BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FEDERATED SERVICES MANAGEMENT APPLICATION

APLIKACE PRO SPRÁVU FEDEROVANÝCH SLUŽEB

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **JAN PAVLÍČEK**
AUTOR PRÁCE

SUPERVISOR **Mgr. KAMIL MALINKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

164372

Institut:	Department of Intelligent Systems (DITS)
Student:	**Pavlíček Jan**
Programme:	Information Technology
Title:	**Federated Services Management Application**
Category:	Security
Academic year:	2024/25

Assignment:

1. Learn about federated login technologies such as Security Assertion Markup Language (SAML) and OpenID Connect (OIDC).
2. Learn about the architecture and technologies of the RCIAM Federation Registry application for registering and managing services.
3. Design extensions to the RCIAM Federation Registry application to include at least three selected functionalities from the following list:

   i) a new service model - the service will combine some appropriate features of the clients that will be interfaced with it,

   ii) extension of the agent set with a configurable agent, for forwarding SAML and OIDC client registration data to Apereo Central Authentication Service (CAS),
   iii) interfacing with an access control management system called Perun,
   iv) providing propagation support from multiple integration environments into a single proxy,
   v) enable SAML registration by manually filling in service information,

   vi) support of service registration for "just in case" provisioning by Perun, Support of automatic approval of URL path changes for an already approved domain.
4. Implement selected suggestions from the previous point in the relevant technologies. Document the implementation properly.
5. Perform a deployment to a test server and test the implementation.

Literature:

- RCIAM Federation Registry: https://github.com/rciam/rciam-federation-registry
- openID Specifications https://openid.net/developers/specs/
- SAML V2.0 Specifications: https://saml.xml.org/saml-specifications

Requirements for the semestral defence:
Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Malinka Kamil, Mgr., Ph.D.**
Consultant:	Břoušek Pavel
Head of Department:	Kočí Radek, Ing., Ph.D.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	31.10.2024

Abstract

This thesis extends the Federation Registry application to improve integration with modern identity infrastructures. It introduces support for Perun AAI, enabling automatic synchronization of services with Perun IDM, and adds a deployer agent for Apereo CAS. A feature for service registration from all deployment environments into a single identity infrastructure instance was also implemented. All extensions were tested in a real-world-like environment and demonstrated improved interoperability and flexibility for service registration within academic and research federations.

Abstrakt

Tato práce rozšiřuje aplikaci Federation Registry o podporu moderních infrastruktur pro správu identit. Přidává integraci s Perun AAI pro automatickou synchronizaci služeb s Perun IDM a nového nasazovacího agenta pro registraci služeb do systému Apereo CAS. Součástí je i podpora pro registraci služeb ze všech integračních prostředí do jedné instance infrastruktury identit. Všechna rozšíření byla otestována v prostředí simulujícím reálný provoz a prokázala zlepšení interoperability a flexibility při registraci služeb v akademických a výzkumných federacích.

Keywords

Federation Registry, Perun AAI, Apereo CAS, identity management, service registration, federated identity

Klíčová slova

Federation Registry, Perun AAI, Apereo CAS, správa identit, registrace služeb, federovaná identita

Reference

PAVLÍČEK, Jan. *Federated Services Management Application*. Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Rozšířený abstrakt

V dnešní digitální době je správa digitálních identit klíčovým prvkem zajišťujícím bezpečný a efektivní přístup k online službám. Zvláště v prostředí akademických a výzkumných institucí nabývá na důležitosti koncept federované správy identit, která umožňuje uživatelům přistupovat k různým službám v rámci federace pomocí jediné identity vydané jejich domovskou institucí. Tím se zjednodušuje uživatelská zkušenost, snižuje administrativní zátěž a zároveň se zvyšuje bezpečnost celého systému. Aby bylo možné tyto identity efektivně využívat, je nutné zajistit nástroje pro správu registrace služeb do infrastruktury.

Jedním z takových nástrojů je webová aplikace Federation Registry, kterou vyvíjí řecká organizace GRNET. Aplikace zjednodušuje proces registrace služeb podporujících protokoly SAML a OpenID Connect do federovaných infrastruktur. I přesto, že Federation Registry již podporuje několik platforem proxy/IdP (například Keycloak, MITREid nebo SimpleSAMLphp), její funkčnost nebyla dostatečná pro specifické potřeby infrastruktur postavených nad Perun AAI. Cílem této práce proto bylo rozšíření schopností Federation Registry tak, aby podporovala integraci s Perun AAI a také přidání podpory pro další proxy/IdP řešení Apereo CAS.

V první fázi byla provedena analýza základních konceptů správy identit, používaných autentizačních a autorizačních protokolů, a rovněž architektury systému Federation Registry. Byly identifikovány klíčové komponenty systému a analyzovány jeho registrační procesy. Na základě získaných poznatků byly navrženy tři hlavní rozšíření.

Prvním z nich je rozšíření nasazovacích agentů Federation Registry o podporu registrace služeb do Perun IDM, komponenty systému Perun AAI pro správu identit a přístupů. Po úspěšné registraci služby do proxy/IdP je prováděna synchronizace s Perun IDM, která zahrnuje vytvoření odpovídající entity (facilita) reprezentující službu, přiřazení atributů a vytvoření správcovské skupiny. Tato funkcionalita je plně konfigurovatelná a implementována jako rozšiřitelný modul.

Druhým přínosem je implementace podpory pro slučování registračních prostředí (např. vývojové, testovací, produkční) do jedné instance insfrastruktury identit. Toto rozšíření reflektuje požadavek, aby všechny služby bez ohledu na prostředí byly registrovány do jedné instance proxy/IdP a Perun IDM. Tím se výrazně snižuje náročnost na správu infrastruktury.

Třetím rozšířením je vývoj nového nasazovacího agenta pro Apereo CAS, moderního a flexibilní řešení poskytovatele identit, které je plánováno jako náhrada za současnou proxy vrstvu v Perun AAI. Tento agent využívá REST API, které Apereo CAS nabízí od verze 7.1.0, a umožňuje registraci, aktualizaci i deregistraci služeb jak na bázi SAML, tak OpenID Connect.

Všechna navržená rozšíření byla implementována jako modulární a konfigurovatelné celky. Jejich funkčnost byla ověřena nasazením do testovacího prostředí, které bylo realizováno na dvou serverech provozovaných v rámci e-INFRA CZ Cloudu. Jeden server hostoval Federation Registry a testovací službu, druhý server pak Apereo CAS a příslušného deployovacího agenta. Propojení s ARGO Messaging Service bylo zajištěno pomocí vývojové instance poskytnuté GRNETem a napojení na Perun IDM probíhalo vůči vývojové instanci provozované CESNETem.

Testování bylo provedeno manuálně formou realistických scénářů, které simulovaly běžné procesy registrace, aktualizace a deregistrace služeb. Ověřena byla i funkčnost při výskytu chyb, stejně jako chování při nasazení služby do různých integračních prostředí a jejich vzájemném sloučení.

Výsledky testování prokázaly, že implementovaná rozšíření zajišťují spolehlivou registraci služeb do Perun IDM a Apereo CAS, aniž by narušovala původní logiku a registrační procesy. Nová funkcionalita výrazně zlepšuje interoperabilitu mezi systémy správy identit a přináší vyšší flexibilitu pro správu služeb v prostředí výzkumných a akademických federací.

Federated Services Management Application

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Mgr. Kamil Malinka, Ph.D. The supplementary information was provided by Mr. Mgr. Pavel Břoušek. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jan Pavlíček
May 12, 2025

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Mgr. Kamil Malinka, Ph.D., and my consultant, Mgr. Pavel Břoušek, for their expert guidance, valuable advice, and continuous support through the assignment. I am also thankful to my family and girlfriend, who supported me throughout my studies.

Contents

List of Figures

Chapter 1

Introduction

In today's digital landscape, identity management plays a crucial role in ensuring secure and seamless access to online services. Federated identity systems have become essential in research and academic environments, allowing users to authenticate with a single identity across multiple services and institutions. These systems improve user experience, reduce administrative overhead, and enhance security and privacy [32]. However, maintaining such federations requires robust tools for managing the registration and configuration of services within the identity infrastructure.

One such tool is the Federation Registry, a web-based application developed by GRNET to simplify the registration and lifecycle management of SAML and OIDC-based services [13]. While the Federation Registry already supports integration with several identity proxy platforms, including Keycloak, MITREid, and SimpleSAMLphp [14], additional enhancements are needed to address broader use cases and improve compatibility with identity management infrastructures.

This thesis extends the capabilities of the Federation Registry to support integration with Perun AAI, an identity and access management solution developed by CESNET and Masaryk University [8]. Specifically, the work enables automatic propagation of service metadata to Perun IDM, the core identity management component of Perun AAI [18]. In addition, the thesis introduces support for Apereo CAS, a modern and feature-rich identity provider solution, by developing a dedicated deployer agent for service registration. It also introduces support for merging service deployments from different integration environments (development, demo, production) into a single identity infrastructure instance.

To meet these goals, several key extensions were designed and implemented. The integration with Perun IDM for service synchronization, the support for merged deployment environments, and a new deployer agent enabling service registration with Apereo CAS through its REST API [4].

All extensions were implemented as configurable modules to ensure flexibility and compatibility with diverse deployment scenarios, and the functionality was validated through comprehensive manual testing using a development environment that closely mirrors real-world production infrastructure.

To improve the overall language quality of the thesis, selected sections were rephrased using the ChatGPT model by OpenAI, and grammar and spelling were checked using the Grammarly tool.

Chapter 2

Identity management Basics

This chapter provides essential information about identity management, including key concepts, commonly used protocols, and an overview of the Perun AAI system. This foundation is necessary to understand the integration of the Federated Services Management application into the authentication and authorization infrastructure built on top of Perun AAI, which is the focus of this work.

Like in the real world, in the digital world, every person needs their identity, which uniquely distinguishes them from others. These digital identities allow users to authenticate themselves and gain access to applications and resources to which they are authorized. Here is where Identity Management Systems come into play. Identity management is the method by which these identities are organized, verified, and managed in digital systems. It is about ensuring that the right individuals or entities have access to the appropriate resources in a network or system [12].

2.1 Identity and Identity management

Identity refers to the unique set of attributes and characteristics associated with an individual, organization, or device that can be used to distinguish it from others in a system. Identity is central to both personal interactions and digital transactions, as it defines the roles, permissions, and capabilities of the entity in various contexts [35].

Identity management systems play a crucial role in the digital world. They are responsible for establishing, maintaining, and validating user identities. Therefore, they control access to resources, data, and applications. These systems ensure that the right people have the right level of access based on verified identities, and they help to increase the efficiency of the process of managing these identities across various platforms and environments [31].

Identity management systems perform several key functions. The main ones are authentication, authorization, and user management. Authentication is a function that verifies the identity of a user. It is usually achieved by verifying credentials, such as usernames and passwords, biometric scans, or other verification methods. Authorization is a function that determines whether a user has the right to access certain resources or perform specific actions within a system. User management means handling tasks related to the user's identity lifecycle, such as creating new user accounts, managing user roles, and handling permissions. The identity management system can also monitor and log user activities for compliance, security, and operational purposes [31].

In today's interconnected digital world, identity management systems are vital to protect sensitive information and systems from unauthorized access, efficiently manage user access, and ensure compliance with various regulatory requirements. They have become an integral part of any organization's security infrastructure [27].

2.2 Single sign-on

Websites and applications view user identities as user accounts that they manage. Big organizations usually provide multiple services for their users and employees, and they must have an account for each service. This is problematic for users because they have to remember all the credentials, and also for the organization when they are performing account synchronization [27].

Single sign-on (SSO) is a solution to this problem. It allows users to log in once with a single set of credentials, like a username and password, and then access all the connected services without having to log in again. The SSO system keeps track of who the user is, when they logged in, and how they were authenticated, and shares this information with the different services. This information is usually called user attributes and can allow complex authorization [32].

SSO improves user experience and productivity with fewer login processes. It can also increase security by integrating more robust security policies and multifactor authentication [32].

2.3 Federated identity management

Federated identity management is a framework that enables organizations to connect together into a federation and allows the use of one identity to obtain access to the services or networks of all organizations in the federation. Individual organizations are responsible for the authentication of their users and granting them access [1].
The federated identity management is based on four logical components [22]:

- The user is a person who uses his digital identity and interacts with the network application.
- The user agent is a software application or browser that can run on different platforms, such as a computer or mobile phone. The user agent can mediate the identity information flow or only allow it. The user's online interaction always takes place through an agent.
- The service provider (SP) site is an application that will request user authentication from a third party. The third-party providing authentication might also send user attributes back to the SP. SP is often called a relying party (RP) because it relies on third-party information.
- The Identity Provider (IdP) site is an application that provides user authentication and often holds user attributes to share with various SPs.

Figure 2.1 shows the connection between the components for a better understanding.

