jonathan Langens, Uroš Milošević (TF)
Reviewer (s):	Ben Whittam Smith (TR), Martin Kurze (TLabs)
Participant(s):	TF, WU, ERCIM, TR, TLabs, PROXIMUS
Work package no.:	3
Work package title:	Big Data Policy Engine
Work package leader:	TF
Distribution:	PU
Version/Revision:	1.0
Draft/Final:	Final
Total number of pages (including cover):	35

Disclaimer

This document contains description of the SPECIAL project work and findings.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any responsibility for actions that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the SPECIAL consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 28 Member States of the Union. It is based on the European Communities and the Member States cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors (http://europa.eu/).

SPECIAL has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731601.

Contents

1 Summary **7**

2 Architecture Overview **8**
2.1 Architecture . . . 8
2.2 Big Data Europe . . . 8
2.3 Apache Kafka . . . 9
2.4 Authentication and Authorization . . . 11
2.4.1 Authentication: OpenID Connect . . . 11
2.4.2 Authorization: OAuth2 . . . 15
2.4.3 Implementation . . . 16

3 Consent Management **17**
3.1 API Design . . . 17
3.1.1 Applications . . . 18
3.1.2 Users . . . 20
3.1.3 Policies . . . 21
3.1.4 Authorization . . . 24
3.2 Database Layer . . . 24
3.2.1 Document Store . . . 24
3.2.2 Streaming Queries . . . 24
3.3 Change Feeds . . . 25
3.3.1 Transaction Log . . . 25
3.3.2 Full Policy Log . . . 25

4 Compliance Checking **27**
4.1 Data Flow . . . 27
4.1.1 Application Log Topic . . . 28
4.1.2 Policies Topic . . . 28
4.1.3 Base Ontology . . . 29
4.2 Compliance Checking . . . 30
4.2.1 Application Log Flow . . . 30
4.2.2 Subsumption . . . 30
4.3 Scaling and Fault-Tolerance . . . 30

5 Transparency Dashboard **32**
5.1 Overview of Components 33
5.2 Current State 33

6 Discussion **34**

List of Figures

2.1 Prototype Architecture 9
2.2 Uncompacted Log 11
2.3 Compacted Log 11
2.4 OpenID Connect Authentication Flow 13
2.5 OpenID Connect Implicit Flow 15

3.1 Consent Management 18
4.1 Compliance Checker 27

5.1 Transparency Dashboard 32

Chapter 1

Summary

The goal of this report[1] is to describe the second release of the SPECIAL platform. It builds upon the research done in WP2 by providing working implementations of many of the ideas presented in deliverables D2.1 Policy Language V1, D2.3 Transparency Framework V1 and D2.4 Transparency and Compliance Algorithms V1. Nevertheless, it is worth noting that this is an initial implementation and that, as the work in WP2 progresses, alternative solutions are likely to be considered and implemented

The first chapter presents the platform architecture as a whole. This will give the reader an overview of the various supported features, how the individual components interact and detailed information on some cross cutting concerns.

In subsequent chapters specific components of the architecture are discussed in more detail. SPECIAL focus is placed on documenting design decisions which might not be obvious from the source code.

At the time of publishing a working version of the platform is available on `http://projects.tenforce.com/special/demo`.

[1] D3.2 Policy and Events Release is a DE (demonstrator) type deliverable.

Chapter 2

Architecture Overview

This chapter documents the overall architecture of the SPECIAL platform as it is currently envisioned. It documents the guiding design principles, and focus on cross cutting concerns and how data flows between the various components.

2.1 Architecture

A high level overview of the architecture is shown in Figure 2.1.

In its current state, four main components can be identified:

1. Existing line of business applications
2. Consent management service
3. Compliance checker service
4. Transparency service

Each of these services will be covered in more detail in their own chapters. All the components are integrated using message passing through Apache Kafka [1].

2.2 Big Data Europe

The architecture proposed here builds on the experience from the H2020 Big Data Europe (BDE) project [2].

BDE leveraged Docker technologies to simplify installing and running big data technologies. Software is packaged into Docker images for ease of distribution and composed into working systems using Docker Compose [3]. BDE leveraged Docker Swarm [4] to deploy a system onto a cluster of machines rather than a single machine.

The architecture proposed here follows all the best practices from BDE and the live prototype has been deployed using this tooling. This allows us to move the system from a single machine during development to a robust clustered deployment with ease.

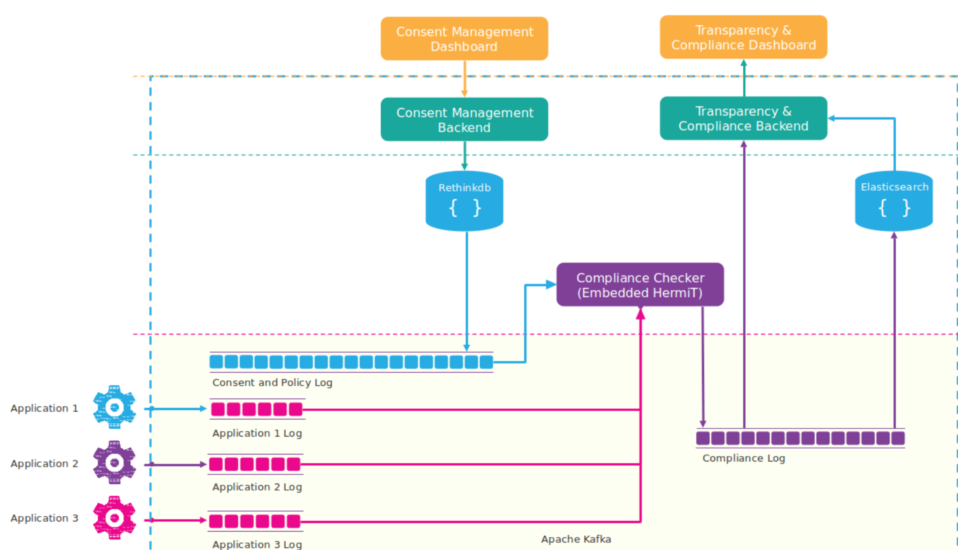

Figure 2.1: Prototype Architecture

2.3 Apache Kafka

Kafka [1], describes itself as a distributed streaming platform. It is easiest to think of it as a fault tolerant, append-only log. This is a very generic primitive to build robust distributed systems with.

Kafka has three main capabilities:

- Publish and subscribe to a stream of records (similar to a queue or Enterprise Service Bus (ESB))
- Store records in a fault-tolerant durable way (unlike a queue or ESB)
- Optionally process records as they occur using the Kafka Stream library

In the software system described in this deliverable, Kafka will be used as a datastore, but its data processing capabilities will not be leveraged. These will be handled using other software. This should make the approach less intrusive and make it easier for companies with existing line of business applications to adopt the platform.

