PROCEEDINGS OF SCIENCE

PoS(ISGC2026)013

Federation Manager: a New Component for the Federated Resource Management in INFN Cloud

Savarese Giovanni,[a,*] **Alessandra Casale,**[c] **Alessandro Costantini,**[b] **Giacinto Donvito,**[a] **Jacopo Gasparetto,**[b] **Luca Giommi**[b] **and Ettore Serra**[a]

[a] *INFN Sezione di Bari,*
Via Edoardo Orabona 4, 70125, Bari, Italy
[b] *INFN-CNAF,*
Viale Berti Pichat 6/2, 40127, Bologna, Italy
[c] *INFN-LNGS,*
Via G. Acitelli 22, 67100, L'Aquila, Italy
E-mail: giovanni.savarese@ba.infn.it, alessandra.casale@lngs.infn.it, alessandro.costantini@cnaf.infn.it, giacinto.donvito@ba.infn.it, jacopo.gasparetto@cnaf.infn.it, luca.giommi@cnaf.infn.it, ettore.serra@ba.infn.it

*Speaker

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0) All rights for text and data mining, AI training, and similar technologies for commercial purposes, are reserved. ISSN 1824-8039 . Published by SISSA Medialab.

As part of the strategic refactoring and modernization of the INFN Cloud orchestration system, the Federation Manager has been developed to enhance the flexibility, scalability, and interoperability of the distributed DataCloud infrastructure. This initiative represents a key step in the long-term evolution of INFN Cloud toward a more modular, service-oriented architecture capable of supporting hybrid and multi-cloud orchestration across heterogeneous resource providers. Within this context, the Federation Manager plays a central role by automating the federation of new providers and managing resource access requests from scientific communities in a unified and secure procedure.

The service architecture is built on a Python backend, leveraging the FastAPI framework to implement RESTful APIs that follow best practices and relevant RFC standards. This ensures high performance, consistency, and ease of integration with other components of the INFN Cloud. A modern, responsive web interface, developed with React and Next.js, provides users with a clear and intuitive experience for managing provider federations and resource allocations. Designed using Figma, the interface focuses on scalability and maintainability, aligning with the shared design principles adopted across the PaaS ecosystem.

Authentication and authorization mechanisms are based on OAuth2 and OpenID Connect (OIDC), ensuring secure, standards-compliant identity management. Centralized and flexible policy enforcement is achieved through Open Policy Agent (OPA), while Apache Kafka handles asynchronous communication with other PaaS components, guaranteeing reliability and scalability in data exchange.

A dedicated Python-based monitoring component complements the Federation Manager by periodically executing Rally benchmarking tests on the federated cloud providers. This module automates the evaluation of performance indicators such as compute, storage, and network efficiency, collecting metrics that are stored and analyzed to detect anomalies or degradation in service quality. The integration of this testing component provides continuous insight into the health and reliability of the federated infrastructure, supporting proactive maintenance and capacity planning.

By defining reusable API specifications, adopting shared UI frameworks, and integrating modern security and policy technologies, the Federation Manager contributes to a coherent and sustainable technical framework for INFN Cloud. This unified approach simplifies the integration of new services, improves maintainability, and strengthens the overall interoperability of the platform. Ultimately, the Federation Manager supports INFN Cloud's mission to provide a robust, extensible, and standards-aligned infrastructure for data-intensive scientific research in federated and multi-cloud environments.

International Symposium on Grids and Clouds (ISGC2026)
15-20, March, 2026
Academia Sinica, Taipei, Taiwan

PoS(ISGC2026)013

1. Introduction

The increasing scale, heterogeneity, and geographic distribution of modern scientific computing infrastructures have driven the adoption of federated cloud models, where multiple resource providers collaborate to deliver unified services to user communities. In this context, initiatives such as the INFN DataCloud aim to integrate diverse computing and storage resources into a coherent platform capable of supporting data-intensive research across disciplines [1]. However, the federation of independent infrastructures introduces significant challenges in terms of interoperability, policy enforcement, identity management, and resource orchestration.

To address these challenges, the Federation Manager has been developed as a core component of the INFN Cloud ecosystem [2]. It emerges from the ongoing refactoring and modernization of the orchestration layer, with the goal of enabling a more modular, scalable, and service-oriented architecture for federated environments [3]. The Federation Manager is designed to automate and govern the life-cycle of federation processes, including the on-boarding of new providers and the management of resource access requests from scientific communities through unified and secure procedures.