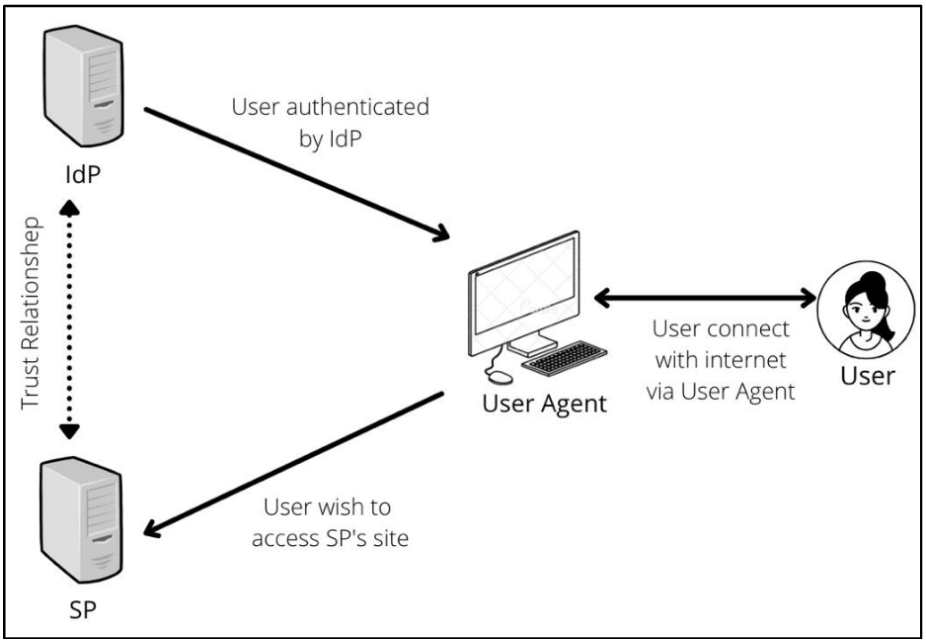

Figure 2.1: Components of Federated identity management system. Taken from [1].

In federated identity management, the user's credentials are always stored by the IdP, and the identification data and user attributes flow from the IdP to the SP. When accessing the service, the user does not have to provide his credentials to SPs. SP redirects the user to the IdP and trusts the IdP to verify the user's credentials. After authentication, the IdP can decide if the user can access the service. If the user is authorized, IdP provides the user's identity and attributes to the SP. SP can also perform its authorization processes and decide if the user will be allowed to access the service [22].

Both users and stakeholders benefit from federated identity management. It offers single sign-on for users between multiple organizations within the federation without sharing credentials. It delegates the cost of managing credentials and user attributes from SPs to IdPs, which can focus on authentication methods and the addition of attractive features. It also provides better scalability [9].

However, there are also disadvantages and challenges. Federation is based on mutual trust. Each participating member must follow the federation's policies and security protocols. There is no guarantee that user identity information will not be misused, and the system can also be very complex for users. There are also multiple security risks. The most serious concern is identity theft, which affects all partners in the federation. Identity theft can be performed by stealing the user's security token after successful authentication. The stolen token can be used to access resources within the entire federation [1].

2.4 Protocols

The communication between identity providers and service providers is established using standard authentication and authorization protocols. The commonly used protocols are SAML, OAuth 2.0, and OpenID Connect [34]. These protocols define how identity and authentication data are exchanged securely between parties. Each is designed for different use cases. SAML is used for establishing education and enterprise federations, OAuth 2.0

is useful to protect APIs, and OpenID Connect is used for the integration of new web or mobile applications [28].

It is necessary to understand the protocols as they are used for the service provider integration into the identity infrastructure.

Security Assertion Markup Language

Security Assertion Markup Language (SAML) is a standard that defines rules and syntax for data exchange. It provides a secure XML-based solution to exchange user security information between an IdP and an SP. It is flexible and, when needed, allows custom data transmission to the external SP. The main goal behind SAML is to form an XML framework to allow authentication and authorization from a single sign-on perspective, supporting W3C XML encryption for privacy requirements [19].

The SAML transaction consists of three roles. An asserting party, a relying party, and a subject. The asserting party is the IdP, which provides information about the user. The relying party is the SP, which trusts the user information provided by the IdP and provides the service application to the user. The user with his identity involved in the transaction is the subject of the transaction [19].

The SAML standard comprises four components: assertions, protocols, bindings, and profiles. All components can be customized for specific business case needs. The assertion is a transaction from the asserting party (IdP) to the relying party (SP). The relaying party counts on the validity of the assertion data. The XML schema defines the assertion structure and consists of header information, subject, and information about the subject in the form of attributes and conditions. Protocols define the form of requests and responses to communicate the assertion between SP and IdP. Bindings define the communication protocol used in the network. For example, the HTTP redirect binding uses an HTTP redirect message, and the HTTP POST binding uses base64-encoded content to transport assertions. Profiles is the highest SAML component and defines how the assertions protocol and bindings will work together to provide single sign-on [19].

For entities to work in a federation, the partner entities must share their flexible configuration. This is achieved by exchanging SAML metadata, which can be uploaded to the federation software without additional configurations. This process reduces the possibility of errors and also saves time. The metadata file contains an EntityDescriptor to specify the type of entity and an EntityID to clearly identify the entity. It can also contain many different elements and attributes that can be optional or required.

More information about SAML configuration can be found in the official documentation [26].

OAuth 2.0

OAuth 2.0 is the standard protocol for authorization. It primarily solves the access delegation problem. When a user wants to share a resource with a third-party application, he would need to provide his credentials to the resource. This is called delegation by credential sharing. The problem with this solution is that the third-party application can use user credentials to do whatever it wants. OAuth 2.0 solves this problem by sharing a temporary time-bound token instead of user credentials, which can only be used for specific, well-defined purposes [30].

In OAuth 2.0 flow, there are four typical actors [30]:

- The resource owner owns the resources. For example, a user who owns his profile information.
- The resource server is a place that hosts protected resources and shares them with authorized parties.
- The client is the application that wants to access a resource in the name of the resource owner.
- The Authorization server is an entity that provides OAuth 2.0 access tokens to client applications. It also validates provided access tokens.

A grant type defines the method by which a client application obtains an authorization grant from the resource owner, enabling access to protected resources. The scope specifies the level of access requested by the client and represents a clearly defined set of permissions that determine which actions the client is allowed to perform on the resource [30].
The core specification of OAuth 2.0 defines six grant types [30]:

- Authorization code grant type is recommended for applications with web browser access. It can be extended with a Proof Key for Code Exchange to prevent CSRF and authorization code injection attacks.
- Implicit grant type is mostly used by JavaScript clients running in web browsers and is not recommended in favor of authorization code due to security issues.
- Resource owner password credentials grant type, providing credentials to the client application based on trust.
- With client credentials grant type, the client becomes the resource owner by providing his Client ID and client secret.
- Device authorization grant type designed for browserless devices such as smart TVs, smart speakers, printers, etc.
- Refresh grant type is a special type for issuing a refresh token with an access token. The refresh token can be used to obtain a new access token after the lifetime expiration of the previous one without the involvement of the resource owner. The grant type can be combined with the authorization code and the resource owner password credentials grant types.

The OAuth 2.0 specification is built around three types of clients. Web applications are considered to be confidential clients running on a web server. End users or resource owners can access these applications through a web browser. Applications based on the user agent are considered public clients. These applications download the code from a web server and run it on the user agent, for example, a JavaScript running in the browser. Public clients cannot protect their credentials because the end user can see anything in the JavaScript. Native applications are also considered public clients. Native applications are under the control of the end user, and any confidential data stored in those applications can be extracted. Examples of native applications are native Android and iOS applications [30].

OpenID Connect

OpenID Connect (OIDC) provides a lightweight framework for identity interactions. It was developed under the OpenID Foundation based on OpenID and is built on top of OAuth 2.0. OpenID Connect is the most popular identity management protocol, and most of the applications developed in recent years support it [30].

OIDC is an extension of OAuth 2.0, adding an identity layer to the existing OAuth 2.0 protocol. This identity layer is represented by an ID token. OAuth 2.0 authorization server supporting OIDC provides an ID token along with the access token. While OAuth 2.0 is focused on access delegation and authorization, OIDC is focused on user authentication. Securing an API with OIDC is more related to the client or application level than to the API or resource server level. OIDC enables the client application to ascertain the user's identity, but the API or resource owner expects only the access token, and in this case, the user's identity is meaningless [30].

The ID token is the primary add-on. It is typically a JSON Web Token and provides the transport of the information about an authenticated user from the authorization server to the client application [30]. JSON Web Token is a compact and self-contained way to securely transmit information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret with the HMAC algorithm or a public/private key pair using RSA or ECDSA.

OIDC, independently of OAuth 2.0 grant types, defines a set of flows to authorization code flow, implicit flow, and hybrid flow. The flows differ by the value of the parameter `response_type`, which is always included in the request to the authorization endpoint. The `response_type` also defines the expected type of response from the authorization endpoint [30].

The OpenID Connect Identity provider can share its configuration to make it accessible to relaying parties on the metadata endpoint. The route to the metadata endpoint must be `/.well-known/openid-configuration`. In most cases, the endpoint is not secure, but it is accessible to anyone. The provider configuration can be retrieved by HTTP GET request and is usually in JSON format.

2.5 Service provider registration

Service provider registration is a key process in federated identity management systems as it facilitates the integration of SPs into the identity infrastructure. The registration process begins with the SP applying to join the infrastructure, providing essential details such as the nature of the service offered, domain specifics, and technical contact information. Adherence to infrastructure standards is a critical requirement for SPs.

An essential aspect of the registration process is the submission of the SP's metadata based on the selected protocol, usually SAML and OIDC. These metadata are a collection of vital SP information, including a unique identifier, service endpoints, and security credentials, such as public keys. This information plays a key role in enabling secure communication and interaction within the infrastructure [25].

After the metadata submission, the infrastructure or its constituent IdPs work on integrating the SP into the system. This phase is where the authentication protocols and user information exchange mechanisms are set up and tested. Successful integration implies that the SP can authenticate users via connected IdPs, using a streamlined process that improves user experience and security.

Maintenance and updates are an ongoing part of being a registered SP in the infrastructure. SPs must regularly update their metadata, renew security certificates, and comply with the evolving infrastructure policies. This ongoing process ensures the infrastructure remains secure and efficient in managing identities across various platforms.

The service provider registration can be provided by another service provider that is already part of the infrastructure. This service provider is usually a web-based application that provides a user-friendly interface not only for users who want to register and manage their SPs, but also for infrastructure administrators who are handling these registrations. Behind the scenes, there can be automated processes to build SP metadata and propagate it to configured identity providers. These applications are very important because SP owners do not have to be experts in the area of federated identity management to register their services, and they also simplify the work for infrastructure administrators [21, 20].

2.6 Perun AAI

Perun Authentication and Authorization Infrastructure (AAI) is the system developed collaboratively by CESNET and Masaryk University. It is based on the Authentication and Authorisation for Research and Collaboration (AARC) blueprint architecture[1], and it is used in multiple national and international research e-infrastructures [8]. Perun AAI is built from two main components: Proxy IdP and Perun IDM [6].

Proxy IdP is built on top of MITREid Connect, providing support for the OIDC protocol, and SimpleSAMLphp, providing support for the SAML protocol. Proxy IdP can act both as a proxy and an IdP. It provides the authentication layer with SSO for the connected services, and with the usage of the discovery service, it integrates IdPs from the federation or separately connected IdPs. Proxy IdP is using Perun IDM as an additional source of authentication and authorization data [7].

Perun IDM is an application designed for the management of virtual organizations (projects), users, groups, facilities, and resources. A virtual organization is a well-known concept from the computational grids, and it is a core unit for user management in Perun. This concept can be effectively applied beyond the grid environment, as it establishes membership restrictions and delegates responsibilities among organization members. Groups and resources exist within the virtual organization. Users can become members of the virtual organization and can be organized into groups. These groups of users can be allowed to utilize the resources. Resources are assigned to the facilities and constitute a binding between facilities and groups, and facilities are the Perun representation of the services. This connection between the facility, resources, and groups of users can represent the authorization to the service and can be released in user attributes. These attributes can also be used to map specific user roles within the service [23].

Perun IDM offers a wide range of advanced functionalities like user registration, user identity consolidation, management of user lifecycles, data propagation to another system, and many more. Additionally, it is highly flexible and can be tailored to meet specific use cases [23].

[1] AARC blueprint architecture: https://aarc-community.org/architecture/

Chapter 3

Federation Registry

This chapter presents an analysis of the Federation Registry, covering its architecture, provided use cases, service registration, and deployment flows. This analysis provides the necessary background for designing extensions to the Federation Registry to enable integration with Perun AAI. Most of the information in this chapter was acquired by studying documentation[1] and code[2] of the Federation Registry, code[3] of Deployer agents and by studying documentation[4] of the ARGO messaging service (AMS).

As mentioned in the previous chapter, the registration of service providers is a critical process in the area of federation identity management, which can be simplified by specialized web-based applications. One of these applications is the Federation Registry developed by GRNET [13].

GRNET is one of the largest public-sector technology companies operating in Greece. Since August 2019, it has operated under the auspices of the Ministry of Digital Governance. GRNET provides networking, cloud computing, data management services, e-infrastructures, and services to academic and research institutions, educational bodies at all levels, and public sector agencies. GRNET is also part of the EGI federation, like CESNET and many other organizations. EGI is a federation of computing and storage resource providers united by a mission to support research and development [16].

Federation Registry provides a secure web interface. This interface can be used by Service Owners to register and manage their OpenID Connect and SAML-based Services. Service owners can view and manage their services by creating and submitting service requests. Once a request is submitted, users with reviewing privileges are notified by email and can view and review it. When the service request is approved, the service deployment process is executed, and the service is deployed to the requested integration environment [13].

3.1 Architecture

The architecture is based on three together interacting applications. The first is the main web-based application named Federation Registry, which interacts with the service administrators who want to register their services with the identity infrastructure. The second one is the ARGO messaging service, which serves as a layer for exchanging messages with

[1] Federation Registry docs: https://federation.rciam.grnet.gr
[2] Federation Registry code: https://github.com/rciam/rciam-federation-registry
[3] Deployer agents code: https://github.com/rciam/rciam-federation-registry-agent
[4] AMS docs https://argoeu.github.io/argo-messaging/docs

information about the services. The last one is a set of deployer agents that read the messages from the ARGO messaging service, parse them into the specific format required by the proxy/IdP software used by the identity infrastructure, and register the service metadata with the identity infrastructure. The architecture is shown in the figure 3.1.