Kafka has a few core abstractions:

- Kafka runs on a cluster of 1 or more servers, called *brokers*.
- Kafka stores *records* in categories called *topics*.
- Topics are subdivided into *partitions*.
- Records consist of a *key*, *offset* and *value*.

Unlike normal queueing systems, records in Kafka are persisted whether they are consumed or not. It is a kind of special purpose distributed filesystem dedicated for high-performance, low-latency commit log storage, replication, and propagation. How long records are persisted inside of Kafka is governed by a retention policy, which can be set on a topic by topic basis:

- **Time based retention**: records are kept for a certain period of time.
- **Size based retention**: records are kept in a topic until it reaches a certain size, after which the oldest records are purged until the storage quota has been met.
- **Log compaction**: Kafka ensures that at least 1 record for every key is present in the topic. Due to its importance, for the sake of clarity, log compaction is described in more detail below.

Assume there is a topic with product descriptions. Each time a product description is updated a new record is posted onto this topic with the `productId` as key and the product description as value. An example of such a topic with 6 elements is shown in Figure 2.2. In this picture the colour of the box represents the key of the record, the number below is the offset of the record.

When log compaction gets triggered, Kafka will remove all older messages for a given key, retaining only the latest one. This results in a log with "gaps" as shown in Figure 2.3. With log compaction, the size of the topic will be bounded, provided the size of the keyspace is bounded (no infinite number of products).

These various strategies give Kafka a lot of flexibility. Time based retention is great in an IOT scenario where individual records have a short half life. Size based retention is very useful in traditional queueing scenarios. Log compaction provides an elegant way to synchronise reference data between various systems.

For data consumption, Kafka combines the features of a queing system with a pub-sub system. Each consumer of data is part of a *consumer-group*. Every message on a topic will be sent to every consumer-group, implementing the broadcast behaviour of a pub-sub system. Within a consumer-group Kafka will assign the partitions of a topic to the individual consumers in the group. This allows processing of a topic to be scaled out horizontally, like what is possible with a queue and a worker pool. Because a partition can only be assigned to a single consumer within a consumer-group, the number of consumers in a consumer group can never be larger than the total number of partitions.

When compared with other storage systems, such as Hadoop, the advantage of Kafka is that it has the API of a pub-sub and queuing system. It allows us to treat data and data updates as immutable event and has well defined semantics for how to consume these, while in Hadoop's file oriented world most of the semantics need to be communicated out of band. (Are records updated in place? Are they appended to the bottom of the file?)

All these features make Kafka a very flexible data layer for our system:

- It can act as a buffer.
- It can minimize the coupling between the various components.

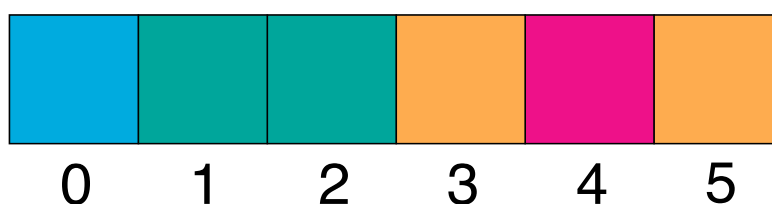

Figure 2.2: Uncompacted Log

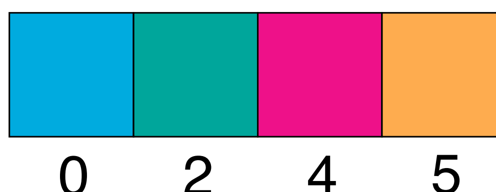

Figure 2.3: Compacted Log

- Its streaming nature allows the system to do near real time data processing, while still providing support for more batch oriented workload (you can always slow down from real time, but it's hard to speed batch jobs up to realtime).
- Its easy to understand and implement semantics make it easier to build robust and scalable data processing systems.

Combined with the fact that it is mature, well supported, and proven open source software in use by some of the largest companies in the world [11, 12, 14], the authors feel confident in its selection as the data substrate for the SPECIAL system.

2.4 Authentication and Authorization

In order to authenticate users, the SPECIAL platform relies on the OpenID Connect, [7] industry standard for authentication and OAuth2 for authorization, [6].

2.4.1 Authentication: OpenID Connect

Authentication is the process by which the user makes a claim about his identity and proves this claim. It can losely be described as "logging in". OpenID Connect is a protocol by which both native apps and web applications[1] can delegate the authentication to a 3rd party identity provider. It builds upon OAuth2 by combining various OAuth2 message flows into an authentication flow.

Before describing the two main OpenID Connect flows in more detail, some terms are introduced:

[1]Unlike the often used Security Assertion Markup Language 2.0 (SAML2) which only supports web applications.

- **Identity Provider (IDP)**: The party that offers authentication as a service. It is the service that will confirm the identity of the user (using e.g. passwords or two-factor authentication tokens). Examples of identity providers are Google, Facebook or a country's eID system.
- **Relying Party (RP)**: This party is the application which would like to establish the identity of a user. By implementing OpenID Connect it delegates this task to the IDP. The applications described in this deliverable act as RPs.
- **Claim**: This is information asserted by a user, such as name or email address.

OpenID Connect presents 3 flows:

1. **Authentication or Basic Flow**: This flow is useful for web and native applications with a trusted backend components.
2. **Implicit Flow**: This is flow is useful for web applications without a trusted backend component, such as single page web applications.
3. **Hybrid Flow**: This flow is a mix of the implicit flow and authentication flow. It is hardly ever used and won't be further discussed in this deliverable.

All OpenID Connect (and OAuth2) flows assume that all communication happens over Transport Layer Security (TLS) encrypted HTTP connections (HTTPS), preventing any secrets or tokens transmitted from being leaked to attackers which evesdrop on the network connection. This moves a lot of encryption and security complexity away from developers implementing these standards in their application, into the underlying infrastructure.

The following subsections will describe the Authentication Flow (2.4.1.1) and the Implicit Flow (2.4.1.2) at a relatively high level. The goal is for the reader to get an idea how these flows work, why they are secure and that they cover the authentication needs of the platform, without losing ourselves into too many implementation details.

A more detailed overview of the tradeoffs to consider when chosing the flow an application should use can be found in [10].

2.4.1.1 Authentication Flow

The *Authentication Flow* is the most secure, and most commonly used OpenID Connect flow. A call diagram is shown in Figure 2.4

The flow describes an interaction between the following parties:

- **User-agent**. This could be a client application written by the RP, but it can also be any other application that can speak to the backend. It is an untrusted party. In Figure 2.4, this is represented as the user.
- **Backend**. This is the Relying Party (RP) which wishes to authenticate a user. It is a trusted party.
- **IDP**. The Identity Provider to which the RP wants to delegate the proving of the identity of the user.