A key aspect of the Federation Manager is its ability to abstract the complexity of heterogeneous and geographically distributed infrastructures, providing a consistent interface for both providers and users. By integrating policy enforcement, identity and access management, and service-level agreement (SLA) handling, the system ensures coherent operation across multiple administrative domains. This approach enables a user-centric model in which resources and services can be dynamically discovered and consumed, fostering interoperability and simplifying the integration of new services.

From a technical perspective, the Federation Manager adopts modern design principles based on RESTful APIs and modular microservices, leveraging contemporary frameworks to enhance maintainability and extensibility. It also incorporates monitoring and evaluation mechanisms to assess performance indicators—such as compute, storage, and network efficiency—thereby supporting proactive infrastructure management and capacity planning.

Overall, the Federation Manager represents a key enabler for the evolution of federated cloud infrastructures, contributing to the sustainability, scalability, and interoperability of the INFN DataCloud. By providing a unified and standards-aligned framework for resource federation [4], it supports the broader objective of delivering robust and flexible platforms for data-driven scientific research in multi-cloud environments.

2. System Architecture and Workflow

The Federation Manager adopts a modular, service-oriented architecture designed to support scalability, interoperability, and automation across federated infrastructures. As illustrated in Figure 1, the system is organized into three main layers: a frontend interface, a backend service layer, and an event-driven messaging and monitoring subsystem.

The frontend is implemented using Next.js [5] with server-side rendering capabilities, providing a responsive and user-friendly interface for both site administrators and operators. Through this interface, users can submit federation requests, monitor provider status, and interact with the system

PoS(ISGC2026)013

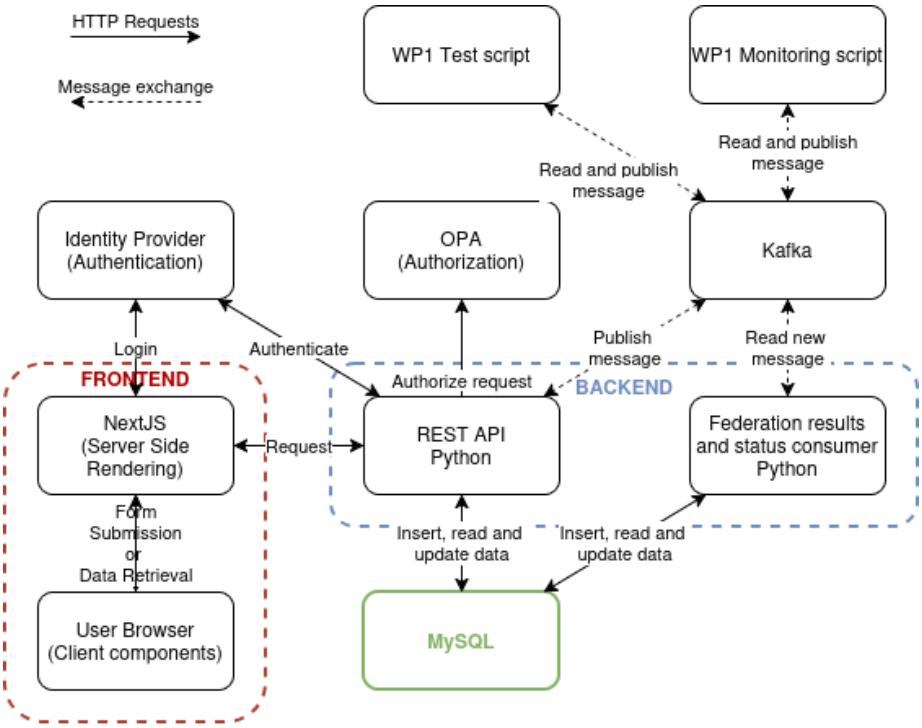


Figure 1: Architectural design of the Federation-manager.

via standard HTTP requests. Authentication is delegated to an external Identity Provider, ensuring secure access through established authentication protocols and enabling integration with federated identity systems.

The backend is built around a Python-based REST API that acts as the central orchestration component. It handles incoming requests from the frontend, enforces business logic, and coordinates interactions with external services. Authorization decisions are delegated to Open Policy Agent (OPA) [6], which evaluates access control policies in a decoupled and flexible manner. Persistent data management is handled through a MySQL [7] database, where information about providers, federation states, and monitoring results is stored and updated.