The Federation Registry supports multiple integration environments, so the services can be tested before integration into production. The number of integration environments is configurable, and the service validation process can also be different for each integration environment. There are three supported validation processes for the service registration request.

- Service can be registered without any validation. This can be used for development.
- With one-step approval, an authorized user must validate the service registration request
- With two-step approval, an operator with technical knowledge can approve the technical details of the service, and then the administrator validates general information about the service. This is usually used for production.

There must be an individual ARGO messaging topic for each integration environment.

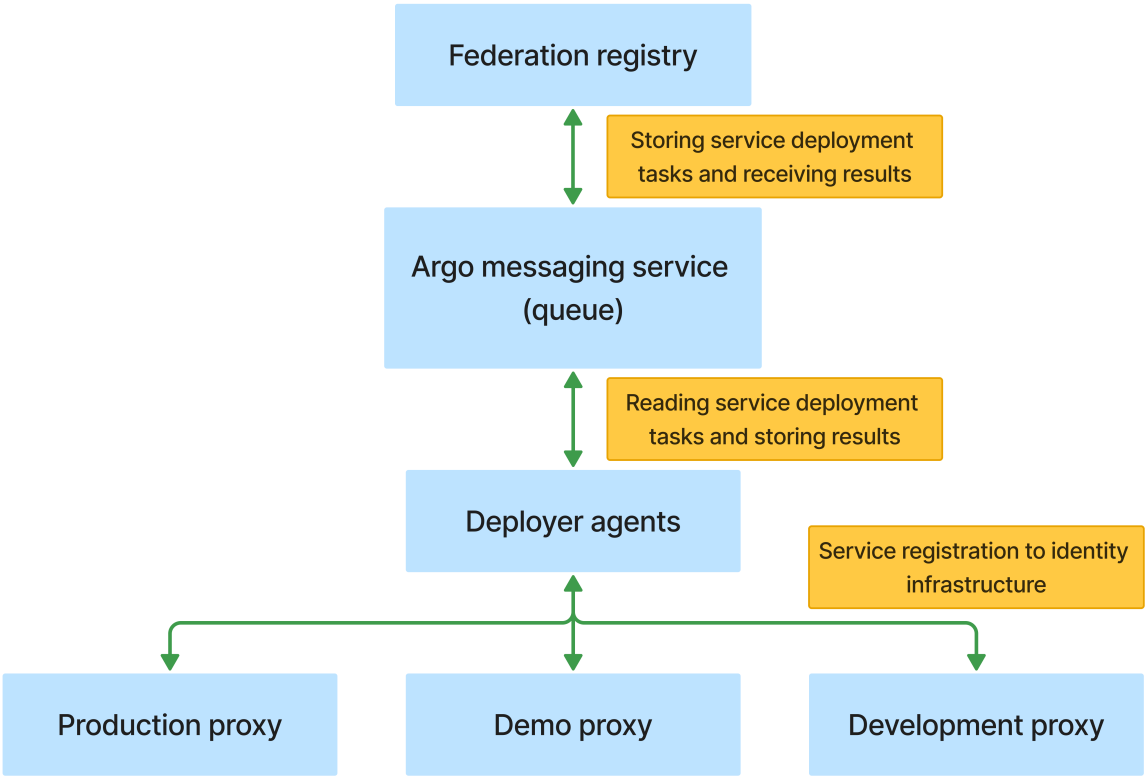

Figure 3.1: Federation Registry architecture schema

3.2 Main application

The Federation Registry is built from three components: the frontend, the backend, and the ARGO messaging service agent.

The frontend is an application based on the React framework, and it is built and directly served by the Nginx web server. The Nginx web server must have correctly configured HTTPS for proper functioning. Without the correct HTTPS configuration, the login functionality won't work. The backend provides a RESTful API and is created using the Express.js application framework. ARGO messaging service agent is a JavaScript application that initializes topics and subscriptions in the ARGO messaging service, reads the deployment tasks using specialized backend endpoints, and pushes them to the desired topics.

The PM2 process manager runs the backend and the ARGO messaging service agent. PM2 is widely used because it provides a simple and efficient way to manage Node.js application processes, including load balancing, logging, monitoring, and clustering features.

User authentication is provided by the OIDC identity provider, and the role of the authenticated user is evaluated by the OIDC eduPersonEntitlement claim. The eduPersonEntitlement is a URL representation of the set of rights to a specific resource [11].

The PostgreSQL database provides the backend data storage. The SQL script for creating the correct database schema can be found in the GitHub repository[5]. The database has to be filled with roles and their assigned actions, the deployer agent's specifications for creating relevant topics in the ARGO messaging service, and the OIDC registration information of the Federation Registry, since it is registered as a service provider in the identity infrastructure.

3.3 ARGO messaging service

The ARGO Messaging Service facilitates real-time communication between separate applications by allowing them to send and receive messages. This service operates on a Publish/Subscribe model. In this framework, publishers are individuals or systems capable of sending messages to designated channels, known as a topic, and the subscribers are individuals or systems that set up Subscriptions to these specific topics in order to receive messages. The benefits of this architecture are durability, scalability, and availability [15].

3.4 Deployer agents

Deployer agents are Python scripts designed to run as system services. The deployer agent periodically calls the ARGO Messaging Service API and pulls new messages from the specific topic. The idea is that there should be one deployer agent for each AMS topic. After pulling the new message from the topic, the message is parsed into the format desired by the software used by the proxy/IdP. When the message is in the desired format, it is propagated to the proxy/IdP, and the service can start using the proxy/IdP for authentication. After the propagation, the agent parses the result and pushes it to the AMS topic intended for the deployment results. AMS then calls the Federation Registry backend API endpoint to store the service deployment result.

There are three implementations of deployer agents. Each of them is intended to support deploying the service data to the following software that can power the proxy/IdP in the identity infrastructure.

[5]Database schema: https://github.com/rciam/rciam-federation-registry/blob/master/db_schema.sql

- SimpleSAMLphp is an application written in native PHP that mainly focuses on supporting identity provider and service provider over SAML2 protocol [29].
- MITREid Connect implements a relying party and an OIDC Provider over the OIDC and OAuth 2.0 protocols. It is built using Java, Spring framework, and Spring Security framework [33].
- Keycloak is an identity and access management software. This means that it implements the identity provider over SAML2 and the OIDC Provider over OIDC and OAuth 2.0 protocols. Keycloak is written in Java programming language [17].

3.5 Components communication schema

The schema in the figure 3.2 shows the communication between all of the components.

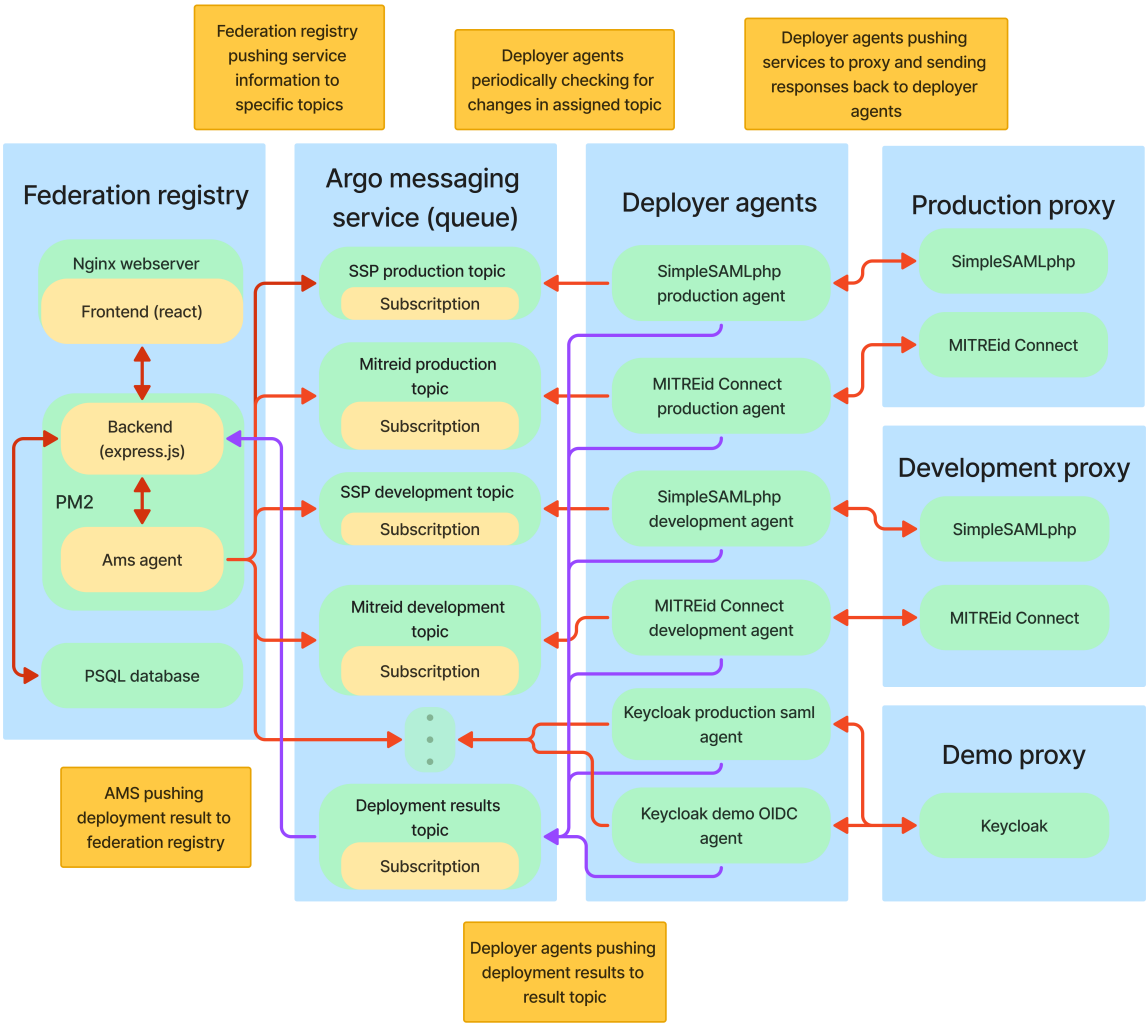

Figure 3.2: Federation Registry components schema

The Federation Registry frontend calls backend endpoints to interact with the server side. The Backend stores and reads the data from the PostgreSQL database. The ARGO

messaging service agent interacts with specialized backend endpoints to read and update deployment tasks, and then pushes the service information to the appropriate ARGO messaging service topics. ARGO messaging service topics are just storing the data. Deployer agents read the data from the topics, parse it, and register services with the proxy/IdP. After service registration, deployer agents push the registration status to the special topic to store the result of the deployment. This special topic is configured to call the federation registry backend endpoint, which is designed to gather and parse the deployment result data.

3.6 Provided use cases

Figure 3.3 shows actors with actions they can perform in the federation registry.

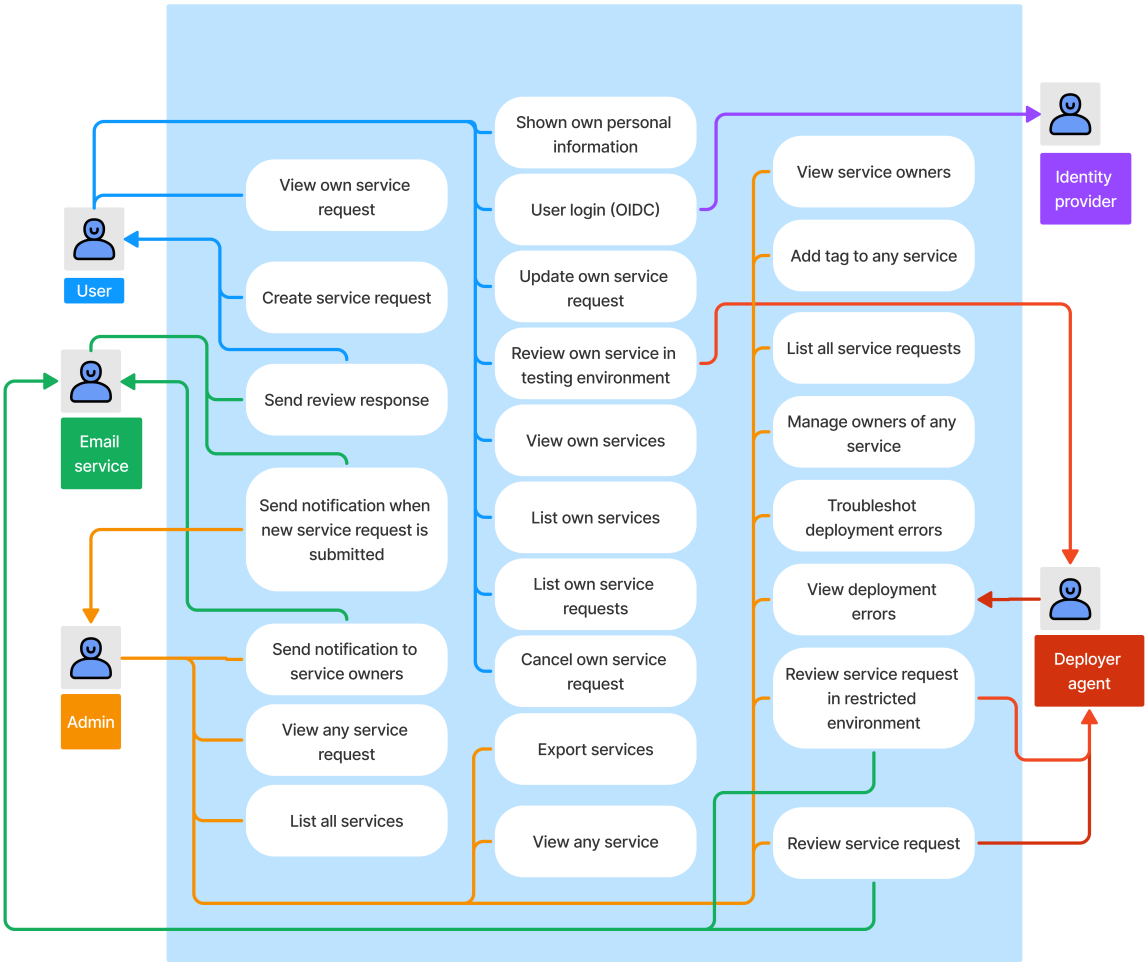

Figure 3.3: Federation Registry use cases schema

The Federation Registry has four main actors. These actors are the identity provider, users, email service, and deployer agents. The identity provider provides the user login through the OIDC protocol. The users can have different roles. Roles and assigned actions are fully configurable, but from the application design, there should be at least two roles. A role for regular users and the administrator's role. Regular users can view, create,

update, and delete their own service requests and registered services. The administrator role is enriched by managing all services and their owners, reviewing the service requests, the ability to notify the service owners, exporting services, tagging services, and managing the service deployment. The email service is responsible for sending email notifications regarding service requests. The deployer agent is responsible for deploying the service when the registration request is accepted and also for propagating the result of the deployment.

