UNICO I+D Project
6G-INTEGRATION 02

6G-INTEGRATION-02-E12

Federation analysis and proposal

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia

Document properties

Document number	6G-INTEGRATION-02-E12
Document title	Federation analysis and proposal
Document responsible	Monica Villalobos, Yaima Fiallo (Capgemini Engineering)
Document editor	Alvaro Redondo, Ernesto Correa, Yaima Fiallo, Noel Ruiz, Miguel Juaniz (Capgemini Engineering)
Target dissemination level	Private
Status of the document	Final
Version	1.0
Delivery date	30/04/2024
Actual delivery date	30/11/2023 (Partial)

Production properties

Reviewers

Disclaimer

This document has been produced in the context of the 6G-INTEGRATION Project. The research leading to these results has received funding from the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D programme.

Contents

List of Figures

List of Acronyms

APIs: Application Programming Interfaces

BS: Base Station

CNCF: Cloud Native Computing Foundation

CPU: Central Processing Unit

CRC: Cyclic Redundancy Checks

CRI: Container Runtime Interface

ECM: Enterprise Content Management

FeDeCoP: Federation Deployment Control Protocol

FIM: Federated Identity Management

FSS: Federation of Satellite Systems

GEO: Geostationary Earth Orbiting

GIS: Geographic Information Systems

HAPS: High Altitude Platforms

HTTPS: Hypertext Transfer Protocol Secure

IaaS: Infrastructure as a Service

ISL: Inter-satellite links

IoT: Internet of Things

IT: Information Technology

LDAP: Lightweight Directory Access Protocol

LEO: Low Earth Orbiting

NTN: Non-Terrestrial Networks

PaaS: Platforms as a Service

PoC: Proof of concept

SAR: Synthetic Aperture Radar

TNs: Terrestrial Networks

TLS: Transport Layer Security

UAS: Unmanned Aircraft Systems

UE: User Equipment

UI: User Interface

VPN: Virtual Private Network

1. Introduction

The federation of computing nodes represents an innovative strategy in the field of distributed computing, where autonomous systems converge to collaborate in the execution of complex tasks. This analysis explores in depth the purposes, strategies and requirements essential for the successful implementation of this federation.

The primary purpose of node federation is resource optimization, allowing independent computational entities to collaborate in a coordinated manner to execute operations that require combined capabilities. This strategy focuses on maximizing the efficiency, scalability, and agility of distributed systems.

To achieve an effective federation, it is crucial to establish clear and defined acceptance policies. These policies outline the criteria and standards for the admission of nodes to the federation, ensuring operational consistency, security and reliability of the federated ecosystem.

The requirements for the federation of nodes cover technical, security and performance aspects. From protocol compatibility to the ability to dynamically adapt to changing demands, these requirements ensure the smooth and secure integration of nodes into the federation.

Regarding tools for managing federated nodes, successful implementation requires robust solutions that facilitate centralized management, real-time monitoring, resource coordination, and efficient application of federation policies.

Together, this analysis dives into the fundamentals of compute node federation, exploring its objectives, implementation strategies, essential policies, fundamental requirements, and the key tools necessary for effective management in federated environments.

2. Proposed logical architecture

2.1. Definition of federation

Systems integration, known in the technological context as "federation", is a model in which several autonomous and heterogeneous entities come together to provide a unified service while maintaining their autonomy. This paradigm encompasses several elements, from standardised communication protocols and middleware to facilitate data exchange, to identity federation for authentication and authorisation, and policy management to implement federation agreements [1].

This concept becomes particularly relevant in the context of Non-Terrestrial Networks (NTN), which refer to communication networks that do not rely exclusively on terrestrial infrastructure, such as satellite or drone networks. Federation in this context enables integration and coordination between multiple NTNs, and even between NTNs and terrestrial networks, with the objective of providing resilient, scalable and high-quality communication services to users, regardless of their location or the type of network they are using. This is especially useful in situations where terrestrial networks are not feasible or efficient, such as in rural or remote areas, in maritime or aerial environments, or in emergency or disaster situations where terrestrial infrastructure may be damaged or inaccessible.