A key feature of the architecture is its event-driven communication model, implemented through Apache Kafka [8]. The backend publishes events related to provider life-cycle changes and monitoring activities to Kafka topics. External components, such as testing frameworks or monitoring agents, consume and produce messages asynchronously. A dedicated *test results consumer* service processes incoming results, updating the system state and triggering subsequent actions when necessary. This decoupled approach improves scalability and fault tolerance while enabling continuous monitoring and automated decision-making.

The provider life-cycle managed by the Federation Manager follows a structured, state-driven workflow that combines manual actions performed via the user interface with automated or semi-automated transitions triggered by validation and monitoring components. The workflow, shown in Figure 2, introduces a more granular progression from on-boarding to steady-state operation, including explicit validation and SLA negotiation steps.

The life-cycle begins in the DRAFT state, where a *site administrator* registers a new provider

PoS(ISGC2026)013

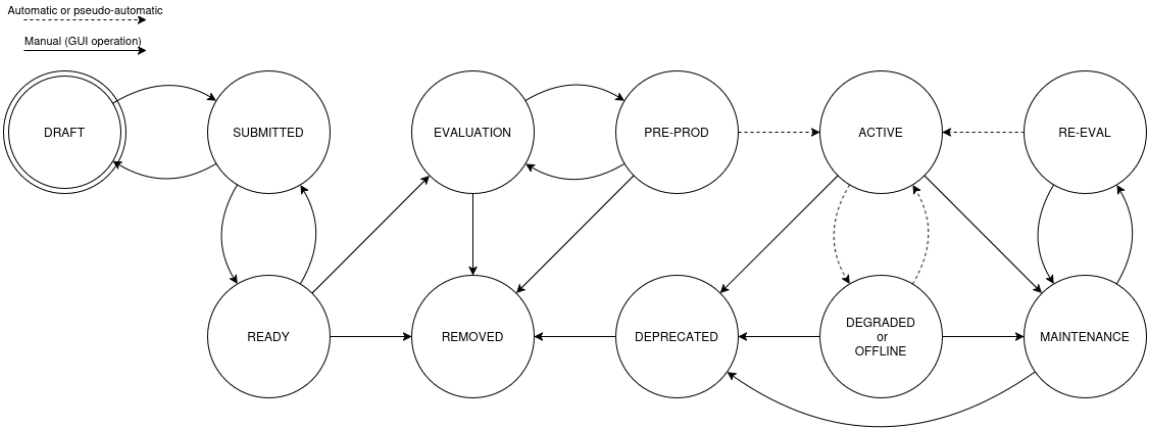


Figure 2: Finite state machine of the federation procedure of a new resource provider.

within the system. At this stage, key requirements, such as the definition of an SLA for the root project (i.e., the reference project used to validate the provider's eligibility for federation), may still be incomplete. The site administrator can freely modify or withdraw the provider entry, resulting in its removal from the system.

Once the federation request is formally submitted, the provider transitions to the SUBMITTED state. Here, the information undergoes an initial verification phase. An SLA moderator reviews the submission, either approving the SLA and associating it with the root project or requesting modifications. This step ensures that policy and service-level requirements are satisfied before proceeding.

After SLA approval, the provider enters the READY state, indicating that it is eligible for testing. At this point, the provider is assigned to testing activities and moves into the EVALUATION phase. During evaluation, a combination of manual and automated tests—executed by external validation services—assesses compliance with federation requirements. If the provider fails to meet the criteria, the request may be revoked, leading to the REMOVED state. Otherwise, successful validation allows progression to the PRE-PRODUCTION (PRE-PROD) phase.

In the PRE-PROD stage, testing has concluded and the system collects operational metrics over a predefined observation period (e.g., for ranking or quality assessment purposes) [9]. If monitoring detects no issues, the provider is automatically promoted to the ACTIVE state, where it becomes a fully integrated and operational member of the federation.

While in the ACTIVE state, continuous monitoring ensures service reliability. If transient or persistent issues are detected, the provider may transition to a DEGRADED or OFFLINE state, indicating partial or complete service disruption. Depending on the severity and expected resolution time, administrators or testers may place the provider into MAINTENANCE, during which corrective actions are performed. Monitoring continues throughout this phase.

Once issues are resolved, the provider does not immediately return to active service; instead, it enters a RE-EVALUATION (RE-EVAL) phase. This step ensures that the provider again satisfies all operational requirements before being reinstated as ACTIVE. This additional validation loop improves robustness and prevents unstable resources from re-entering production prematurely.