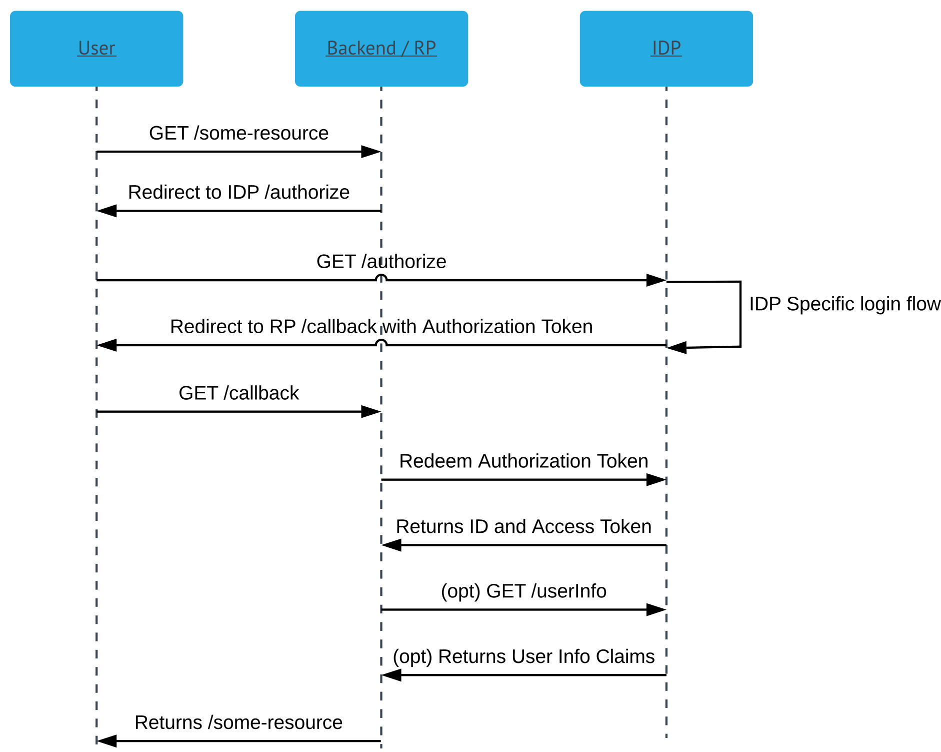

Figure 2.4: OpenID Connect Authentication Flow

The flow consists of the following steps:

1. The user requests a resource, or performs an action which requires him to be authenticated.
2. The RP notices that this is not an authenticated session and redirects the user to the `/authorize` of the IDP, embedding information about the RP.
3. The user follows the redirection and goes through the IDP login flow.
4. After the user has successfully followed the login flow the user is redirected to a callback at the RP. This callback embeds an *Authorization Token*.
5. The RP retrieves the *Authorization Token* from the callback and sends a request to the IDP to redeem it.
6. The IDP verifies that the *Authorization Token* is valid and issued for the RP that tries to redeem it and returns an *Access Token*, an *ID Token* and optionally a *Refresh Token*.
7. The RP can now optionally call the `/userInfo` endpoint at the IDP with the *Access Token* if it needs more user claims than those included in the *ID Token*.
8. If the token is valid and has the necessary grants, the IDP will return the claim (user information) to the RP.

At the end of this flow, the IDP will have produced 3 different tokens, each of which serves a different purpose:

- **Authorization Token**: Since the IDP does not have backchannel to talk to the RP directly, it needs to relay the results of the login flow through the untrusted user-agent. In order to prevent any confidential data from leaking, the IDP sends this single-use Authorization Token with a short time to live (typically less than an hour). This token can be exchanged only by the RP for the actual Identity and Access Tokens.
- **ID Token**: This token encodes the claims (user data such as email and name) the RP has requested from the end user. While it is theoretically possible to embed any user data the IDP has, most IDPs put limits here, requiring the RP to call the `/userInfo` endpoint for additional, more sensitive, information.
- **Access Token**: The access token grants its bearer the right to call the `/userInfo` at the IDP for specific user information. The RP can use this to retrieve additional user claims not included in the ID Token, but it can also pass this token in request to downstream services which might need to verify the end users identity. The token can optionally encode additional authorizations.
- **Refresh Token**: This token can be used to refresh the Identity and Access token. This allows the time to live on these tokens to be short (limiting potential damage should they get leaked), but allows the RP to renew them without forcing the end user to go through the login flow repeatedly.

The main advantage of OpenID Connect is that the IDP can introduce new and better ways of verifying the identity claim of the user (such as 2FA or biometric methods) without any code or logic changes in the RP. The protocol does not need to change.

2.4.1.2 Implicit Flow

The implicit flow is useful for single page applications without a trusted backend. A call diagram is shown in Figure 2.5. The flow describes an interaction between the following parties:

- **User-agent**: In this flow, the user-agent is typically the browser used to interact with a client side application. It is an untrusted party.
- **Client**: This is the Relying Party (RP) trying to establish the identity of the user. In this flow the RP is assumed to run in an untrusted environment.
- **IDP**: The Identity Provider to which the RP wants to delegate the proving of the identity of the user.

The flow consists of the following steps:

1. The user requests a resource, or performs an action which requires him to be authenticated.
2. The RP notices that this is not an authenticated session and redirects the user to the `/authorize` of the IDP, embedding information about the RP.

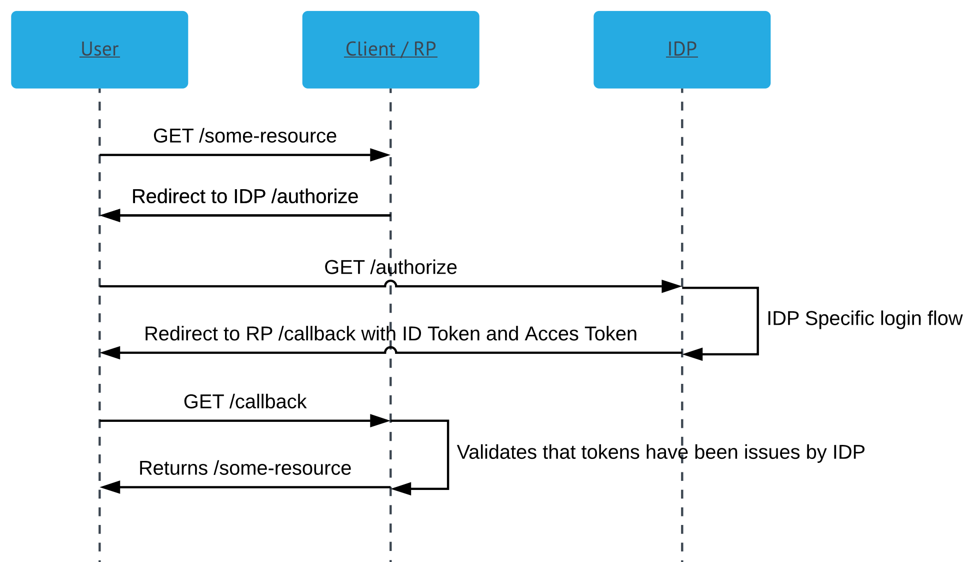

Figure 2.5: OpenID Connect Implicit Flow

3. The user follows the redirection and goes through the IDP login flow.
4. After finishing the login flow the IDP redirects the user to a callback at the RP. This callback embeds the *Identity Token* and an optional *Access Token*.