2.2. Types of federations

Federation, a critical concept in the Information Technology (IT) world, is a model that promotes seamless integration and collaboration between independent and heterogeneous systems. Its main appeal lies in facilitating interoperability and information exchange in an efficient and secure manner. While the term 'federation' may appear to be a single concept, in fact covers a variety of implementations and strategies, each with its own particular technical characteristics.

These variations in federation implementation result in different types of federation, each suitable for particular systems and processes. The choice of which type of federation to implement will often depend on the specific use case, the technical and business needs, and the particular advantages that each type of federation can offer. These can range from simplifying data access and improving interoperability to increasing network resilience and scalability.

In this exploration, we will look at some of these federation types, providing a clearer picture of their inherent complexity, specific advantages, and the circumstances under which they may be most appropriate. Some of them could be:

1. Identity Federation: Enables digital identities, roles, and access rights to be shared and recognised between autonomous systems, networks or domains [2] [3] . This is done by exchanging information about users between the organisations involved. Common protocols for identity federation include SAML, OAuth and OpenID [4]. These protocols enable the secure exchange of authentication and authorisation information [5].

2. Database federation: Enables the integration of multiple independent and possibly heterogeneous databases into a single virtual resource [6]. In a database federation, databases are typically independent and possibly heterogeneous, which are integrated into a single virtual resource [7]. This allows users to access data in a uniform manner, regardless of its location or format. In a federated database architecture, each database maintains its autonomy and queries and transactions can be performed on multiple databases as if they were one.
3. Cloud Computing Federation: Cloud computing federation is a model that enables integration and collaboration between cloud service providers. In this model, computing resources are shared among several organisations to achieve common goals. This allows organisations to leverage cloud resources as needed without the need to maintain their own cloud computing infrastructure [8] [9].
4. Machine Learning Federation: Enables joint training of machine learning models across multiple distributed databases while maintaining data privacy. In machine learning federation, multiple devices or servers participate in the training of a machine learning model maintaining the privacy of their data. Each device trains a machine learning model on its own data and then sends updates to the model to a central server. The central server aggregates the model updates and sends them back to the devices [10].
5. Content Federation: This federation refers to the collaboration between different content management systems. Here, content data from multiple sources are combined and made available through a unified interface. Content federation is especially useful when data is too large to be transferred between systems or when data must remain in its original system for security or compliance reasons. Digital library management systems, enterprise content management (ECM) systems, and geographic information systems (GIS) often employ this type of federation. [11]
6. Directory Services Federation: Directory services, such as Lightweight Directory Access Protocol (LDAP), are used by organisations to store and manage information about their IT resources, such as users, servers and networks. In a federation of directory services, multiple directory services can be linked together to appear as a single directory service to applications and users. This can simplify identity and access management in large, distributed organisations. [12]
7. Federation in Edge Computing: With the growing popularity of edge computing, federation can be used to coordinate and manage distributed computing resources at the edge of the network. This may involve, for example, load balancing between different edge nodes, or data sharing between different edge devices. [13]
8. Internet of Things (IoT) Federation: In the context of IoT, federation refers to the collaboration and sharing of data and services between different IoT systems. This may include data sharing between different IoT devices, or collaboration between different IoT networks or IoT service platforms [14] [15].

In a communication system, most of these types of federations are connected to bring security, capability, efficiency, and scalability to the whole process.

2.3. Background federation software

Software federation is an essential component in systems engineering, enabling interaction and interoperability between several independent computer systems. It emerged as a logical extension of distributed database systems, which are systems in which the database is stored in multiple physical locations but can be accessed and manipulated as if it were in a single location [16]. This functionality is achieved through an abstraction layer that sits between the user and the underlying database systems, providing a unified view of the data.

This concept was developed in the late 1980s, where distributed database systems evolved to form what are known as federated database systems. In this model, data is divided among several autonomous databases that communicate and synchronise with each other but are presented to the user as a single logical database. Communication and synchronisation are managed by a federation coordinator, which handles user requests and distributes them to the appropriate databases [7] [17] [18] [6].

With the rise of the Internet and the proliferation of data in the 1990s, the concept of federation expanded to encompass not only databases, but also web data and data from other disparate sources. This led to more advanced and complex software federation technologies that could handle a wide variety of data sources and data types. The foundations of software federation rely on database federation, identity federation and in short cloud federation to provide proper interoperability and privacy [16] .