3.7 Service registration flow

Figure 3.4 shows the service registration process.

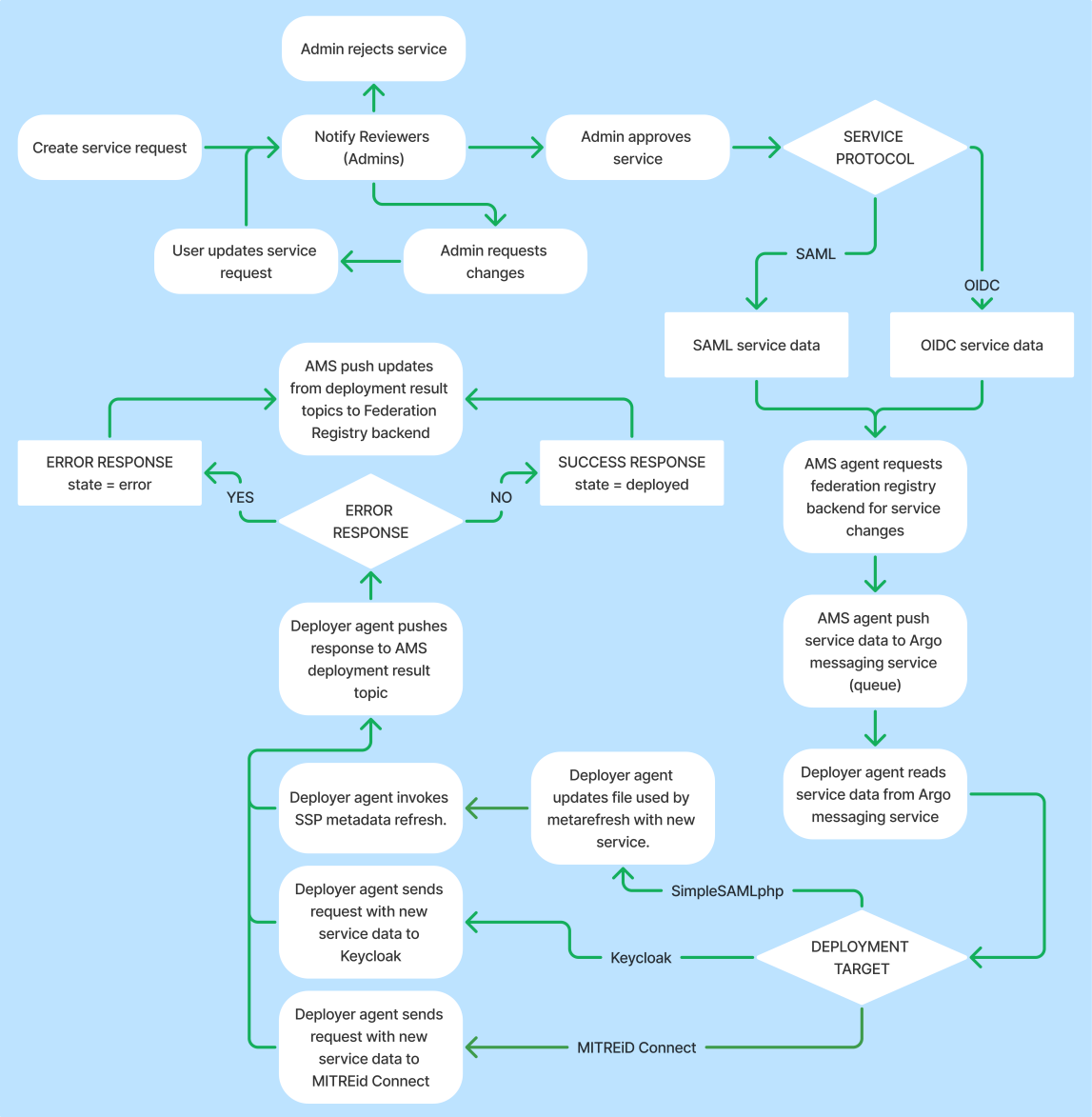

Figure 3.4: Service registration flow schema

It starts with the user creating the service registration request. After the registration request submission, the administrators who will be validating the request will receive an email notification. The administrator can approve or reject the registration request or return the request back to the user with a description of the properties the user must change. When the administrator validates the registration request, the user gets an email notification about the request status. When the registration request is approved, the deployment task will be created, and the AMS agent will propagate service data to the ARGO messaging service. The deployer agent will then read and parse the service data from the ARGO messaging service and register the service with the proxy/IdP. At the end of the flow, the deployer agent, using the ARGO messaging service, will propagate the registration result back to the Federation Registry.

Chapter 4

Design of Federation Registry extensions

This chapter describes the extensions of the Federation Registry required for integration with the Perun AAI solution.

The first extension enables service registration to both the proxy/IdP and the Perun IDM system, including automatic synchronization of registered services and management of their corresponding representations in Perun IDM. This requires authenticated communication with the Perun API, correct mapping of service properties to Perun entities, and reliable handling of provisioning errors. This extension will enable centralized identity and access management for services integrated within the Perun AAI.

The second extension allows services from different integration environments (such as development, testing, and production) to be registered within a single proxy/IdP and Perun IDM instance and adds the ability to move services within integration environments. This consolidation reduces operational complexity, but requires unique service identifiers across integration environments, and support for merged-environment configuration in the Federation Registry frontend, backend, and the AMS agent. This extension is necessary because the Perun AAI solution operates all integration environments within a single instance.

The final extension adds support for registering services to a proxy/IdP powered by Apereo CAS. This functionality uses the CAS REST API to perform service registration, updates, and deregistration for both SAML and OpenID Connect services. It requires authenticated access to the CAS REST API, compatibility with CAS-specific service representations, and integrates the functionality provided by the first extension. This extension ensures that the Federation Registry remains compatible with the Perun AAI system following its Perun Proxy IdP migration from MitreID and SimpleSAMLphp to Apereo CAS.

All of the proposed designs are configurable, giving Federation Registry administrators the ability to enable or disable them according to their specific use cases.

4.1 Design of integration with Perun IDM

The service registration to the proxy/IdP using the Federation Registry was described in the previous chapter. The same process is used when registering the service with Perun Proxy IdP. However, Perun Proxy IdP is getting additional attributes and authorization

data from Perun IDM, and for this reason, there must also be a service representation in Perun IDM.

The main goal of the integration is to create a facility (Perun IDM service representation) with the right attributes, create a group intended for users who should have the right to manage the facility, and add the user who requested the service registration as a member of this group.

Possible approaches

The most suitable stage for interaction with Perun IDM in the service registration flow from the Federation Registry to the proxy/IdP is within the deployer agents.
There are two main design questions:

1. Should the interaction with Perun IDM occur **before** or **after** service registration to the proxy/IdP?

2. What should happen if a task within the Perun IDM interaction fails?

The first question can be answered by examining the service registration process for MITREid and Keycloak. In both cases, the OIDC `client_id` is automatically generated during service registration to the proxy/IdP when not manually specified by the user. After registration, the `client_id` is included in the registration response and returned to the Federation Registry, where it is assigned to the service. Based on this, the correct approach is to interact with Perun IDM **after** successful service registration to the proxy/IdP.
The second question has two possible approaches:

1. Roll back the service registration changes in the proxy/IdP, and upon retrying deployment, repeat all registration tasks.

2. Store the fact that the service registration to the proxy/IdP was successful and skip it when retrying deployment.

The first approach is problematic due to its complexity. When updating or deleting the registered service, there would have to be a mechanism to persist previous states to allow the rollback functionality. Furthermore, if the rollback mechanism fails, subsequent deployment retries could lead to inconsistent or unexpected behavior. Rollback implementation would also vary across proxy/IdP software.

The second approach is simpler. An additional property in the deployment response sent back to the Federation Registry could signal the result of the service registration to the proxy/IdP. This signal would be included in any retry deployment task, allowing the deployer agent to decide whether to skip the service registration to the proxy/IdP and perform only interaction with Perun IDM. The only drawback is the need to update the Federation Registry's database schema to store this information.

Considering the advantages and drawbacks of both approaches, **the second approach will be used**.

Updated service registration flow

Figure 4.1 shows the service registration flow introduced in the section 3.7 extended with the Perun IDM integration. The updated flow introduces a new decision point before service registration to the proxy/IdP, as well as an additional branch of tasks executed by the

deployer agent after successful registration. The new decision state handles the scenario of retrying service deployment after failure with the ability to skip service registration to the proxy/IdP. The added task branch includes storing a property in the deployment response to indicate successful registration to the proxy/IdP, and interacting with Perun IDM to create a corresponding facility (service representation in Perun) with the appropriate attributes. It also involves creating a managers group associated with this facility.

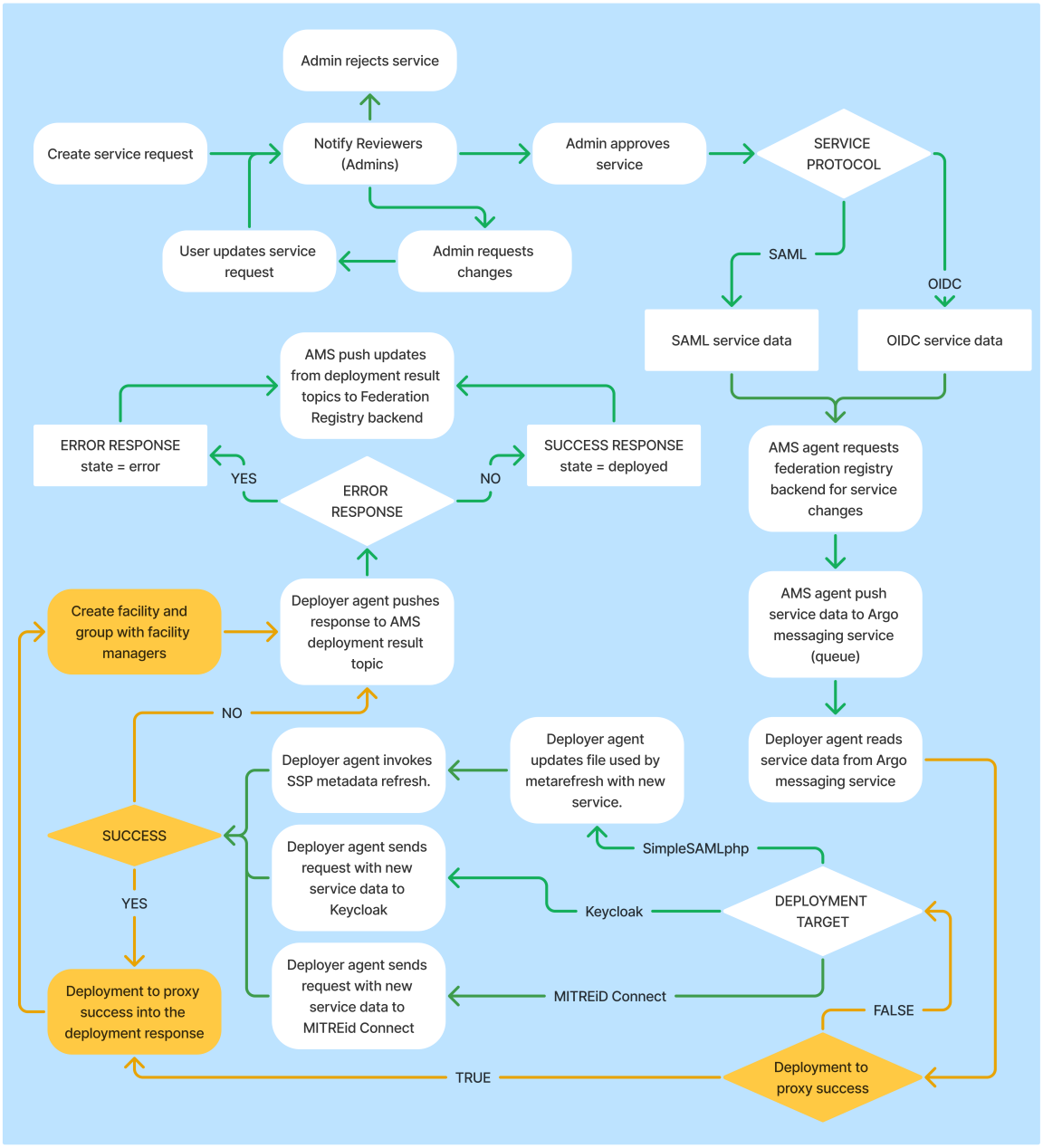

Figure 4.1: Service registration flow schema with Perun IDM integration

Communication with Perun IDM

The standard way of communication with Perun IDM is through the Perun RPC API. Perun RPC is not a traditional REST API, so it is required to be used according to the documentation [24].