At any point in the life-cycle, a provider may be intentionally removed by administrators.

PoS(ISGC2026)013

In such cases, it transitions to the DEPRECATED state, which serves as a grace period during which users can still access and retrieve their resources. After this period expires, the provider is permanently transitioned to the REMOVED state and is no longer part of the federation.

This refined workflow introduces a submission checkpoint, and a re-evaluation loop, thereby strengthening governance and reliability. Combined with automated monitoring and event-driven state transitions, it enables the Federation Manager to enforce consistent federation policies while maintaining flexibility and operational control.

3. User Interface Design and Implementation

The Federation Manager user interface (UI), depicted in Figure 3 was developed using a modern web technology stack to ensure scalability, maintainability, and a responsive user experience. The frontend is built with the React library [10], leveraging its component-based architecture to enable modular development and efficient state management. This approach facilitates code reuse and simplifies the integration of complex interactive elements.

To support server-side rendering and optimized performance, the application is structured using the Next.js framework. This choice enables improved load times, better search engine visibility, and streamlined routing, all of which are critical for a distributed federation platform where usability and accessibility are essential.

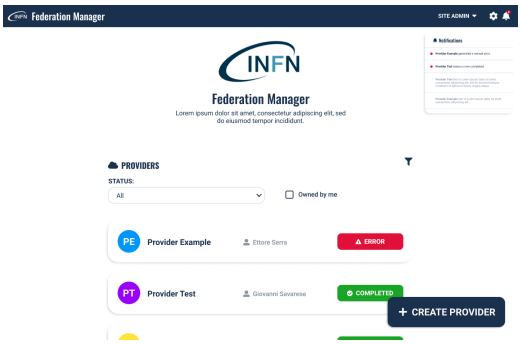
For component design, the interface was built using the Headless UI library [11], providing unstyled, fully accessible UI primitives. This approach enables complete control over the visual design while ensuring robust accessibility and interaction patterns by default. Headless UI integrates naturally with utility-first styling approaches, allowing components to be composed and styled consistently across the platform without imposing a fixed design system. As a result, the implementation achieves a high degree of flexibility, maintainability, and design coherence while reducing the need for custom-built interactive components.

Authentication and access control are managed using the BetterAuth library [12], which provides a secure and flexible mechanism for handling user sessions. A role-based access control model is implemented to enforce separation of concerns among users. Depending on their assigned roles, users are granted access to specific pages and functionalities, ensuring that sensitive operations are restricted to authorized personnel while maintaining a clear navigation structure.

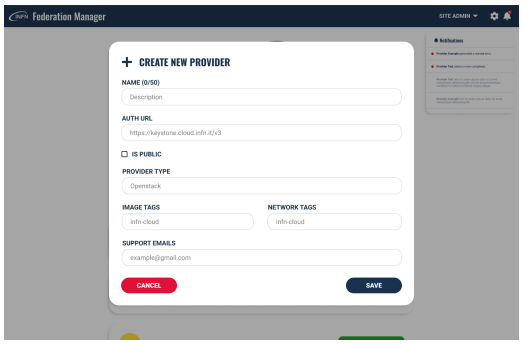
A key feature of the UI is the implementation of a carousel-based workflow for resource provider registration and management. This interactive component guides users through a sequence of steps required to input and validate information. By structuring the process as a step-by-step progression, the system reduces cognitive load, minimizes input errors, and enhances overall usability. The carousel design also allows users to navigate between steps easily, review previously entered data, and complete the process in a structured manner.

The design process of the user interface was supported by the use of Figma [13], a collaborative design and prototyping platform. Figma was employed to create high-fidelity mockups and interactive prototypes of the UI before implementation. This enabled iterative refinement of layout, component structure, and user flows in close collaboration with stakeholders. By validating design decisions early in the development cycle, the team was able to ensure consistency between the