Software federation is now an integral part of the architecture of many modern IT systems, such as cloud computing, microservices, business intelligence and big data systems. It enables data sharing and synchronisation across independent and heterogeneous systems and facilitates collaboration and data-driven decision-making across teams and organisations. All this is achieved while maintaining autonomy and local control of individual systems, ensuring data privacy and security [1].

2.4. Service networks

Service Networks are highly connected structures where services are delivered through multiple entities, which could be individuals, organisations, or software systems. The underlying infrastructure enables the interaction between these entities to deliver specific services efficiently and reliably. These networks are not limited to a specific domain; instead, they can span multiple domains such as telecommunications, finance, health, among others. This is where federation plays a crucial role, as it enables interoperability between these independent and heterogeneous systems. [19]

Going into more detail, a Service Network can be modelled through different levels of abstraction. For example, there may be an Infrastructure Layer, which includes the hardware and physical

resources, the Platform Layer, which handles the operating system and network technologies, and the Application Layer, which includes the specific applications and services offered to users.

The services in these networks may be distributed, meaning that they may reside on different systems in separate physical locations. This poses challenges for interoperability and data management. Service federation, through a federated architecture, enables the integration of these disparate services to provide a unified view to users. [1]

For example, consider a music streaming service that has users all over the world. The music could be stored on servers in different geographic locations. Without a federation, each user would have to access each server individually to get their music. However, with a federated architecture, the service can unify access to music for all users, regardless of their location.

Referring to NTNs, service networks can be vital to ensure interoperability and connectivity in NTNs. By using a federated architecture, NTNs can provide services in a heterogeneous network that includes terrestrial and non-terrestrial systems, enabling constant and seamless connectivity.

In short, service networks are highly connected structures that enable the delivery of services across multiple entities and domains. Federation plays a crucial role in these networks, enabling interoperability between independent and heterogeneous systems. NTNs can benefit from this federated architecture to ensure interoperability and connectivity in a heterogeneous network.

2.5. Federation architecture

The growing need for connectivity and access to real-time information and services, especially in the context of terrestrial and NTNs, has motivated the development of more complex and sophisticated network architectures. In this scenario, federation architecture presents itself as a crucial solution to facilitate interoperability and network efficiency between independent systems and platforms, such as satellites, drones, and high-altitude balloons.

Federation of Satellite Systems (FSS) is a paradigm that enables collaboration between different satellites to optimise the use of available resources. This approach is based on a federation architecture, which provides a framework for independent systems to work together more efficiently [20].

According to [21], Satellite Federations are in-space cloud-computing environments in which all types of space missions can join to contribute. Any type of space mission may benefit from the resources exchange: data-intensive missions such as Synthetic Aperture Radar (SAR), hyperspectral and high-resolution imaging missions, for instance, may enhance the platforms total data volume downloaded, and latency, through the federation.

This federation architecture is implemented in the context of NTNs, which are networks that use space nodes, such as satellites, to transport and route information. NTNs enable the connection of remote and rural areas and benefit from satellite federations to improve network utilisation and management flexibility.

FSS and NTNs interact through the Federation Deployment Control Protocol (FeDeCoP), which defines a set of guidelines for establishing a federation. This protocol allows a satellite to offer its unused resources as services to other satellites in the federation and establishes the rules for communication and negotiation between satellites to optimise resource utilisation [20]. The FSS, NTNs and federation architecture therefore form an integrated system for optimising satellite networks, allowing better use of available resources and performing tasks more efficiently through collaboration between satellites.

This architecture allows each platform to operate independently, while collaborating and sharing resources with the others to improve network coverage, capacity, and resilience. In short, the need for a federation architecture stems from the continuous evolution of our communication demands and the desire to optimise network performance and security in an increasingly interconnected world. The architecture would consist of several layers, which could include:

1. User Interface (UI) layer: this layer provides the interaction between users and the services offered by the federation. Interfaces can be of various types, from web applications to mobile applications, and should be designed with the user experience in mind, ensuring easy and efficient access to services.
2. Services layer: This layer consists of multiple services that can be offered by different entities within the federation. These services may include, for example, data communication services, video services, surveillance services, among others. Each of these services may be hosted in a different system (such as a satellite, a drone, etc.), but they must be able to interact and cooperate with each other through the use of standard APIs and protocols, such as HTTP, REST, SOAP, etc.
3. Federation Layer: This is the key layer of a federated architecture, in charge of managing the interactions between the different services and handling authentication and authorisation between systems. An important component of this layer is the "Federated Broker", which coordinates communication between the different federated entities, discovers services, negotiates service contracts, and monitors performance. The Federated Broker can also handle conflicts and security issues.
4. Data Layer: This layer consists of several databases that store the data used by the services. Each database may be in a different system, but all must be accessible through the federation system. This access can be achieved through a federated database system that allows users to query and manipulate data from multiple databases as if they were one. Data interoperability can be achieved using common data standards, data transformations, and metadata schemas.
5. Analysis Layer: This layer is responsible for collecting and analysing data on the performance of the federated system. It can collect data on service usage, network performance, resource utilisation, etc., and use this information to optimise system performance. This layer can use big data analytics and machine learning techniques to extract useful information from the collected data.

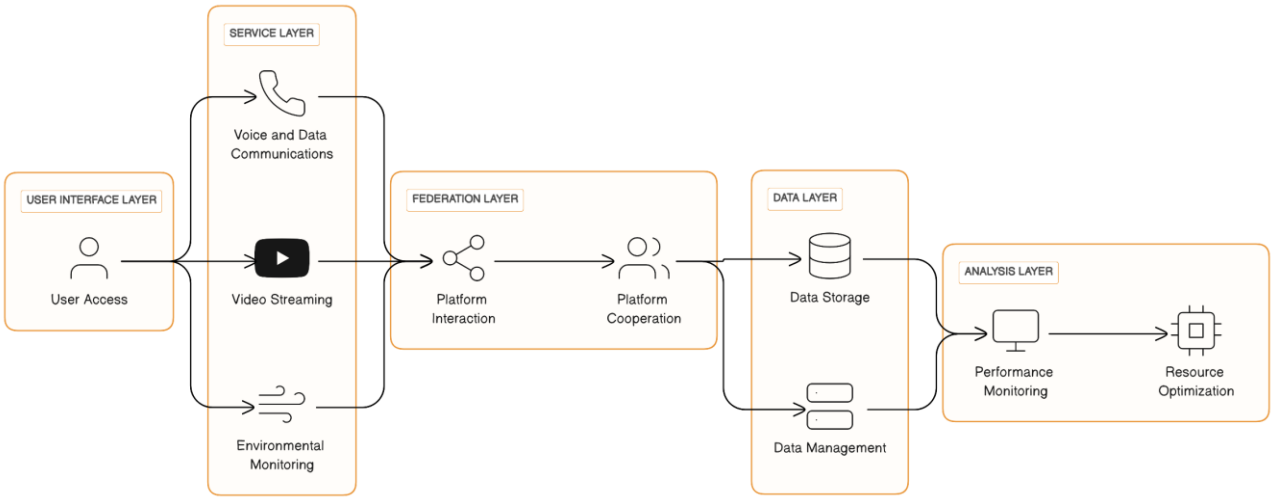


FIGURE 1 FEDERATION ARCHITECTURE

In this architecture, policy management, quality of service, metadata management and data security would be key aspects to consider. Policy management would determine how resources are shared and used in the federation, how user identities are managed, how service interruptions are handled, etc. Quality of service would be crucial to ensure that users receive a reliable and high-quality service, regardless of network conditions. Metadata management would allow the federation to efficiently organise and use the data it generates and stores. Data security would be of vital importance to protect users' privacy and ensure data integrity and confidentiality.

In addition to these layers, a federated architecture must consider aspects such as managing system heterogeneity (i.e. how to handle systems with different characteristics and capabilities), variable latency (i.e. how to handle differences in response time between systems), service continuity (i.e. how to ensure that services remain available even when some systems fail or are offline), and spectral efficiency (i.e. how to use the radio frequency spectrum efficiently) [1].

Finally, the architecture must be designed with a focus on security and privacy. This may involve the use of technologies such as encryption, role-based access control, and two-factor authentication to protect data and systems against unauthorised access. In addition, it may involve compliance with privacy laws and regulations to protect users' personal data.