The flow is very similar to the Authentication flow described in Section 2.4.1.1, but in this case the RP is not running in a trusted environment. This means that the additional step of redeeming an authorization token adds no practical security: the ID and Access Token will end up in an untrusted environment anyway. Also because the RP runs in an untrusted environment, the IDP places less trust in it and will not issue a refresh token in this flow.

2.4.2 Authorization: OAuth2

OAuth2 is an authorization protocol. That means it concerns what a particular entity has access to rather than who that particular entity is. The specification describes a large amount of flows which can be implemented and that each have their own security tradeoffs. Because it can be very useful to know who an entity is when deciding what it has access to, quite a few OAuth2 flows also authenticate a user, but because the spec is focused on authorization, these aspects are often underspecified, leaving room for interpretation or custom implementations. This obviously gets in the way of interoperability and are the gaps that OpenID Connect aims to fill.

Because the OpenID Connect flows allow us to obtain authorization at the same time as authenticating a user, they currently satisfy the needs of this architecture. However in case a need for more intricate authorization flows presents itself, additional OAuth2 authorization flows can be introduced.

2.4.3 Implementation

In the demonstrator implementation Redhat Keycloak [5] has been selected as Identity provider and OpenID Connect / OAuth2 server. It is a fully featured Open Source product in use by companies big and small. Notable features of the product are:

- Federation of other identity providers through Active Directory or LDAP
- Federation of social logins such as Google or Facebook
- Broad support for authentication and authorization protocols such as SAML 2, OAuth2 and OpenID Connect

In case a company trying to adopt the system does not have an identity provider which supports OpenID Connect out of the box, Keycloak can be used to provide an OpenID Connect server without invasive changes to the existing landscape.

It is worth reiterating that while the demo system uses Keycloak, there is no strict requirement on it. The system has standardised on the OpenID Connect and OAUth2 protocols, not this particular implementation.

Chapter 3

Consent Management

This chapter describes the backend of the consent management service. Even though the demonstrator includes a frontend, the frontend / UX discussions are handled in WP4 and, more specifically, D4.1 Transparency Dashboard and Control Panel Release V1. The included frontend is just there to make it easier to present and evaluate the backend features.

The purpose of this service is to provide data subjects and data controllers a way to manage their policies. These type of services are commonly refered to as CRUD services: they need to (C)reate, (R)ead, (U)pdated and (D)elete data entities. An architecture that is commonly used to implement CRUD services, augmented with an audit log, has been chosen:

- An API Layer which allows frontends and other clients to call its services. Data validation and authorization checks happen here as well (Section 3.1).
- A database layer which persists the data in a format which is optimized for use by the API Layer (Section 3.2).
- Audit logs which record all transactions (Section 3.3).

3.1 API Design

The consent management API allows the manipulation of 3 different entities:

1. Applications
2. Users
3. Policies

Each of these entities is manipulated in a similar way. The following subsections will briefly describe the various endpoints and show some example payloads. This is not a full nor a final API specification, but should give the interested reader a decent understanding of how the API should be used. It is highly likely that specifics of the API will change as the platform evolves: this is very much a work in progress and no backward compatibility guarantees are given.

The API calls which allow for the retrieval of policy data are intended for use by UI clients

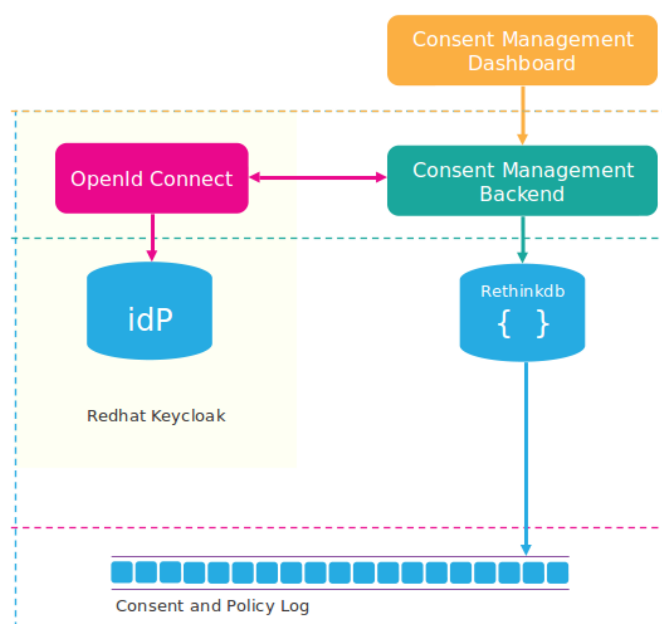

Figure 3.1: Consent Management

which wish to render an individual users policies. Services which require much more intensive use of policy data, such as the compliance checker or a potential authorization server, should preferably consume the policies from the full policy Kafka topic (see Section 3.3). This provides better decoupling, relaxes performance requirements on this service and provides consuming services with the option of reshaping the policy data to better fit their needs.

3.1.1 Applications

The `/applications` endpoints allow applications and their associated policies to be registered with the system.

- GET `/applications`
  Returns a list of all currently registered applications.

  Example Response:

```
[
  {
    "id": "d5aca7a6-ed5f-411c-b927-6f19c36b93c3",
    "name": "invoicer",
    "links": {
      "policies": "/applications/d5aca7a6-ed5f-411c-b927-6
          f19c36b93c3/policies"
    }
  },
  {
    "id": "58916c9e-3ce2-4fdb-94a4-369525582e75",
```

```
    "name": "marketing-machine",
    "links": {
      "policies": "/applications/58916c9e-3ce2-4fdb-94a4-369525582
          e75/policies"
    }
  }
]
```

- POST `/applications`
  Creates a new application. Returns the created application with its generated ID if the request was sucessful.

  Example Request:

```
{
  "name": "new-application"
}
```

  Example Response:

```
{
  "id": "ca775d49-c3a3-4e08-9e6a-9ac49612ad62",
  "name": "new-application"
  "links": {
    "policies": "/applications/ca775d49-c3a3-4e08-9e6a-9
        ac49612ad62/policies"
  }
}
```

- GET `/applications/:id`
  Returns a single registered application.

  Example Response:

```
{
  "id": "d5aca7a6-ed5f-411c-b927-6f19c36b93c3",
  "name": "invoicer",
  "links": {
    "policies": "/applications/d5aca7a6-ed5f-411c-b927-6
        f19c36b93c3/policies"
  }
}
```

- PUT `/applications/:id`
  Updates a single registered application. Returns the updated data.