Authentication against Perun RPC can be done with the identity provided by Kerberos, Shibboleth IdP, Certificate, Remote user (basic authentication), or OIDC using device code flow. Since the authentication against Perun RPC will be performed by the deployer agent, which is an application and not a real person, the best approach is to create a Perun service account with the Perun admin role intended for integration with the Federation Registry and use the Remote user authentication type. The other types would be much harder to implement without security benefits, as the credentials would still be stored on the machine running the Federation Registry software. The authorization is done by checking the rights of the authenticated user who has an existing account in Perun IDM. Regarding the request type, it is recommended to use POST requests all the time with JSON format. The JSON format will also be used for the responses [24].

4.2 All deployment environments to single proxy/IdP registration

The Federation Registry architecture allows service registration across multiple integration environments, such as production, development, and demo. The number and names of these integration environments are configurable. Each integration environment requires a dedicated ARGO messaging service topic and a corresponding proxy/IdP. This architecture generally makes sense, but Perun AAI can manage all integration environments with a single instance. This greatly reduces operational costs and complexity because all the information is in one place. This is achieved by Perun IDM storing the integration environment attribute on the facility and providing access control features. From this ability of Perun AAI came up the request to create a design for service registration from all deployment environments to a single proxy/IdP.

Service deployment within Federation Registry and AMS agent

The database schema 4.2 contains tables and bindings within the Federation Registry relevant to service deployment.

When a service registration request is approved, the Federation Registry stores service information in `service_details` table (there are many more tables related to this, but they are not shown for simplicity). Then it stores the create state in `service_state` table and sets the deployment task to `deployment_tasks` table. The `tenant_deployer_agents` represents the deployer agents used for the service registration to the proxy/IdP. The AMS agent is using them to create relevant ARGO messaging service topics and subscriptions and to send deployment tasks to these topics. The `service_errors` table stores the service registration errors.

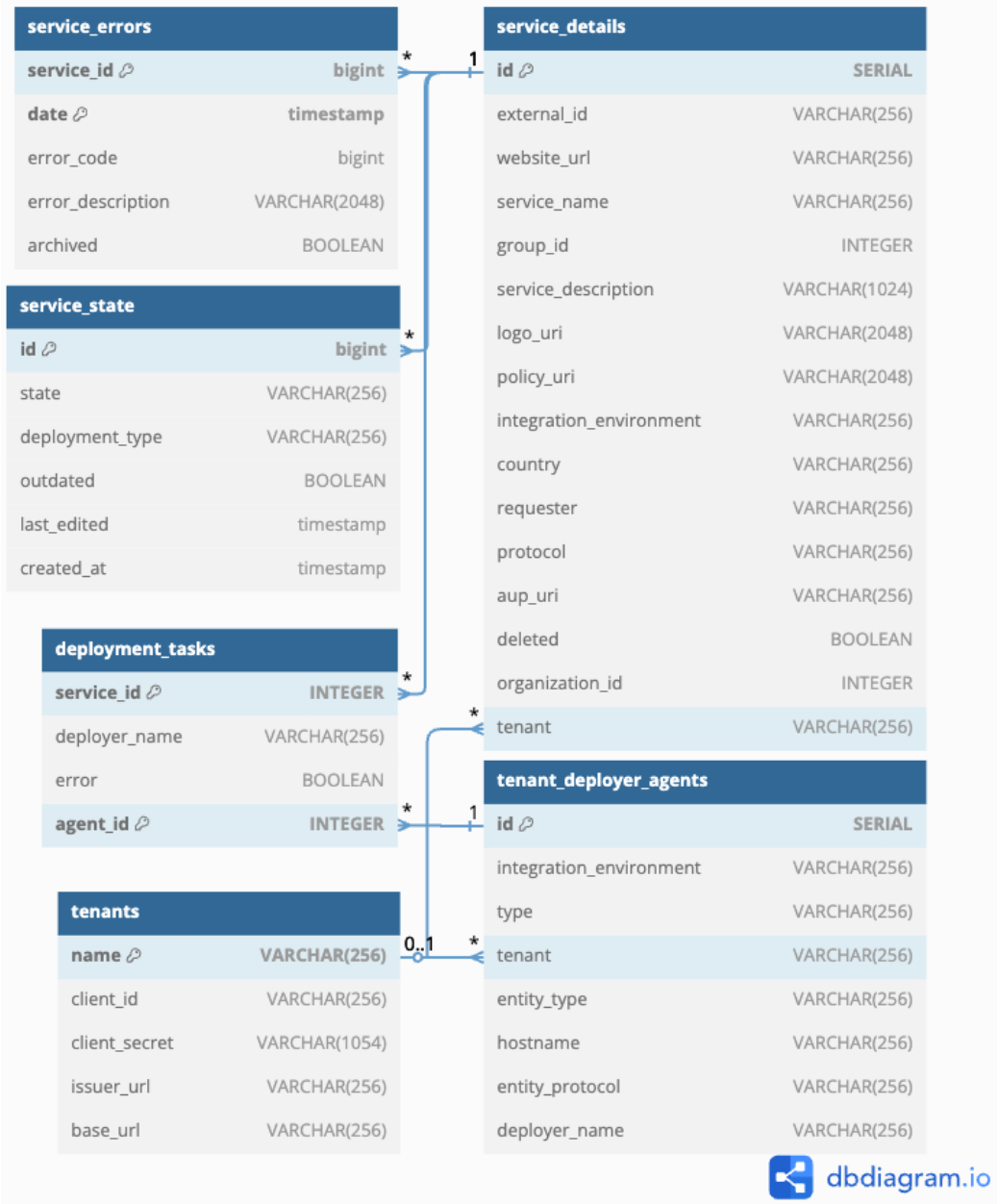

Figure 4.2: Part of database schema relevant to service deployment

Required updates

Merging multiple service integration environments into one deployment requires multiple updates. The first is to create a configuration option to turn it on and a configuration option to identify the tenant deployer agent used for merged deployment. The next one is to extend the OIDC Client ID and SAML Entity ID uniqueness checking between integration environments. Since the registration to proxy/IdP would result in an error. The third is to change the creation of deployment tasks to associate them with tenant deployer agents for merged deployment and update the AMS agent to push service data to the merged deployment topic. The fourth is to clear the Client ID and Entity ID when copying the

service to different integration environments when merged deployment is turned on. The last one is to add functionality to move the service to another integration environment when merged deployment is turned on. The process will behave the same as the service registration update since services from all deployment environments are registered with a single proxy/IdP, and the integration environment is represented as a facility attribute in Perun IDM.

4.3 Propagation to Apereo CAS

The Apereo Central Authentication Service (CAS) is a multilingual enterprise-grade single sign-on solution and identity provider for the web, designed to serve as a comprehensive platform for authentication and authorization. CAS is an open, well-documented authentication protocol. Its primary implementation is an open-source Java server component of the same name, offering support for a wide range of authentication protocols and additional features [3].

As mentioned earlier, the Federation Registry supports integration with Keycloak, MITREid Connect, and SimpleSAMLphp proxy/IdP software. This design and the following implementation will add the possibility of integrating the Federation Registry with Apereo CAS. This is particularly relevant for Perun AAI, as the current objective is to replace the existing Perun Proxy IdP software stack, which combines SimpleSAMLphp and MITREid, with Apereo CAS. The goal is to design and implement a new type of deployer agent that will be able to register, update, and deregister both SAML and OIDC services, as both protocols are supported by Apereo CAS [3]. The new deployer agent also has to be compatible with Perun IDM integration.

Possible approaches

There are two possible ways to manage registered services in Apereo CAS.

1. By updating service information directly in the storage used by Apereo CAS.

2. By using Apereo CAS API for service management.

Apereo CAS supports multiple options for storing registered services, such as in-memory, JSON file, MongoDB, or relational database [5]. The problem with this approach is that for full support, there would have to be an implementation for each of the storage options, or at least for the most commonly used ones. Fortunately, Apereo CAS version 7.1.0 introduced new API methods for registering and updating services included in the Palantir Admin Console update, enhancing service management capabilities [2]. This allows implementations to manage registered services through a standardized API, without worrying about the underlying storage method used in Apereo CAS. The only drawback of this approach is that the deployer agent would be compatible only with Apereo CAS versions higher than 7.1.0. However, this will not affect the Perun Proxy IdP software stack update, as it will use the latest software with regard to new features and the highest level of security.

Based on the evaluation of the available options, **the second approach**, using API calls for integration with Apereo CAS, proves to be the better choice and will be used in the deployer agent implementation.

Apereo CAS deployer agent details

The communication between the ARGO messaging service is provided by the implementation used by other deployer agents. This implementation relies on the argo-ams-library[1], which handles the interaction with the AMS, including authorization, message publication, subscription management, and error handling.

After retrieving the service management message from the ARGO messaging service, the Apereo CAS deployer agent parses the service properties and constructs a new JSON message in the format required by Apereo CAS. During this process, the deployer agent maps property names, transforms values as needed, and discards any properties that are not supported by Apereo CAS. The property mapping implementation will be according to the service management[2], SAML service[3], OAuth2.0 client[4] and OIDC client[5] Apereo CAS documentations.

For communication with Apereo CAS, the deployer agent can use the following CAS Actuator API endpoints:

- `/cas/actuator/registeredServices/{id} [GET, DELETE]`
- `/cas/actuator/registeredServices/type/{type} [GET]`
- `/cas/actuator/registeredServices [GET]`
- `/cas/actuator/registeredServices/export/{id} [GET]`
- `/cas/actuator/registeredServices/export [GET]`
- `/cas/actuator/registeredServices/import [POST]`
- `/cas/actuator/registeredServices [POST, PUT]`

For authentication, Basic Auth can be used with the Authorization header in the format: `Authorization: Basic base64(username:password)`. The API supports multiple content types for requests and responses, including the commonly used `application/json`. The response codes follow standard HTTP conventions [4].

After the deployer agent receives and validates the response from the Apereo CAS API, it will then perform the tasks related to integration with Perun IDM described in the section 4.1.

The last task of the deployer agent is to create and push the deployment result message to the ARGO messaging service topic used for propagation of the deployment result to the Federation Registry.

[1] argo-ams-library: https://pypi.org/project/argo-ams-library/
[2] Service management: https://apereo.github.io/cas/7.1.x/services/Service-Management.html
[3] SAML service: https://apereo.github.io/cas/7.1.x/services/SAML2-Service-Management.html
[4] OAuth2.0 client: https://apereo.github.io/cas/7.1.x/authentication/OAuth-Authentication-Clients.html
[5] OIDC client https://apereo.github.io/cas/7.1.x/authentication/OIDC-Authentication-Clients.html

Chapter 5

Implementation

This chapter details the implementation of the key extensions made to the Federation Registry to enable seamless integration with Perun AAI. These extensions support service management, including registration, updates, and deregistrations, across multiple systems, including Perun IDM and various proxy/IdP solutions. The implementation is based on the designs introduced in the previous chapter and focuses on three major areas:

- Integration with Perun IDM, which ensures that service metadata is consistently synchronized between the Federation Registry and Perun identity management system.
- Support for merged deployment environments, allowing services from different integration environments (e.g., development, production, demo) to be registered to a single proxy/IdP and Perun IDM instance.
- Development of a new deployer agent for Apereo CAS, expanding the Federation Registry's support to include service registration to the Apereo CAS proxy/IdP solution.

Each section in this chapter describes the corresponding implementation details, including changes to deployer agents and the Federation Registry's components. Together, these updates enhance the Federation Registry's flexibility across different identity infrastructure architectures.

5.1 Implementation of integration with Perun IDM

This section describes the implementation of the Federation Registry integration with Perun IDM. It is based upon the design introduced in the previous chapter, ensuring reliable communication between the systems and supporting service registration, updates, and deletions. The main part of integration is realized as an extension of the deployer agents, which coordinates deployments to both the proxy/IdP and Perun IDM. The additional modifications required for repeated deployment after Perun IDM deployment failure are implemented in the Federation Registry backend.

The deployer agents extension is implemented in Python and is structured into several key components. The components are Perun RPC Adapter and Perun Client API. The extended deployers are Deployer Keycloak, Deployer MitreID, and Deployer SSP. Part of the implementation also involves generic methods in a common utils.

The federation registry backend modifications are implemented in JavaScript, and update service deployment error handling.

All of the implementation details are described in the following subsections.