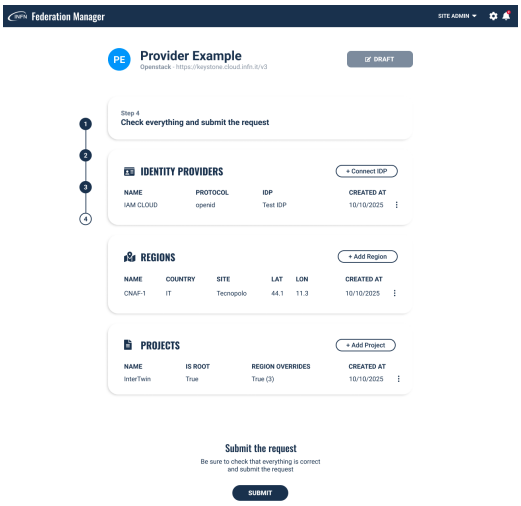
PoS(ISGC2026)013



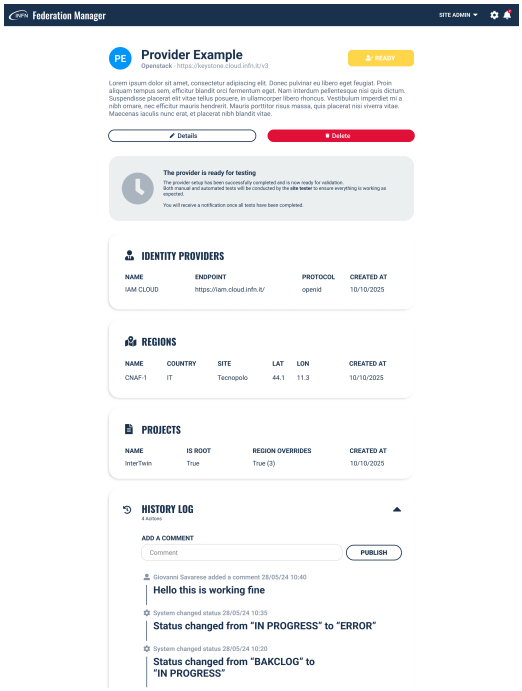
(a) List of resource providers. Site administrator view



(b) Form to create a new resource provider. Site administrator view



(c) Sum up of the submission details. Site administrator view



(d) Status of the federation process. Site administrator view

Figure 3: Federation Manager user interface.

conceptual design and the final implementation, while also reducing development overhead and improving overall usability.

Overall, the combination of these technologies and design choices results in a robust, user-friendly interface that supports the operational requirements of the Federation Manager while maintaining high standards of usability and security.

PoS(ISGC2026)013

4. Backend Design and Implementation

The backend of the Federation Manager is designed to ensure security, scalability, and maintainability through the integration of asynchronous processing, policy-driven access control, and reliable data management strategies.

The backend architecture is designed as a modular and scalable service-oriented system, implemented in Python. At its core, it exposes a RESTful API built using the FastAPI [14] framework, chosen for its high performance, asynchronous capabilities, and automatic generation of OpenAPI-compliant documentation. This enables efficient handling of client requests while maintaining clear and well-documented interfaces between components.

User authentication is managed through the flaat [15] Python module, which provides integration with federated identity providers and supports token-based authentication mechanisms. This ensures that only verified users can access the system, aligning with the federated nature of the platform.

Authorization is handled separately from authentication, following a clear separation of concerns. The backend communicates with an external OPA server to evaluate access control policies. This approach enables fine-grained, policy-driven authorization decisions that can be updated independently of the application logic, improving flexibility and maintainability.

Persistent data management is implemented using SQLAlchemy Python library [16] as Object-Relational Mapping (ORM) layer provides an abstraction over the underlying relational database. The system relies on a MySQL database for storage, offering reliability and widespread support. Database schema evolution is handled through the Alembic Python module [17], which enables version-controlled migrations and consistent deployment of schema updates across environments.

To support asynchronous processing and integration with external systems, the backend includes a dedicated asynchronous thread responsible for consuming and producing messages from an Apache Kafka cluster. This component handles the ingestion and dissemination of test suite results, enabling decoupled communication between the Federation Manager and other services. Communication with Kafka is secured, ensuring the confidentiality and integrity of exchanged messages through appropriate encryption and authentication mechanisms.

5. Validation Test Suite and Monitoring

The Federation Manager integrates a validation framework based on a periodic test suite designed to assess the operational status and compliance of federated resource providers. This test suite is implemented as a Python script that is executed at regular intervals, enabling continuous verification of provider functionality and early detection of potential issues. The validation test suite plays a critical role in maintaining the integrity and performance of the federation by continuously verifying that participating providers meet the expected operational standards.