  Example Request:

```
{
  "id": "d5aca7a6-ed5f-411c-b927-6f19c36b93c3",
  "name": "accounting"
}
```

Example Response:


```
{
  "id": "d5aca7a6-ed5f-411c-b927-6f19c36b93c3",
  "name": "accounting",
  "links": {
    "policies": "/applications/d5aca7a6-ed5f-411c-b927-6
       f19c36b93c3/policies"
  }
}
```


- DELETE `/applications/:id`
  Deletes a single registered application. Takes no payload and returns no data.

- GET `/applications/:id/policies`
  Returns the policies associated with a single application.

  Example Resonpse:


```
{
  "policies": [
    "d5bbb4cc-59c0-4077-9f7e-2fad74dc9998",
    "54ff9c00-1b47-4389-8390-870b2ee9a03c",
    "d308b593-a2ad-4d9f-bcc3-ff47f4acfe5c",
    "fcef1dbf-7b3d-4608-bebc-3f7ff6ae4f29",
    "be155566-7b56-4265-92fe-cb474aa0ed42",
    "8a7cf1f6-4c34-497f-8a65-4c985eb47a35"
  ]
}
```


3.1.2 Users

The `/users` endpoints allow the policies of individual data subjects to be retrieved and modified. Registration of data subjects with the system is handled by the identity provider, so the API does not provide any specific endpoints for these actions.

- GET `/users/:id`
  Return a single user information and its policies.

  Example Response:


```
{
  "id": "9b84f8a5-e37c-4baf-8bdd-92135b1bc0f9",
  "name": "Bernard Antoine",
  "links": {
    "policies": "/users/9b84f8a5-e37c-4baf-8bdd-92135b1bc0f9/
       policies"
  }
}
```


- PUT `/users/:id`
  Update a single user information and its policies

Example Request:

```
{
  "id": "9b84f8a5-e37c-4baf-8bdd-92135b1bc0f9",
  "policies": [
    "d5bbb4cc-59c0-4077-9f7e-2fad74dc9998",
    "0cb2b717-a442-4da5-818c-1c1c2e762201"
  ]
}
```

Example Response:

```
{
  "id": "9b84f8a5-e37c-4baf-8bdd-92135b1bc0f9",
  "name": "Bernard Antoine",
  "links": {
    "policies": "/users/9b84f8a5-e37c-4baf-8bdd-92135b1bc0f9/
       policies"
  }
}
```

- GET `/users/:id/policies`
  Returns the policies associated with a particular user.

  Example Response:

```
{
  "policies": [
    "d5bbb4cc-59c0-4077-9f7e-2fad74dc9998",
    "0cb2b717-a442-4da5-818c-1c1c2e762201"
  ]
}
```

3.1.3 Policies

- GET `/policies`
  Returns a list of all policies currently registered in the system.

  Example Response:

```
[
  {
    "id": "d5bbb4cc-59c0-4077-9f7e-2fad74dc9998",
    "dataCollection": "http://www.specialprivacy.eu/vocabs/data#
       Anonymized",
    "locationCollection": "http://www.specialprivacy.eu/vocabs/
       data#EU",
    "processCollection": "http://www.specialprivacy.eu/vocabs/
       data#Collect",
    "purposeCollection": "http://www.specialprivacy.eu/vocabs/
       data#Account",
```

```
    "recipientCollection": "http://www.specialprivacy.eu/vocabs/
       data#Delivery",
    "explanation": "I consent to the collection of my anonymized
       data in Europe for the purpose of accounting."
  },
  {
    "id": "54ff9c00-1b47-4389-8390-870b2ee9a03c",
    "dataCollection": "http://www.specialprivacy.eu/vocabs/data#
       Derived",
    "locationCollection": "http://www.specialprivacy.eu/vocabs/
       data#EULike",
    "processCollection": "http://www.specialprivacy.eu/vocabs/
       data#Copy",
    "purposeCollection": "http://www.specialprivacy.eu/vocabs/
       data#Admin",
    "recipientCollection": "http://www.specialprivacy.eu/vocabs/
       data#Same",
    "explanation": "I consent to the copying of my derived data
       in Europe-like countries for the purpose of administration
       ."
  }
]
```


- POST `/policies`
  Creates a new policy. Returns the created policies if the operation was successful.

  Example Request:


```
{
  "dataCollection": "http://www.specialprivacy.eu/vocabs/data#
     Computer",
  "locationCollection": "http://www.specialprivacy.eu/vocabs/data
     #ThirdParty",
  "processCollection": "http://www.specialprivacy.eu/vocabs/data#
     Move",
  "purposeCollection": "http://www.specialprivacy.eu/vocabs/data#
     Browsing",
  "recipientCollection": "http://www.specialprivacy.eu/vocabs/
     data#Public",
  "explanation": "I consent to the moving of my computer data on
     third-party servers for the purpose of browsing."
}
```


  Example Response:


```
{
  "id": "d308b593-a2ad-4d9f-bcc3-ff47f4acfe5c",
  "dataCollection": "http://www.specialprivacy.eu/vocabs/data#
     Computer",
  "locationCollection": "http://www.specialprivacy.eu/vocabs/data
     #ThirdParty",
  "processCollection": "http://www.specialprivacy.eu/vocabs/data#
     Move",
```

```
  "purposeCollection": "http://www.specialprivacy.eu/vocabs/data#
     Browsing",
  "recipientCollection": "http://www.specialprivacy.eu/vocabs/
     data#Public",
  "explanation": "I consent to the moving of my computer data on
     third-party servers for the purpose of browsing."
}
```


- GET `/policies/:id`
  Returns an individual policy.

  Example Response:


```
{
  "id": "d5bbb4cc-59c0-4077-9f7e-2fad74dc9998",
  "dataCollection": "http://www.specialprivacy.eu/vocabs/data#
     Anonymized",
  "locationCollection": "http://www.specialprivacy.eu/vocabs/data
     #EU",
  "processCollection": "http://www.specialprivacy.eu/vocabs/data#
     Collect",
  "purposeCollection": "http://www.specialprivacy.eu/vocabs/data#
     Account",
  "recipientCollection": "http://www.specialprivacy.eu/vocabs/
     data#Delivery",
  "explanation": "I consent to the collection of my anonymized
     data in Europe for the purpose of accounting."
}
```


- PUT `/policies/:id`
  Updates an individual policy. The updated policy is returned if the operation was successful.