Perun RPC Adapter

The Perun RPC Adapter is implemented to handle communication with the Perun RPC API. This adapter abstracts the complexities of the Perun RPC interface and provides a simplified interface for higher-level operations. The adapter implements specialized methods and, through them, provides functionality for:

- **Facility management**: creating, updating, retrieving, and deleting facilities
- **Group management**: creating, deleting, and assigning administrative groups associated with facilities
- **User and membership management**: Resolving user identities based on external authentication sources (IdPs) and managing group memberships.
- **Attribute management**: Setting and retrieving Perun attributes associated with facilities (e.g., service identifiers, group IDs).

All of these methods are built on top of a core `call_perun_api` method that constructs HTTP POST requests and interacts with the Perun RPC API using the Python requests library[1]. Additionally, the `call_perun_api` method includes error handling logic to distinguish between known Perun-specific exceptions and unexpected or connection-related failures. This ensures that the integration can gracefully recover from predictable errors while surfacing critical issues.

The configuration must include the API URL and, if needed, the username and password for basic authentication with the Perun RPC API.

Perun Client API

Perun Client API is built on top of the RPC Adapter. It orchestrates the business logic for service registration, update, and deletion operations. It translates Federation Registry service metadata into Perun IDM objects and ensures consistency across systems. The component provides three core methods for managing service metadata in the Perun IDM. These core methods rely on several supporting methods that handle internal logic, such as mapping service properties to perun attributes. The core methods are `register_new_service_in_perun`, `update_service_in_perun`, and `delete_service_in_perun`.
The key operations done by `register_new_service_in_perun` method are:

- Creating a facility in Perun to represent the service.
- Creating a dedicated administrative group for managing the facility.
- Adding the requesting user as a member of the administrative group.
- Setting facility attributes based on service properties mappings configured within the deployer agent.

The key operations done by `update_service_in_perun` method are:

- Locating the facility via its unique service identifier attribute.

[1]Python requests library: https://pypi.org/project/requests/

- Synchronizing the facility's name, description, and attributes with updates from the Federation Registry.

The key operations done by `delete_service_in_perun` method are:

- Locating the facility via its unique service identifier attribute.
- Removing the facility's associated administrative group from Perun IDM.
- Removing the facility from Perun IDM.

Each of these methods receives a deployment message as a parameter and parses the relevant service information from it. The mapping of service properties to facility attributes is handled dynamically based on the `properties_mapping` configuration option. This allows administrators to define how service properties correspond to facility attributes, including the possibility to map a single property to multiple facility attributes. Additionally, value mapping is supported, enabling the translation of service property values into different facility attribute values as needed. The `static_attributes` configuration option is also available to define static values for specific facility attributes. These general mappings are essential, as attribute definitions may vary across different Perun IDM instances.

The example configuration with the configuration options description is provided in the `README.md` file.

Deployer agents changes

As mentioned earlier, there are three types of deployer agents: the SSP, Keycloak, and MitreID. All of them can newly include `perun` configuration option in their config with configuration for the Perun Client API and enable Perun IDM integration features. If enabled, deployer agents create an instance of the Perun Client API, and after successful deployment to the proxy/IdP use it together with the new common function `deploy_to_perun` to trigger deployment to Perun IDM.

The deployer agents are now signaling the result of deployment to the proxy/IdP in a dedicated property `proxy_deploy_success` in the deployment response. The Keycloak and MitreID deployers are now working with this signal, which is also included in the deployment messages, and based on this, are performing or skipping deployment to the proxy/IdP to prevent the emergence of inconsistencies. The SSP deployer does not need to work with this signal as it works with a metadata file, not the API, and the deployment is idempotent.

Since integration with Perun IDM increased the complexity of the `process_data` functions, which control deployments to both the proxy/IdP and Perun IDM and offer similar functionality for both the Keycloak and MitreID deployer agents, these functions have been refactored and the shared logic has been extracted into a common utility method called `process_data_generic`.

Federation Registry backend changes

The Federation Registry backend was extended to handle the signaling of successful service deployments to the proxy/IdP. Specifically, the `service_errors` database table was modified to include two new columns:

- `proxy_deploy_success` (`BOOLEAN`, default `FALSE`): Indicates whether the service deployment to the proxy/IdP succeeded.

- `solved` (`BOOLEAN`, default `FALSE`): Tracks whether the service deployment error has been resolved.

These changes required updates to the Service Error Repository, which now handles the population of the `proxy_deploy_success` column when inserting new service errors. Additionally, the repository now provides a method to mark all deployment errors related to a specific service as resolved. This step is crucial, as unresolved service deployment errors would prevent subsequent deployments to the proxy/IdP from being triggered in the deployer agent.

The `deploymentUpdate` function, which processes and records service deployment results, was updated to handle the new logic. It now stores the `proxy_deploy_success` flag as part of the deployment error and invokes the repository method to mark related service errors as resolved upon a successful deployment.

Finally, the deployment message sent to the ARGO Messaging Service (AMS) was extended to include two additional fields:

- `requester`: The identifier of the user who registered the service.
- `proxy_deploy_success`: The deployment success status for the proxy/IdP, derived from the current unresolved service deployment errors.

To support this, the `getPending.sql` query was enhanced to retrieve the `requester` and determine the `proxy_deploy_success` status specifically from unresolved deployment errors related to the service.

5.2 Implementation of all deployment environments to single proxy/IdP registration

This section describes the implementation of the ability to register services from all deployment environments to a single proxy/IdP instance (integration environments merge), together with the ability to move services between integration environments when this feature is turned on. The implementation is based on the design introduced in the section 4.2 in the previous chapter. The merging of multiple service integration environments is realized in the Federation Registry backend and ARGO messaging service agent. The service movement between integration environments is implemented only in the Federation Registry frontend, as it uses the service edit functionality on the backend. The following subsections describe the implemented updates across the Federation Registry components.

ARGO messaging service agent updates

The AMS agent is responsible for periodically obtaining pending service deployment tasks and publishing them to the appropriate AMS topics. One of the key criteria for selecting the target topic is the service's integration environment.

The new implementation introduces the ability to override this behavior when merged deployment is enabled. Instead of selecting the topic based on each service's integration environment (one of several selection criteria), the agent uses the environment name specified in the `merged_integration_environment_name` configuration property. This ensures that all deployment tasks, regardless of their original integration environment, are routed to a single AMS topic.

This behavior is controlled by the `merge_environments_on_deploy` configuration option, which activates the merged deployment mode for AMS topic selection when set to `true`.

Federation Registry backend updates

To prevent conflicts during registration to the proxy/IdP, the federation registry enforces the uniqueness of the OIDC Client IDs and SAML Entity IDs within integration environments.

The Federation Registry must force the uniqueness of OIDC Client IDs and SAML Entity IDs across all integration environments when merged deployment is enabled. This ensures that identifiers are globally unique, avoiding potential collisions that could disrupt service registrations.
The implementation achieves this through the following updates:

- Addition of two SQL queries at the database level:
  - `checkClientIdAllEnvironments.sql` – verifies the uniqueness of OpenID Connect Client IDs across all integration environments.
  - `checkEntityIdAllEnvironments.sql` – verifies the uniqueness of SAML Entity IDs across all integration environments.
- Integration of the new queries into the `ServiceDetailsProtocolRepository`, allowing the backend to perform these global checks as part of its standard operations.
- Update of the `isAvailable` function to utilize the new repository methods, ensuring that identifier availability checks respect the merged deployment configuration.
- Modification of the service validators to incorporate the new repository methods, ensuring that submitted service data is validated against global uniqueness constraints across all integration environments when merged deployment is enabled. This validation occurs at the API endpoint level, preventing conflicting identifiers from being accepted into the system.

The last update of the Federation Registry backend is related to the endpoint for updating the service state and storing a new deployment task in the database. Previously, the service's integration environment was one of the criteria for assigning a deployment task to the deployer agent representation in the Federation Registry. However, with merged integration environments enabled, the integration environment is no longer considered in this selection process. Instead, deployment tasks are assigned directly to the merged deployer agent.

Federation Registry frontend updates

The primary objective within the Federation Registry frontend is to introduce functionality that allows services to be moved between integration environments when the merged deployment environments feature is enabled. This feature is activated by setting the `merge_environments_on_deploy` configuration option to `true` in the frontend configuration.
The implementation achieves this functionality through the following updates:

- Addition of a new route in `Routes.js` that renders the page for editing service information during the service move process.

- Introduction of the new `MoveDialog.js` component, which presents the user with available integration environment options for moving the service. Once an environment is selected, the user is redirected to the move service page, allowing them to adjust the service information as needed to meet the requirements of the target integration environment.
- The `ServiceForm.js` component, which handles viewing, editing, and updating services. By default, when viewing a registered service, the form includes a button that opens the copy dialog. With merged deployment enabled, this button instead opens the move dialog.
- The `ServiceList.js` component, which presents the list view of the service registrations that the user can manage. Each service entry includes a dropdown menu with available actions. When merged deployment is enabled, this menu is extended with an additional action for moving the service between integration environments. Selecting this action opens the move dialog.
- Update of condition in the data preparation logic for the service copy form to always remove the OIDC Client ID and SAML Entity ID from the data when merged deployment. Keeping these identifiers would violate the enforced uniqueness across all integration environments and result in submission errors.

5.3 Implementation of Propagation to Apereo CAS

A new deployer agent was implemented to enable the Federation Registry to propagate service registrations to the Apereo CAS. This agent leverages the Apereo CAS REST API, available from version 7.1.0, to manage the registration, update, and deregistration of services for both SAML and OIDC protocols. The deployer agent architecture follows the design introduced in the previous chapter in the section 4.3, the existing design for other deployers (e.g., Keycloak, MITREid, SimpleSAMLphp), and integrates seamlessly with the ARGO Messaging Service (AMS) via the `argo-ams-library`.

Additionally, the deployer agent incorporates integration with Perun IDM as described in the section 5.1. After completing service registration to Apereo CAS, the agent updates service metadata within Perun IDM, ensuring service synchronization between the Federation Registry and Perun IDM.

The example configuration with the description is provided in the `README.md` file.

Deployer Agent Workflow

The core logic for the CAS deployer agent is encapsulated in the `deployer_cas` script. The workflow proceeds as follows:

1. **Receiving Messages from AMS**: The agent continuously polls AMS for service deployment messages using the existing `PullPublish` mechanism. Each message contains service metadata and deployment instructions (create, update, or delete).
2. **Message Transformation**: Upon receiving a message, the agent translates the service metadata into the JSON format required by Apereo CAS. This transformation handles both OIDC and SAML service types.

3. **Interaction with Apereo CAS API**: The agent communicates with the CAS Actuator API endpoints to manage registered services. Authentication is performed via Basic Auth, using credentials provided in the configuration.

4. **Handling API Responses**: After performing the API operation, the agent parses the CAS response and constructs a deployment result message. The response includes key identifiers such as `id` (external service ID) and `Client ID` (for OIDC services).

5. **Integration with Perun IDM**: The agent updates service metadata in Perun IDM according to the implementation described in the section 5.1, ensuring synchronization between the Federation Registry and the Perun IDM.

6. **Result Propagation to AMS**: Finally, the deployment result message is published back to AMS, signaling the outcome of the operation to the Federation Registry backend.

Supporting Components

The `deployer_cas` script relies on the following supporting components to perform its functionality:

- **CasClientApi**: a new dedicated Python module that wraps interactions with the CAS REST API, including service lookup, creation, update, and deletion operations.
- **Unified HTTP Request Handling**: The `http_request` function was extracted from MitreID and Keycloak deployer agents into a common utility. It standardizes API calls across deployers, handling errors, timeouts, and response parsing. This function improves code maintainability and consistency across different deployer agents.
- **Systemd Service**: a new systemd unit file (`deployer_cas.service`) manages the lifecycle of the CAS deployer agent, ensuring it runs continuously and restarts on failure.
- **Common data processing function**: Common utils function `process_data_generic` introduced in the section 5.1 is used for controlling deployment to both the proxy/IdP and Perun IDM.

Chapter 6

Deployment and testing

This chapter outlines the deployment process and testing strategies used to validate the implemented extensions of the Federation Registry. It describes the steps required to deploy the updated Federation Registry and CAS deployer agent, integrating them with Apereo CAS, Perun IDM, and the ARGO Messaging Service (AMS). Additionally, it covers the deployment of the Apereo CAS and testing service to demonstrate service registration to Apereo CAS.

The schema 6.1 illustrates all applications after deployment together with their interactions.