3. Tools to facilitate the management of federated elements

3.1. System architecture

In this section, we provide a detailed description of the design of the NTN architecture. In particular, the implementation of different algorithms, which are detailed later. Different architectural options can be considered depending on the type of solutions used, which is also defined based on the capabilities of the payload. Section 1 describes the system architecture and available design options, while in Section 3.1.2 we discuss how the algorithms impact different models and metrics.

3.1.1. Architecture design options

The architecture of the NTN satellite system is affected by many design options, such as:

1. The type of satellite payload
2. The type of functional division when assuming payloads, i.e., which layers are implemented on board in the Distributed Unit, and which are implemented on the ground in the Centralized Unit
3. Network security, in terms of new federates and shared information.

Considering these needs, we can use the orchestration system approach. Service orchestration is the coordination and deployment of multiple services exposed as a single aggregate service. It is used to automate business processes by loosely coupling different services and applications, thus creating composite services [22].

Therefore, it is necessary to have an orchestration system that can manage the activities of the orchestrator, such as mounting machines, initializing containers, replacing failed containers, linking containers with each other, exposing services to machines outside the cluster, organize code deployments without affecting the availability of the service, among other services. Container orchestration does all these things through infrastructure automation. The great benefit of orchestration is to make the most of automated tasks and already defined processes. Orchestration allows you to manage very complex infrastructure in a simple way.

Since the orchestration challenge requires tools that are easy to install, configure, and maintain, Kubernetes was created to meet that need.

Kubernetes provides a container-centric management environment and orchestrates computer, networking, and storage infrastructure so user workloads don't have to. This offers the simplicity of Platforms as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS) and enables portability between infrastructure providers.

In summary, the benefits of using Kubernetes include:

- Agile application creation and deployment: Greater ease and efficiency when creating container images instead of virtual machines.

- Development, integration, and continuous deployment: Allows the container image to be built and deployed frequently and reliably, facilitating rollbacks since the image is immutable.
- Separation of duties between Dev and Ops: You can create container images at build time and not at deploy, decoupling the application from the infrastructure.
- Observability Not only the information and metrics of the operating system are presented, but also the health of the application and other signals.
- Consistency between development, testing and production environments: The application works the same on a laptop and in the cloud.
- Portability between clouds and distributions: Works on Ubuntu, RHEL, CoreOS, your physical datacenter, Google Kubernetes Engine, and everything else.
- Application-centric management: Raises the level of abstraction from the operating system and virtualized hardware to the application running on a system with logical resources.
- Distributed, elastic, loosely coupled microservices: Applications are separated into small, independent pieces that can be deployed and managed dynamically, rather than as a monolithic application that operates on a single large-capacity machine.
- Resource isolation: Makes application performance more predictable.
- Resource utilization: Allows greater efficiency and density.

Kubernetes architecture

Before explaining the examples of some of the platform proposals developed for Kubernetes management, we will generally explain the elements of its architecture:

- **Nodes:**

Kubernetes runs your workload by placing containers into Pods to run on Nodes. A node may be a virtual or physical machine, depending on the cluster. Each node is managed by the control plane and contains the services necessary to run Pods. Typically, you have several nodes in a cluster; in a learning or resource-limited environment, you might have only one node. The components on a node include the kubelet, a container runtime, and the kube-proxy.

Management: There are two main ways to add nodes to the API server:

1. The kubelet on a node is automatically registered in the control plane.
2. User manually adds a Node object.

After a Node object is created, or the kubelet on a node is automatically registered, the control plane checks whether the new Node object is valid. For example, if you try to create a node from the following JSON manifest:

```
{
  "kind": "Node",
  "apiVersion": "v1",
  "metadata": {
    "name": "10.240.79.157",
    "labels": {
      "name": "my-first-k8s-node"
    }
  }
}
```

FIGURE 2 JSON FOR CREATE NODE

- **Node-Master Communication:**

Kubernetes has a "hub-and-spoke" API pattern. All API usage from the nodes (or the pods they run) ends up on the API server. None of the other control plane components are designed to expose remote services. The API server is configured to listen for remote connections on a secure Hypertext Transfer Protocol Secure (HTTPS) port with one or more forms of client authentication enabled. One or more forms of authorization must be enabled, especially if anonymous requests or service account tokens are allowed.