  Example Request:


```
{
  "id": "d5bbb4cc-59c0-4077-9f7e-2fad74dc9998",
  "locationCollection": "http://www.specialprivacy.eu/vocabs/data
     #EULike"
}
```


  Example Response:


```
{
  "id": "d5bbb4cc-59c0-4077-9f7e-2fad74dc9998",
  "dataCollection": "http://www.specialprivacy.eu/vocabs/data#
     Anonymized",
  "locationCollection": "http://www.specialprivacy.eu/vocabs/data
     #EULike",
  "processCollection": "http://www.specialprivacy.eu/vocabs/data#
     Collect",
  "purposeCollection": "http://www.specialprivacy.eu/vocabs/data#
     Account",
```

```
  "recipientCollection": "http://www.specialprivacy.eu/vocabs/
     data#Delivery",
  "explanation": "I consent to the collection of my anonymized
     data in Europe for the purpose of accounting."
}
```

- DELETE `/policies/:id`
  Removes an individual policy. References to this policy are also removed from applications and users. Takes no payload and returns no data.

3.1.4 Authorization

The consent management service relies on the OpenID Connect Authentication Flow (see 2.4.1). This is the most secure OpenID Connect flow. The consent management service uses the data from the ID Token to bootstrap a user in the system, if it does not yet exist. This is why no POST `/users` and DELETE `/users/:id` endpoints exist. The entire lifecycle of a user is outsourced to the identity provider.

Similarly the information contained in the ID Token is used to filter data to just the data from the authenticated user.

3.2 Database Layer

For the database layer Rethinkdb, [8], has been chosen. It is an open source document database, with first class streaming support that is maintained by the linux foundation, [13]. The choice for Rethinkdb is not critical for the SPECIAL platform, most databases can be easily used for CRUD services, but Rethinkdb does offer a few features which are beneficial.

3.2.1 Document Store

Rethinkdb is a document store which can persist native json. Because our API layer communicates using json, this aspect minimizes the impedance mismatch between the two parts. The document model is also very flexible and allows us to easily modify the schema of the data, which is much more likely to happen at this early stage of development. A downside of Rethinkdb is that it does not offer multi document transanctions. It does offer document level atomicity: an update to a document is either completely written or it is not written. With careful document design this level of consistency is sufficient for our purposes.

3.2.2 Streaming Queries

Rethinkdb offers first class support for streaming queries. In most databases queries will return the matching data at the point the query was issued. If the application is interested in updates on this data it will need to poll the database, by regularly issuing the query again. Rethinkdb on the other hand allows a client to subscribe to a query. When the subscription starts Rethinkdb will return the results of the query as usual, but when changes to the database happen, which impact the results of the query, Rethinkdb will push the delta between the original query result and the current query result to the client.

This feature makes it very easy to implement the audit log. A query can be created which returns the data in exactly the right shape. The data can then be written onto Kafka for long term persistance as it arrives.

3.3 Change Feeds

The consent management service provides two change feeds:

1. Transaction Log
2. Full Policy Log

The consent management service is the only service which can write data to either of these logs. The access control mechanisms in Kafka are used to enforce this. The logs allow the reconstruction of the current state by replaying them in their entirety, but they do serve distinct purposes which are described in more detail in the following subsections.

3.3.1 Transaction Log

The transaction log is retained for audit purposes. It logs every command sent by a client. It could be described as the log of the intent of the user. It is strictly ordered and contains only the differences between two states. For example (this is not the actual format used on the log, but a more human readable version):

```
SET "3bd2731b-2361-4de6-b0e5-dd12e64827a9" [{"id": "3bd2731b-2361-4
    de6-b0e5-dd12e64827a9", "purpose": "charity"}]
```

It can be used to figure out who changed what data at a particular point in time. It also allows the state of the consent management service to be reconstructed at any particular point in time, by replaying the log in a fresh instance until that timestamp. This can be useful for what-if analysis or proving to an auditor that particular processing was lawfull at a particular time.

3.3.2 Full Policy Log

The full policy log is retained for integration purposes. It is a compacted Kafka topic (see Section 2.3): only the latest version of the policies of a data subject are retained. This makes it easy for services like the compliance checker to bootstrap their own materialised view of the policies and consume updates to those policies, without placing any load on the consent management service.

The data subject ID is stored in key of the record. The record value is a json represenation of the data subject policies. For example:

```
{
  "simplePolicies":[
    {
      "data":"http://www.specialprivacy.eu/vocabs/data#Anonymized",
      "processing":"http://www.specialprivacy.eu/langs/usage-policy#
          AnyProcessing",
```

```
      "purpose":"http://www.specialprivacy.eu/langs/usage-policy#
         AnyPurpose",
      "recipient":"http://www.specialprivacy.eu/langs/usage-policy#
         AnyRecipient",
      "storage":"http://www.specialprivacy.eu/langs/usage-policy#
         AnyDuration"
    },{
      "data":"http://www.specialprivacy.eu/vocabs/data#AnyData",
      "processing":"http://www.specialprivacy.eu/langs/usage-policy#
         AnyProcessing",
      "purpose":"http://www.specialprivacy.eu/langs/usage-policy#
         Charity",
      "recipient":"http://www.specialprivacy.eu/langs/usage-policy#
         AnyRecipient",
      "storage":"http://www.specialprivacy.eu/langs/usage-policy#
         AnyDuration"
    }
  ]
}
```

Because this log is compacted, it cannot be used to reconstruct the state of the consent management system at an arbitrary point in the past.

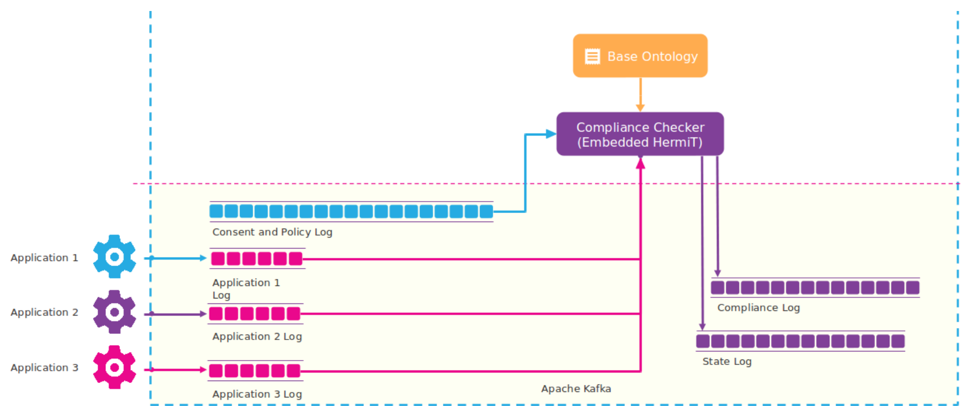

Figure 4.1: Compliance Checker

Chapter 4

Compliance Checking

This chapter describes the compliance checker service in more detail. Its purpose is to validate that application logs are compliant with a users policy. These application logs are delivered in the format described in deliverable D2.3, the policies are an implementation of the policy language described in deliverable D2.1.

Figure 4.1 shows an overview of the components that will be discussed in this section.