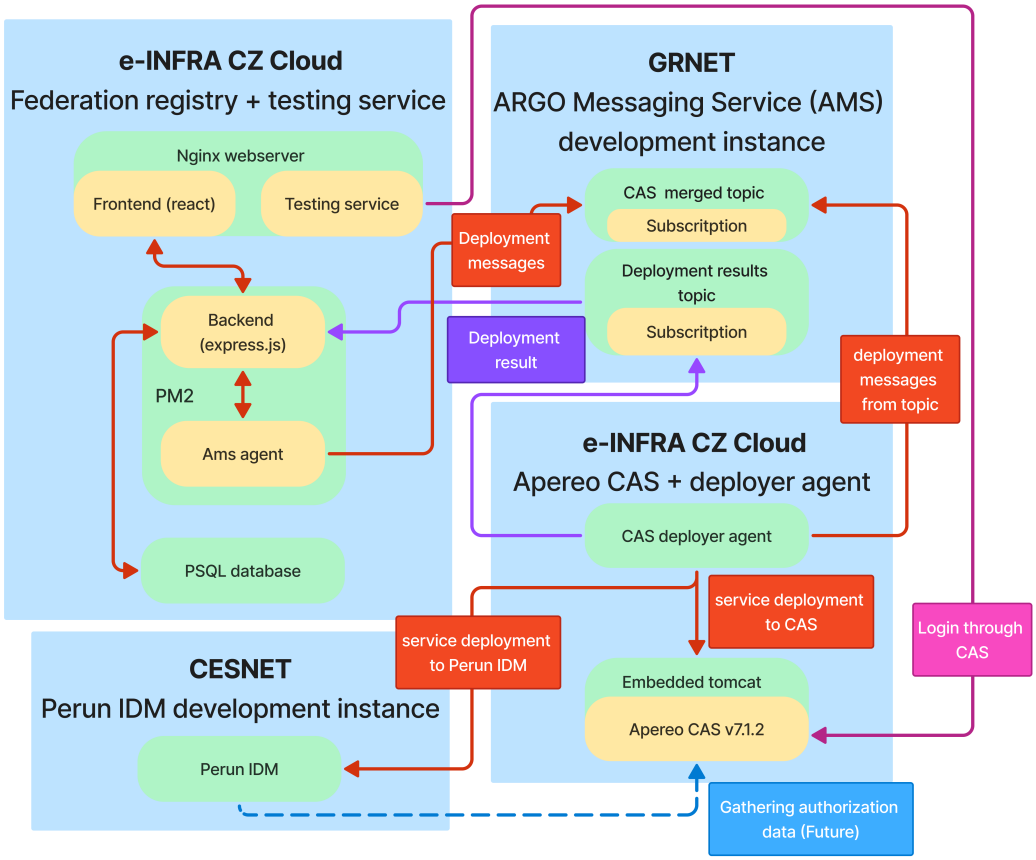

Figure 6.1: Deployment schema

For the deployment, two Debian-based servers running in the e-INFRA CZ Cloud operated by MetaCentrum were used [10]. One server hosts the Federation Registry and testing service, while the second hosts Apereo CAS and the CAS deployer agent. Integration with the ARGO Messaging Service was carried out using the development instance provided by GRNET, while integration with Perun IDM was performed against the development instance provided by CESNET running at `https://gui-dev.perun-aai.org/login`.

The chapter also presents the testing methodology applied to verify the correctness and reliability of the new features, including integration with Perun IDM, the handling of merged deployment environments, and the operation of the newly developed Apereo CAS deployer agent.

All testing was performed manually, simulating real-world scenarios to confirm the correctness of the implementation. This approach ensures that Federation Registry extensions meet the intended functional requirements and operate smoothly in production environments.

6.1 Federation Registry deployment

The Federation Registry was deployed on a Debian-based server running in an OpenStack platform. The deployment began with the installation of the PostgreSQL relational database[1], and creating a database which is used for storing Federation Registry data.

The next step involved obtaining an SSL certificate for the server. Initially, a certificate from Let's Encrypt[2] was used. However, due to validation issues encountered by the ARGO Messaging Service (AMS) development instance, the certificate was later replaced with one issued by the Trusted Certificate Service (TCS) provided by the CESNET Certification authority[3].

Following this, the Nginx web server[4] was installed and configured. The configuration covers redirection from HTTP to HTTPS, SSL setup, routing for the backend and frontend, as well as support for a test service endpoint. It also includes a special redirect to support AMS push endpoint verification by forwarding requests to the appropriate backend handler. The complete Nginx configuration is provided in the appendix A.

The next step was to register the Federation Registry as a service within the Czech national e-infrastructure e-INFRA CZ[5] in order to enable user authentication via federated login. Once the service registration was completed, the necessary credentials were then used in the Federation Registry configuration to allow integration with the identity infrastructure, providing authenticated access.

The final step involved deploying the Federation Registry backend, frontend, and AMS agent. This was accomplished using the `federation-registry.yml` Ansible[6] playbook together with the `federation-registry` role, both provided as part of the RCIAM Ansible playbook collection `rciam-deploy`[7]. All variables and configuration files associated with the `federation-registry` role had to be filled with real values specific to the target deployment environment.

[1] PostgreSQL: `https://www.postgresql.org/`
[2] Let's Encrypt: `https://letsencrypt.org/`
[3] CESNET Certification authority: `https://pki.cesnet.cz/en/intro.html`
[4] Nginx: `https://nginx.org/en/`
[5] e-INFRA CZ: `https://www.e-infra.cz/`
[6] Ansible: `https://docs.ansible.com/`
[7] RCIAM deploy collection: `https://github.com/rciam/rciam-deploy`

The source code used for the deployment was obtained from a fork of the official Federation Registry GitHub repository[8], specifically from the `thesis` branch[9], which contains the implementation of the extensions described in the previous chapter.

After the deployment was completed, the Federation Registry became accessible at https://federation-registry.dev.perun-aai.org/cesnet/home.

6.2 Apereo CAS deployment

Apereo CAS was deployed on a second Debian-based server operated within the OpenStack platform. The initial step involved obtaining an SSL certificate for the server using Let's Encrypt, issued via the Certbot utility[10].

Subsequently, an Apereo CAS overlay was generated using the CAS Initializr[11]. The tool was configured to include all required dependencies, such as support for OIDC and SAML protocols and an embedded Tomcat web server. The generated overlay project included a Dockerfile used to build the CAS Docker image.

Following the overlay generation, the main CAS configuration file, `cas.yaml`, was created and populated with the necessary settings, including enabling API endpoints for service registration and connection to e-INFRA CZ (IdP) as SP. Additionally, a Docker Compose[12] configuration file was prepared to manage the deployment. This file handles tasks such as copying configuration files into the Docker container and mapping the required ports for external access. The Apereo CAS became available at https://cas.dev.perun-aai.org/cas/login after running Docker Compose. The Apereo CAS was running in memory mode, and every restart clears all of the data.

6.3 CAS deployer agent deployment

The CAS deployer agent, responsible for registering services in both Apereo CAS and Perun IDM, was deployed on the same server as the Apereo CAS instance. The deployment process followed a similar approach to the deployment of the Federation Registry. It utilized the same RCIAM Ansible playbook collection, `rciam-deploy`[13], but uses the `fedregagents.yml` Ansible playbook along with the `fedreg-agent` role. The configuration for the CAS deployer was extended according to the implementation requirements and populated with variables to enable integration with the already deployed Federation Registry, Apereo CAS, and the development instance of Perun IDM and ARGO messaging service.

The source code used for the deployment was obtained from a fork of the official RCIAM Federation Registry Agent GitHub repository[14], specifically from the `thesis` branch[15], which contains the implementation of the CAS deployer agent and the Perun IDM extension described in the previous chapter.

[8] Federation Registry repository: https://github.com/rciam/rciam-federation-registry

[9] Federation Registry thesis fork: https://github.com/xpavlic/rciam-federation-registry/tree/thesis

[10] Certbot: https://certbot.eff.org/

[11] CAS Initializr: https://getcas.apereo.org/ui

[12] Docker Compose: https://docs.docker.com/compose/

[13] RCIAM deploy collection: https://github.com/rciam/rciam-deploy

[14] RCIAM Federation Registry Agent: https://github.com/rciam/rciam-federation-registry

[15] RCIAM Federation Registry Agent thesis fork: https://github.com/xpavlic/rciam-federation-registry-agent/tree/thesis

6.4 Testing service deployment

To test service integration with Apereo CAS, a simple test service[16] was deployed. The service enables users to authenticate via Apereo CAS and displays the received user attributes after a successful login. It utilizes an embedded Tomcat server, is exposed through an Nginx endpoint, and runs as a `systemd` service. The service is accessible at https://federation-registry.dev.perun-aai.org/dev-services/login. It is configured to integrate with the deployed Apereo CAS instance using both OIDC and SAML protocols. The corresponding testing service configuration is included in Appendix B.

6.5 Testing

From a performance perspective, the implemented extensions are not expected to impact the overall performance of the Federation Registry, as they do not alter its underlying architecture. The only potential source of additional latency arises when service registration to Perun IDM is enabled, since it involves multiple API calls. However, this additional delay is expected to remain within a few seconds.

Given that service registration is not a time-critical process, unlike, for example, identity provider login availability, and that service administrators can typically tolerate delays of even several minutes for data propagation from the registration system to the proxy/IdP, performance will not be considered as a factor in the testing scenarios.

The testing scenarios simulate the real-world service registration process and, due to the involvement of multiple interacting components, are performed manually. The primary objective of these scenarios is to verify the correctness of the implementation.

All implemented extensions were tested according to the following testing scenarios, and **the system behaved as expected in all cases**.

Scenario one

This scenario simulates the successful registration process for both SAML and OIDC services with the use of the testing service. The result is that the user is able to log in to the testing service through connected Apereo CAS.
Scenario steps for SAML service:

1. Login to the federation registry.
2. Create SAML service registration and use testing service SAML metadata available at: https://federation-registry.dev.perun-aai.org/dev-services/saml2/service-provider-metadata/dev-saml.
3. Approve the service request and wait for the deployment result.
4. Check service registration through Apereo CAS endpoint at https://cas.dev.perun-aai.org/cas/actuator/registeredServices.
5. Find facility by service name or Entity ID in Perun IDM GUI at https://gui-dev.perun-aai.org/facilities and check the managers group and facility attributes.

[16] Test service: https://gitlab.ics.muni.cz/469355/dev-test-service

6. Login in to the testing service at https://federation-registry.dev.perun-aai.org/dev-services/login using SAML.

Scenario steps for OIDC service:

1. Log in to the Federation Registry.
2. Create SAML service registration with
   - Client ID: `ef3fd914-11fb-4965-aaad-5e6438047c10`
   - client secret: `89ef8bb8-c385-41e8-a50c-12e9728fc211`
   - redirect uri: https://federation-registry.dev.perun-aai.org/dev-services/login/oauth2/code/dev-oidc
3. Approve the registration request and wait for the deployment result.
4. Check service registration through Apereo CAS endpoint at https://cas.dev.perun-aai.org/cas/actuator/registeredServices.
5. Find facility by service name or Client ID in Perun IDM GUI at https://gui-dev.perun-aai.org/facilities and check the managers group and facility attributes.
6. Login in to the testing service at https://federation-registry.dev.perun-aai.org/dev-services/login using OIDC.

Similar steps can be used for the updating service registration and service deregistration processes.

Scenario two

This scenario tests the merge of the integration environments and deployment to the single proxy/IdP and Perun IDM instances. It also tests the service identifier uniqueness check between the integration environments. This scenario requires having already registered both SAML and OIDC services. The result is that service from different integration environment is registered in the same proxy/IdP and Perun IDM instance.

Scenario steps for both SAML and OIDC services:

1. Copy the service identifier from an existing service and initiate a new service registration request using the same protocol, but within a different integration environment than the one associated with the original service.
2. Paste service identifier into the identifier field. The registration form should show an error, and submitting will not proceed.
3. Use a different identifier and complete the registration request.
4. Approve the registration request and wait for the deployment result.
5. Service registration should be visible through the Apereo CAS endpoint at `https://cas.dev.perun-aai.org/cas/actuator/registeredServices`.
6. Find facility by service name or service identifier in Perun IDM GUI at `https://gui-dev.perun-aai.org/facilities` and check the managers group and facility attributes.

Scenario three

This scenario verifies the functionality of moving services between integration environments when the merging of environments is enabled. It assumes that a service has already been registered. The objective is to move an existing service to a different integration environment.
Scenario steps:

1. Select an existing service that should be moved, and in its action menu, choose the `Move service` option.
2. In the move dialog, select the target integration environment: choose `production` if the current environment is `demo` or `dev`, or select `demo/dev` if the service is currently in `production`.
3. Fill all required fields in the service form and submit the move request.
4. Approve the submitted request and wait for the deployment process to complete.
5. Verify that the updated service registration appears via the Apereo CAS API endpoint at `https://cas.dev.perun-aai.org/cas/actuator/registeredServices`.
6. In the Perun IDM GUI at `https://gui-dev.perun-aai.org/facilities`, locate the facility by service name or identifier and verify that the `Is test service` attribute is:
   - not set for services in the `production` environment
   - set to `true` for services in the `dev` or `demo` environment

Scenario four

This scenario verifies that the copying service registration always removes the service identifier in service copy form, improving the user experience when the merging of environments is enabled. It assumes that a service has already been registered.
Scenario steps:

1. Select an existing service that should be copied, and in its action menu, choose the `Copy service` option.
2. Verify that the service copy form has an empty Client ID for the OIDC service and an Entity ID for the SAML service.

This scenario verifies that when the merging of integration environments is enabled, copying a service registration automatically removes the service identifier (Client ID or Entity ID) in the copy form to improve the user experience. It assumes that a service is already registered.

Scenario five

This scenario verifies error handling during the deployment process. If the deployment to Apereo CAS fails, the subsequent deployment to Perun IDM must be skipped. Additionally, if the deployment to Apereo CAS succeeds but the deployment to Perun IDM fails, an appropriate error message should be displayed in the Federation Registry. In this case, service redeployment should be possible without triggering a repeated deployment to the proxy/IdP.
Scenario steps for error handling of the failing deployment to proxy/IdP:

1. Turn off the Apereo CAS docker container.
2. Create new service registration.
3. Approve the service registration and initiate the service deployment.
4. The service deployment should fail with an appropriate message.
5. Start the Apereo CAS docker container.
6. Retry service deployment.
7. Verify that service registration is visible through the Apereo CAS endpoint at `https://cas.dev.perun-aai.org/cas/actuator/registeredServices`.
8. Find facility by service name or service identifier in Perun IDM GUI at `https://gui-dev.perun-aai.org/facilities` and check the managers group and facility attributes.
9. Create a new service registration with the same service name as for first service.
10. Approve the service registration and initiate the service deployment.
11. The service deployment should fail when trying to deploy to Perun IDM with an appropriate message. The reason for the error is that the facility name must be unique within Perun IDM.