The testing framework is built upon OpenStack Rally [18], a widely adopted tool for benchmarking and validating cloud environments. It is currently tailored to support providers based on OpenStack [19], reflecting the primary target infrastructure of the federation at this stage.

The suite performs a range of functional tests across key OpenStack components, including OpenStack Glance, OpenStack Neutron, and OpenStack Nova. These tests are designed to validate

PoS(ISGC2026)013

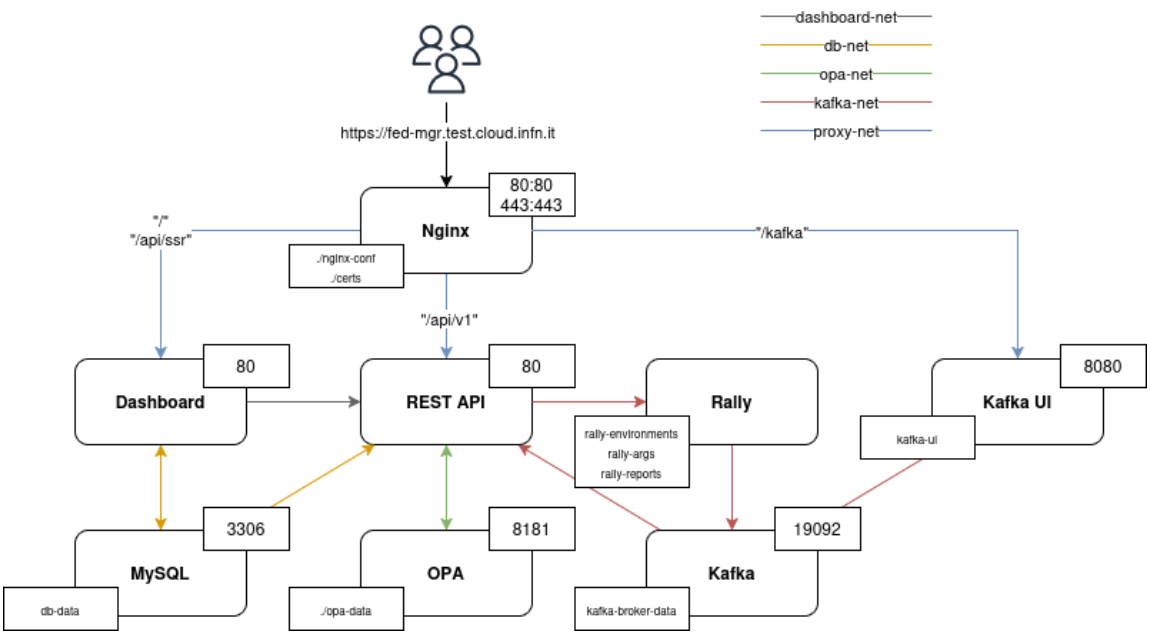


Figure 4: Instances deployed using Docker compose.

core cloud operations such as virtual machine instantiation, image handling, volume management, and IP address allocation. By simulating real-world usage scenarios, the framework provides a practical assessment of the provider's readiness and reliability.

The periodic execution of the test suite ensures that changes in the provider infrastructure, whether due to updates, misconfigurations, or failures, are promptly detected. The results of these tests are then propagated through the system (e.g., via messaging infrastructure) to inform monitoring components and support decision-making processes within the federation.

6. Deployment strategies

The Federation Manager is deployed as a set of loosely coupled microservices, following a container-based architecture that ensures modularity, portability, and reproducibility. The system is currently provisioned using Docker containers [20], as shown in Figure 4, with a parallel set of manifests designed for deployment on Kubernetes clusters [21]. Future continuous delivery workflows are planned through integration with ArgoCD [22].

At the edge of the system, an NGINX reverse proxy handles all incoming traffic from external clients. It performs TLS termination and routes requests to internal services based on URL paths. Specifically, requests targeting the root context (e.g., / or /api/ssr) are forwarded to the Dashboard service, while API requests (e.g., /api/v1) are directed to the REST API component. A dedicated route is also configured to expose Kafka-related endpoints.

Persistent storage is provisioned through dedicated volumes attached to stateful services. A MySQL database is used to store application state and metadata, while Kafka maintains its own broker data. OPA also persists policy-related data locally. These storage components are iso-

PoS(ISGC2026)013

lated within the container network but remain accessible to the relevant services via well-defined interfaces.