4.1 Data Flow

The compliance checker can be seen as a stream processor which takes in a stream of application logs and emits an augmented stream of logs. The system has the following data inputs:

- **Application Log Topic**: This is a normal Kafka topic that contains all application logs which need to be checked for compliance.
- **Policies Topic**: This is a compacted Kafka topic which holds the complete policies for all data subjects.

- **Base Ontology**: The base ontology are the vocabularies and class relationships which define the policy language as described in deliverable D2.1.

The system has the following outputs:

- **Compliance Topic**: This is a normal Kafka topic which contains the augmented application logs.
- **State Topic**: This is a compacted Kafka topic where the compliance checker can checkpoint the latest offset is has processed. This allows it easily restore its state in case it needs to restart.

4.1.1 Application Log Topic

The application log topic contains, as the name implies, the logs produced by the various line of business applications in the broader ecosystem. The compliance checker assumes that the logs are written in, or have been transformed into, the json serialization of the format described in D2.3. An example log can look as follows:

```
{
  "timestamp": 1524223395245,
  "process": "send-invoice",
  "purpose": "http://www.specialprivacy.eu/vocabs/purposes#Payment",
  "processing": "http://www.specialprivacy.eu/vocabs/processing#Move
     ",
  "recipient": "http://www.specialprivacy.eu/langs/usage-policy#
     AnyRecipient",
  "storage": "http://www.specialprivacy.eu/vocabs/locations#
     ControllerServers",
  "userID": "49d40b22-4337-4652-b463-41b1c23c6b08",
  "data": [
    "http://www.specialprivacy.eu/vocabs/data#OnlineActivity",
    "http://www.specialprivacy.eu/vocabs/data#Purchase", "http://www.
       specialprivacy.eu/vocabs/data#Financial"
  ]
}
```

In a future version of the service, a jsonld context will most likely be added to this file. Alternatively, a turtle serialization could be used to make parsing the logs easier.

Records on this topic use the data subject ID as a key, so that the data can be easily partitioned by data subject. This is helpfull when scaling up the work, see Section 4.3.

4.1.2 Policies Topic

The policies topic is a compacted Kafka topic which contains full policies for all data subjects. Its content is produced by the consent management service, described in 3. The records have the data subject ID as a key, so that the data can be easily partitioned by data subject. This is helpfull when scaling up the work, see Section 4.3.

Data subject policies are stored in a json serialization, an example of which can look as follows:

```
{
  "simplePolicies":[
    {
      "data":"http://www.specialprivacy.eu/vocabs/data#Anonymized",
      "processing":"http://www.specialprivacy.eu/langs/usage-policy#
          AnyProcessing",
      "purpose":"http://www.specialprivacy.eu/langs/usage-policy#
          AnyPurpose",
      "recipient":"http://www.specialprivacy.eu/langs/usage-policy#
          AnyRecipient",
      "storage":"http://www.specialprivacy.eu/langs/usage-policy#
          AnyDuration"
    },{
      "data":"http://www.specialprivacy.eu/vocabs/data#AnyData",
      "processing":"http://www.specialprivacy.eu/langs/usage-policy#
          AnyProcessing",
      "purpose":"http://www.specialprivacy.eu/langs/usage-policy#
          Charity",
      "recipient":"http://www.specialprivacy.eu/langs/usage-policy#
          AnyRecipient",
      "storage":"http://www.specialprivacy.eu/langs/usage-policy#
          AnyDuration"
    }
  ]
}
```


When a record is read from the topic, it is transformed into an OWL XML representation and saved to a temporary file, which can be indexed by the subject ID. When a new version of the policy for a particular data subject is read from Kafka, the existing temporary file is completely overwritten. In a future version of this service, this file based index will be replaced with a (potentially in-memory) key-value store.

4.1.3 Base Ontology

The base ontology is a collection of OWL statements which describe the various classes and their relationships, which are used to express policies. Without these definitions the OWL reasoner does not know how to make sense of any policies. The base ontology is saved in OWL XML format and loaded from disk at startup. These files are shipped together with the binary. In the current version of the policy checker, there is no way to load additional or different versions of the base ontology, they are effectively hard coded.

A planned improvement is to turn the base ontology into data which the compliance checker consumes from a Kafka topic. This will allow companies to more easily update the base ontology, or define additional vocabularies which define company specific attributes to use in policies.

4.2 Compliance Checking

4.2.1 Application Log Flow

When a compliance checker instance starts up, it initialises itself by first loading the base ontology into memory.

The compliance of each application log is checked by going through the following steps:

1. Clone the base ontology into a new `OWLOntology`
2. Read the data subject ID from the key of the application log
3. Lookup the data subject policy by the data subject ID
4. Load the policy into the cloned `OWLOntology`
5. Peform the subsumption check (see Section 4.2.2)
6. Add the check result to the application log data structure
7. Write the augmented application log to the Compliance Kafka topic
8. Discard the cloned `OWLOntology`

It is worth noting that application log is never persisted by the compliance checker and only retained in memory for the duration of the subsumption check. Because the `OWLOntology` used for the subsumption check only contains a small amount of information, these checks are evaluated very quickly.

4.2.2 Subsumption

As can be seen in deliverables D2.1 and D2.3, it can be verified that an application log is compliant with a data subject policy by performing a subsumption check. The subsumption algorithm used by the compliance checker is OWL API 3.4.3[1] compliant. It creates an `OWLSubclassOfAxiom` that takes 2 `OWLClasses` and returns an `OWLAxiom` object that states that the first class is a superclass of the second class.

This is then passed on to the `isEntailed` method which returns `true` or `false`. The implementation of the reasoner is HermiT but this should be easily swapped with any other OWL API 3.4.3 compliant reasoner.

4.3 Scaling and Fault-Tolerance

Section 4.2 details how an individual application log flows through the compliance checker and is validated. This section zooms out a bit and details how the compliance checker can scale to support a load beyond what a single instance can handle. Because scaling the computation to multiple instances turns the compliance checker into a distributed system, it is also important to look at how the processing will handle the various failures which will inevitably occur.

[1] http://owlcs.github.io/owlapi/

Since all of the data being processed by the compliance checker is stored on Kafka, it is important to understand the primitives it gives us to build a distributed stream processing system (see also Section 2.3). Topics in Kafka are divided into partitions. Partitions are the actual log structures which are persisted on disk. As a result ordering between records is only guaranteed within a partition. If the record producer does not specify a partition explicitly, Kafka decides which partition a record gets written to by using a partition function, which can take the key of the record into account. Because a partition can only be assigned to a single consumer in a consumer group, the number of partitions puts an upper limit to how far the processing of records can be scaled out.

The total number of partitions of the application log topic will decide how many instances of the compliance checker can process the data in parallel. Because the records are assigned to a partition based on the data subject ID, it is guaranteed that an individual compliance checker instance will see all logs about a particular data subject in the order they occured. This is currently not necessary for the compliance algorithm to work, but keeps the option open to take into account multiple log messages to make decisions about compliance.