12. Verify that service registration is visible through Apereo CAS endpoint at https://cas.dev.perun-aai.org/cas/actuator/registeredServices
13. Create the deregistration request for the first service.
14. Approve the deregistration request and wait for the deployment to complete.
15. Verify that service is not visible through the Apereo CAS endpoint at https://cas.dev.perun-aai.org/cas/actuator/registeredServices.
16. Verify that the facility with the service name does not exist in Perun IDM and check that the managers group with the service name does not exist as a subgroup of the `SP_Managers` group.
17. Retry the previously failed service registration deployment and wait for the result.
18. Verify that service was not registered again through the Apereo CAS endpoint at https://cas.dev.perun-aai.org/cas/actuator/registeredServices.
19. Find facility by service name or service identifier in Perun IDM GUI at https://gui-dev.perun-aai.org/facilities and check the managers group and facility attributes.

Chapter 7

Conclusion

This thesis focused on extending the capabilities of the Federation Registry application and its related components to enable integration with Perun AAI identity infrastructures and to add support for the Apereo CAS solution. The work began with an analysis of identity management concepts and protocols to better understand the challenges involved in service registration within identity infrastructures. This foundational knowledge was then applied to examine the architecture, use cases, and service deployment workflows of the Federation Registry and its associated systems. Based on this analysis, three key extensions were designed and implemented. The extension that enables integration with Perun identity management system to support automatic synchronization of registered services with the Perun IDM, the support for merged deployment environments that allows services from all environments (development, demo, production) to be registered to a single proxy/IdP and Perun IDM instance and the Apereo CAS deployer agent that enables service registration to Apereo CAS via its REST API.

All extensions were implemented as configurable modules, allowing Federation Registry administrators to selectively enable them according to the needs of their specific infrastructure. The extended system was deployed on two Debian-based servers within the e-INFRA CZ Cloud platform. The first one hosted the Federation Registry and test service, and the other ran Apereo CAS along with the CAS deployer agent. Integration with the ARGO Messaging Service was performed using a development instance provided by GRNET, and communication with Perun IDM was tested against the development instance operated by CESNET.

The implementation was validated through manual testing, simulating real-world service registration workflows. The testing results confirm that the extended Federation Registry can reliably register services to both Perun IDM and Apereo CAS, while maintaining the expected behavior of existing deployment processes. The tested scenarios demonstrated the robustness, configurability, and adaptability of the solution, improving the interoperability of identity federation systems and simplifying service integration within research and academic infrastructures.

The implemented extensions are planned to be merged into the upstream repositories of the Federation Registry, as agreed with the maintainers. The extended Federation Registry will be integrated into identity infrastructure solutions based on the Perun AAI, replacing

the currently used service registration application[1]. This integration will include the Czech national infrastructure e-INFRA CZ, the Life Science Login infrastructure, and others.

[1]SP registration application: https://gitlab.ics.muni.cz/perun/perun-proxyidp/v1/perun-spRegistration-app

Bibliography

[1] Aldosary, M. and Alqahtani, N. A Survey on Federated Identity Management Systems Limitation and Solutions. *International Journal of Network Security & Its Applications (IJCNC)* online. Riyadh: AIRCC Publishing Corporation, may 2021, vol. 13, no. 3, p. 43–59. ISSN 0974–9330. Available at: https://doi.org/10.5121/ijnsa.2021.13304. [cit. 2024-01-13].

[2] Apereo Foundation. *7.1.0-RC6 Release Notes* online. 2025. Available at: https://apereo.github.io/cas/7.1.x/release_notes/RC6.html#palantir-admin-console. [cit. 2025-02-21].

[3] Apereo Foundation. *Apereo CAS – Identity & Single Sign-On* online. 2025. Available at: https://apereo.github.io/cas/7.1.x/index.html. [cit. 2025-02-09].

[4] Apereo Foundation. *Service Management* online. 2025. Available at: https://apereo.github.io/cas/7.1.x/services/Service-Management.html. [cit. 2025-02-22].

[5] Apereo Foundation. *Service Management* online. 2025. Available at: https://apereo.github.io/cas/7.1.x/services/Service-Management.html. [cit. 2025-02-20].

[6] CESNET, z. s. p. o.. *Login process* online. 29. june 2020. Available at: https://aai.cesnet.cz/en/index/documentation/sp/proxy/federated_login. [cit. 2025-01-31].

[7] CESNET, z. s. p. o.. *Proxy IdP architecture* online. 07. october 2020. Available at: https://aai.cesnet.cz/en/index/documentation/sp/proxy/proxy-architecture. [cit. 2025-01-31].

[8] CESNET, z. s. p. o.. *The Perun AAI* online. 2025. Available at: https://www.cesnet.cz/en/services/identity-6/the-perun-aai-25. [cit. 2025-01-31].

[9] Chadwick, D. W. Federated Identity Management. In: Aldini, A.; Barthe, G. and Gorrieri, R., ed. *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures* online. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5705, p. 96–120. Lecture Notes in Computer Science. ISBN 978-3-642-03829-7. Available at: https://doi.org/10.1007/978-3-642-03829-7_3. [cit. 2025-05-09].

[10] e-INFRA CZ. *E-INFRA CZ OpenStack in Brno* online. 2025. Available at: https://brno.openstack.cloud.e-infra.cz/. [cit. 2025-05-01].

[11] EDUCAUSE, Internet2, and the National Science Foundation. *EduPerson 2020-01* online. 13. august 2020. Available at: https://wiki.refeds.org/display/STAN/eduPerson+2020-01. [cit. 2025-01-16].

[12] Garey, L. *What Is Digital Identity?* online. 19. september 2024. Available at: https://www.oracle.com/uk/security/identity-management/digital-identity/. [cit. 2025-03-27].

[13] GRNET. *Federation Registry Documentation* online. 2022. Available at: https://federation.rciam.grnet.gr. [cit. 2024-01-14].

[14] GRNET. *Rciam-federation-registry-agent* online. 08. may 2023. Available at: https://github.com/rciam/rciam-federation-registry-agent/blob/master/README.md. [cit. 2025-05-08].

[15] GRNET. *AMS – The Service* online. 2024. Available at: https://argoeu.github.io/argo-messaging/docs. [cit. 2024-01-15].

[16] GRNET. *Company* online. 2025. Available at: https://grnet.gr/en/company/. [cit. 2025-03-28].

[17] Keycloak Authors. *Open Source Identity and Access Management* online. 18. october 2024. Available at: https://github.com/keycloak/keycloak?tab=readme-ov-file#readme. [cit. 2025-01-14].

[18] Kuba, M. *Perun AAI* online. 10. may 2022. Available at: https://www.e-infra.cz/file/d44b6a06f6c77e2224b241aa67db4870/629/Kuba.pdf. [cit. 2025-05-08].

[19] Lewis, K. D. and Lewis, J. E. Web Single Sign-On Authentication using SAML. *International Journal of Computer Science (IJCSI)* online. IJCSI Press, september 2009, vol. 2, p. 41–48. ISSN 1694-0784. Available at: https://doi.org/10.48550/arXiv.0909.2368. [cit. 2024-01-13].

[20] Liampotis, N. and Fernández, E. *Service Providers* online. 03. march 2024. Available at: https://docs.egi.eu/providers/check-in/sp/#service-provider-integration-workflow. [cit. 2025-05-09].

[21] Linden, M. and Bucik, D. F. *Documentation – How to connect a service to the Life Science AAI* online. 25. march 2024. Available at: https://docs.google.com/document/d/17pNXM_psYOP5rWF302ObAJACsfYnEWhjvxAHzcjvfIE. [cit. 2025-05-09].

[22] Maler, E. and Reed, D. The Venn of Identity: Options and Issues in Federated Identity Management. *IEEE Security & Privacy* online. IEEE, april 2008, vol. 6, no. 2, p. 16–23. ISSN 1558-4046. Available at: https://doi.org/10.1109/MSP.2008.50. [cit. 2024-01-13].

[23] Masaryk University. *About Perun* online. 2025. Available at: https://perun-aai.org/about-perun/overview. [cit. 2025-01-31].

[24] Masaryk University. *How to use Perun RPC* online. 22. january 2025. Available at: https://perun-aai.org/documentation/technical-documentation/rpc-api/index.html. [cit. 2025-02-02].

[25] OASIS. *Security Assertion Markup Language (SAML) V2.0 Technical Overview* online. 25. march 2008. Available at: https://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html. [cit. 2024-05-09].

[26] OASIS. *OSASIS SAML Wiki* online. 26. june 2020. Available at: https://wiki.oasis-open.org/security/FrontPage. [cit. 2024-01-13].

[27] Pimenidis, E. Digital Identity Management. In: Jahankhani, H.; Watson, D. L.; Me, G. and Leonhardt, F., ed. *Handbook of Electronic Security and Digital Forensics* online. 1st ed. World Scientific Books, March 2010, p. 279–294. ISBN 978-981-283-703-5. Available at: https://doi.org/10.1142/9789812837042_0015. [cit. 2025-05-09].

[28] Security Journey. *Common Federated Identity Protocols: OpenID Connect vs OAuth vs SAML 2* online. 27. december 2019. Available at: https://www.securityjourney.com/post/analysis-of-common-federated-identity-protocols-openid-connect-vs-oauth-2.0-vs-saml-2.0. [cit. 2025-03-27].

[29] simplesamlphp. *SimpleSAMLphp* online. 02. december 2024. Available at: https://simplesamlphp.org. [cit. 2025-01-14].

[30] Siriwardena, P. *Advanced API Security: OAuth 2.0 and Beyond*. 1st ed. New York: Apress Berkeley, CA, december 2017. 449 p. Books for professionals by professionals, no. 2. ISBN 978-1-4842-2049-8.

[31] Sweeney, P. and Gittlen, S. *What is identity and access management? Guide to IAM* online. December 2024. Available at: https://www.techtarget.com/searchsecurity/definition/identity-access-management-IAM-system. [cit. 2025-03-27].

[32] Thakur, M. A. and Gaikwad, R. User identity and Access Management trends in IT infrastructure- an overview. In: Institute of Electrical and Electronics Engineers (IEEE). *2015 International Conference on Pervasive Computing (ICPC)* online. January 2015, p. 1–4. ISBN 9781479960545. Available at: https://doi.org/10.1109/PERVASIVE.2015.7086972. [cit. 2025-05-09].

[33] The MITRE Corporation. *About MITREid Connect* online. 09. february 2018. Available at: https://mitreid-connect.github.io. [cit. 2025-01-14].

[34] Waluyo, T. and Sutarman. Comparative Analysis of the Performance of Single Sign-On Authentication Systems with OpenID and OAuth Protocols: Application of Single Sign-On (SSO) in Information Systems at the University of Technology Yogyakarta. *International Journal of Computer and Information Technology (2279-0764)* online. Lucknow: International Journal of Computer and Information Technology, august 2022, vol. 11, no. 3. ISSN 2279-0764. Available at: https://doi.org/10.24203/ijcit.v11i3.277. [cit. 2025-05-09].

[35] Windley, P. J. *Defining Digital Identity*. 1st ed. O'Reilly Media, august 2005. 8–14 p. ISBN 978-0-596-00878-9.

Appendix A

Nginx configuration for Federation Registry

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name federation-registry.dev.perun-aai.org;

    # Redirect all HTTP requests to HTTPS
    return 301 https://$host$request_uri;
}
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name federation-registry.dev.perun-aai.org;

    ssl_certificate /etc/ssl/federation-registry/server.crt;
    ssl_certificate_key /etc/ssl/federation-registry/server.key;
    ssl_password_file /etc/ssl/federation-registry/password;

    root /var/www/rciam-federation-registry/federation-registry-frontend;
    index index.html index.htm index.nginx-debian.html;

    location /dev-services {
        proxy_pass https://127.0.0.1:8080/dev-services;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location / {
        try_files $uri $uri/ /index.html;
    }

    location = /ams_verification_hash {
        return 302 /federation-backend/ams/ams_verification_hash;
    }
```

```
    location = /federation-backend {
        return 302 /federation-backend/;
    }

    location /federation-backend/ {
        proxy_pass_request_headers on;
        proxy_set_header Host $host;
        proxy_pass http://localhost:5000/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header DN $ssl_client_s_dn;
    }
}
```

Listing A.1: Nginx configuration for `federation-registry.dev.perun-aai.org`

Appendix B

Testing service configuration

```
server:
  port: 8080
  servlet:
    context-path: "/dev-services"
  ssl:
    key-store: "/etc/ssl/federation-registry/keystore.p12"
    key-store-password: "<ANONYMIZED>"
spring:
  application:
    name: Saml2TestService
  security:
    saml2:
      relyingparty:
        registration:
          dev-saml:
            signing:
              credentials:
                private-key-location: classpath:local.key
                certificate-location: classpath:local.crt
            singlelogout:
              binding: POST
              url: "{baseUrl}/saml2/logout"
              response-url: "{baseUrl}/logout/saml2/slo"
            assertingparty:
              metadata-uri: "https://cas.dev.perun-aai.org/cas/idp/metadata"
    oauth2:
      client:
        registration:
          dev-oidc:
            provider: cas-local
            client-id: ef3fd914-11fb-4965-aaad-5e6438047c10
            client-secret: 89ef8bb8-c385-41e8-a50c-12e9728fc211
            authorization-grant-type: authorization_code
            scope: openid,profile,email
        provider:
          cas-local:
            issuer-uri: https://cas.dev.perun-aai.org/cas/oidc
```

Listing B.1: Testing service configuration