A Kafka UI component is available (in development mode only) to provide observability into the messaging layer, enabling inspection of topics, brokers, and message flows. Internal communication between services is organized through logically separated Docker networks (e.g., database, messaging, and proxy layers), improving isolation and simplifying service discovery.

To support scalability and cloud-native deployment, the same architecture has been translated into Kubernetes manifests, including Deployments, Services, and PersistentVolumeClaims. This enables the Federation Manager to run on distributed clusters with automated scaling and resilience features. In the next phase, we plan to adopt a GitOps approach using Argo CD, allowing declarative, version-controlled deployments and automated synchronization between the desired and actual system states.

In conclusion, this deployment strategy provides a flexible pathway from local development environments to production-grade, orchestrated infrastructure, while maintaining consistency across platforms.

7. Conclusions

The Federation Manager represents a flexible and extensible solution for supporting the integration and operation of distributed resource providers within a federated environment. By combining automated validation mechanisms, fine-grained access control, and a user-friendly interface, it streamlines the federation procedure and lowers the barrier for on-boarding new providers. The platform enables consistent verification of provider capabilities, ensuring that only compliant and reliable resources are exposed within the federation. This, in turn, strengthens trust among participants and improves the overall quality of service delivery.

A key area of future development concerns the design of advanced features tailored to community-specific requirements. This aspect is intentionally left open to allow customization based on the needs of different user groups. The goal is to introduce a layer that can be parameterized according to customer or community demands, enabling the definition of service level agreements, policies, and service integrations. Such flexibility will be essential to accommodate heterogeneous requirements while maintaining a coherent and scalable federation framework.

The Federation Manager lays the foundation for a robust and adaptable federation ecosystem, with strong potential to evolve in response to emerging needs and to support a growing range of providers and scientific communities.

References

[1] D. Salomoni, I. Campos, L. Gaido et al. *INDIGO-DataCloud: a Platform to Facilitate Seamless Access to E-Infrastructures*. *J Grid Computing* **16**, 381–408 (2018).

[2] INFN Cloud Home Page, https://www.cloud.infn.it/, (last access: 2026-04-23).

[3] G. Savarese, L. Giommi, et al. *Plan for a renewed PaaS Orchestration solution in the DataCloud Project at INFN*. *PoS* **ISGC2025**, 016 (2025).

PoS(ISGC2026)013

[4] G. Savarese, M. Antonacci, L. Giommi et al. *Federation-registry: the renovated Configuration Management Database for dynamic cloud federation*. *PoS* **ISGC2024**, 021 (2024).

[5] Next.js Home Page, https://nextjs.org/, (last access: 2026-04-23).

[6] Open Policy Agent Home Page, https://www.openpolicyagent.org/, (last access: 2026-04-23).

[7] MySQL Home Page, https://www.mysql.com/en/, (last access: 2026-04-23).

[8] Apache Kafka Home Page, https://kafka.apache.org/, (last access: 2026-04-23).

[9] L. Giommi, F. Debiase, G. Savarese et al. *An AI-based approach for provider selection in the INDIGO PaaS Orchestration System of the INFN Cloud*. *EPJ Web Conf.* **337**, 01160 (2025).

[10] React Home Page, https://react.dev/, (last access: 2026-04-23).

[11] HeadlessUI Home Page, https://headlessui.com/, (last access: 2026-04-23).

[12] BetterAuth Home Page, https://better-auth.com/, (last access: 2026-04-23).

[13] Figma Home Page, https://www.figma.com/design/, (last access: 2026-04-23).

[14] FastAPI Home Page, https://fastapi.tiangolo.com/, (last access: 2026-04-23).

[15] Flaat Home Page, https://flaat.readthedocs.io/en/latest/, (last access: 2026-04-23).

[16] SQLAlchemy Home Page, https://www.sqlalchemy.org/, (last access: 2026-04-23).

[17] Alembic Home Page, https://alembic.sqlalchemy.org/en/latest/, (last access: 2026-04-23).

[18] Rally Home Page, https://rally.readthedocs.io/en/latest/, (last access: 2026-04-23).

[19] Openstack Home Page, https://www.openstack.org/, (last access: 2026-04-23).

[20] Docker Home Page, https://www.docker.com/, (last access: 2026-04-23).

[21] Kubernetes Home Page, https://kubernetes.io/, (last access: 2026-04-23).

[22] ArgoCD Home Page https://argo-cd.readthedocs.io/en/stable/, (last access: 2026-04-23).

PoS(ISGC2026)013