Kafka automatically assigns partitions to individual consumers, and will rebalance the partition assignment in case consumers get added or removed from the consumer group. This ensures that all messages get processed and that each consumer gets a fair share of the work, even if individual consumers fail.

In case of catastrophic failure, where all consumers die, the last processed offsets per partition can be recovered from the state topic. This prevents the new compliance checker instances from redoing work which was already done previously. Provided the new instances are spawned before the existing log messages fall out of retention, this catastrophic failure will also not result in data loss.

It is also worth noting that in order to scale out the work and provide fault tolerance, no other functionality, other than a few primities provided by Kafka, is being relied upon. There are no restrictions on how the compliance checker is programmed or deployed: no special libraries or resource scheduler is required. In fact the compliance checker looks just like any other java application from an operational perspective. This is in stark contrast with data processing frameworks like Spark, which require that the processing is implemented in their own specific framework and deployed on dedicated clusters using specific resource managers.

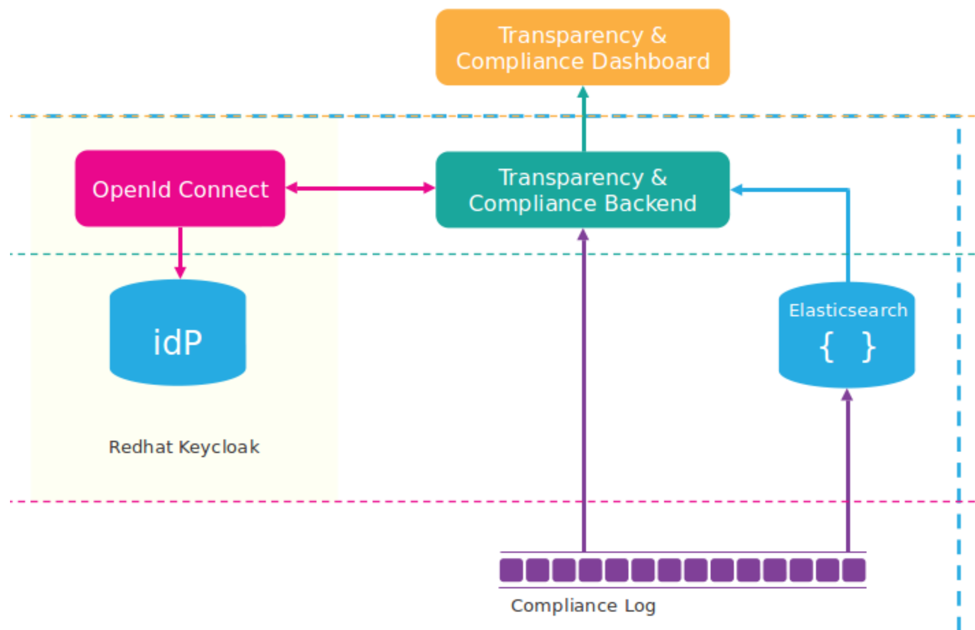

Figure 5.1: Transparency Dashboard

Chapter 5

Transparency Dashboard

This chapter covers the backend of the transparency dashboard. Frontend / UX concerns are handled in WP4 and D4.1.

At the moment the transparency backend is still fairly minimal. The focus for this deliverable has been on the consent management, compliance checking and overall integration mechanisms. Figure 5.1 shows the current architecture proposal. The actual results will be documented in deliverable D3.4.

5.1 Overview of Components

As can be seen in Figure 5.1, the proposal for the transparency service will consist of the following components

- **Compliance Log**: This is the output of the compliance checker and will serve as the reference for any visualisations
- **Elasticsearch**: Elasticsearch will contain an indexed version of the compliance log and will provide faceted browsing, easy lookups and aggregations.
- **Transparency Backend**: The transparency backend will act as the sole entrypoint for any UIs built as part of deliverable 4.1. It will provide access control and enforce authorized access to the data in elasticsearch or the compliance log.

5.2 Current State

The version of the transparency service which is currently available, consists of the following components:

- **Compliance Log**: This is the output of the compliance checker
- **Transparency Backend**: A service which exposes the compliance log as server sent events, [9], to a web client
- **Placeholder Dashboard**: A temporary dashboard which visualises the events on the compliance log in real time

These components prove that it is possible to stream a Kafka topic in real time in a web client. Any other features expected to present in the final solution, such as access control and faceted browsing, have not yet been implemented.

Chapter 6

Discussion

As previously stated, this is only an early implementation of the SPECIAL platform architecture. As we gain new insights in WP2 and receive more feedback from WP5 and WP6, we will identify elements in the architecture and the implementation that are to be augmented or replaced.

For instance, even though HermiT is currently fulfilling its purpose, its performance is yet to be benchmarked against the reasoning capabilities of the SANSA-Stack, as well as the SPECIAL algorithm being developed in WP2.

Further points of investigation could include harmonizing the data representation in motion and at rest under a unifying data model and a single data serialization format (such as JSON-LD) across the architecture, as well as globally unique and dereferenceable URIs. This will also require further evaluation of possible data storage solutions.

Another topic of interest would be adding additional API endpoints, for example, to consult and/or manipulate the policy metadata such as the taxonomies used to describe the data and processing types, storage locations, or purposes.

We might also want to settle on a way to document the historical changes to data subject's policies in RDF.

Bibliography

[1] Apache kafka. URL https://kafka.apache.org/.

[2] Big data europe. URL https://www.big-data-europe.eu/.

[3] Docker compose, . URL https://docs.docker.com/compose/.

[4] Docker swarm mode, . URL https://docs.docker.com/engine/swarm/.

[5] keycloak. URL https://www.keycloak.org/.

[6] Rfc 6749: The oauth 2.0 authorization framework. URL https://tools.ietf.org/html/rfc6749.

[7] Openid connect core 1.0. URL https://openid.net/specs/openid-connect-core-1_0.html.

[8] Rethinkdb. URL https://rethinkdb.com.

[9] Server sent events. URL https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events.

[10] R. Broeckerlmann. When to use which (oauth2) grants and (oidc) flows, 2017. URL https://medium.com/@robert.broeckelmann/when-to-use-which-oauth2-grants-and-oidc-flows-ec6a5c00d864.

[11] L. Engineering. Running kafka at scale, 2015. URL https://engineering.linkedin.com/kafka/running-kafka-scale.

[12] N. Engineering. Kafka inside keynote pipeline, 2016. URL https://medium.com/netflix-techblog/kafka-inside-keystone-pipeline-dd5aeabaf6bb.

[13] M. Glukhovsky. Rethinkdb joins the linux foundation, 2017. URL https://rethinkdb.com/blog/rethinkdb-joins-linux-foundation/.

[14] B. Svingen. Publishing with apache kafka at the new york times, 2017. URL https://www.confluent.io/blog/publishing-apache-kafka-new-york-times/.