Nodes must have the public root certificate for the cluster so that they can securely connect to the API server along with valid client credentials. A good approach is for the client credentials provided to the kubelet to be in the form of a client certificate. See kubelet Transport Layer Security (TLS) boot for automated provisioning of kubelet client certificates. Pods that want to connect to the API server can do so securely by leveraging a service account so that Kubernetes automatically injects the public root certificate and a valid bearer token into the pod when it is instantiated. The Kubernetes service (in the default namespace) is configured with a virtual IP address that is redirected (via kube-proxy) to the HTTPS endpoint on the API server.

- **Cloud Controller Manager Underlying Concepts:**

Cloud infrastructure technologies let you run Kubernetes on public, private, and hybrid clouds. Kubernetes believes in automated, API-driven infrastructure without tight coupling between components. The cloud-controller-manager is a Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API and separates out the components that interact with that cloud platform from components that only interact with your cluster.

By decoupling the interoperability logic between Kubernetes and the underlying cloud infrastructure, the cloud-controller-manager component enables cloud providers to release features at a different pace compared to the main Kubernetes project.

The cloud-controller-manager is structured using a plugin mechanism that allows different cloud providers to integrate their platforms with Kubernetes.

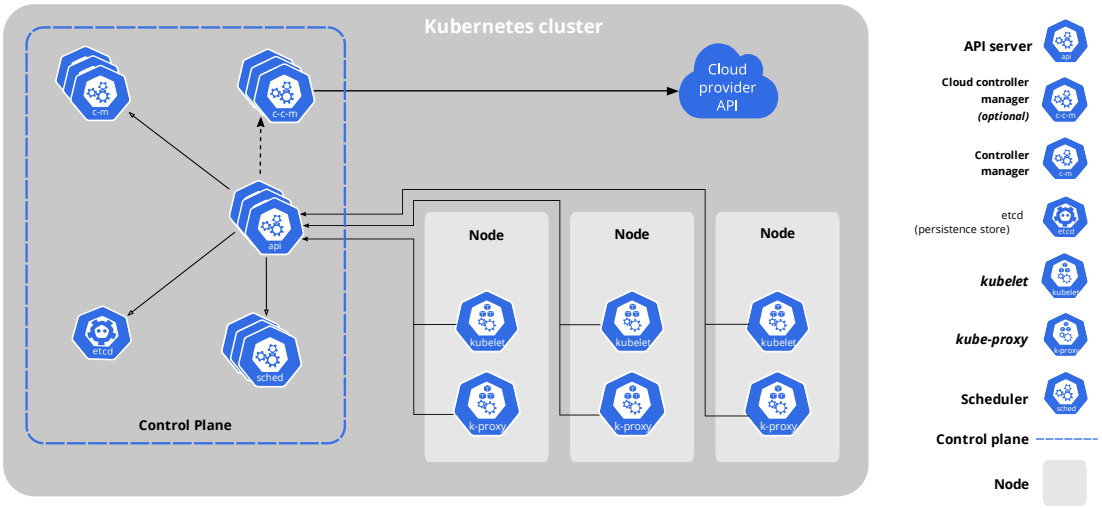


FIGURE 3 CLOUD CONTROLLER MANAGER DESIGN [23]

- **Cloud controller manager functions**

The controllers inside the cloud controller manager include:

1. Node controller:

The node controller is responsible for updating Node objects when new servers are created in your cloud infrastructure. The node controller obtains information about the hosts running inside your tenancy with the cloud provider.

2. Route controller:

The route controller is responsible for configuring routes in the cloud appropriately so that containers on different nodes in your Kubernetes cluster can communicate with each other. Depending on the cloud provider, the route controller might also allocate blocks of IP addresses for the Pod network.

3. Service controller:

Services integrate with cloud infrastructure components such as managed load balancers, IP addresses, network packet filtering, and target health checking. The service controller interacts with your cloud provider's APIs to set up load balancers and other infrastructure components when you declare a Service resource that requires them.

- **Container Runtime Interface (CRI)**

The CRI is a plug-in interface that allows kubelet to use a wide variety of container runtimes, without needing to recompile cluster components. Need a container runtime running on every node in your cluster, so that kubelet can start Pods and their containers.

The CRI is the main protocol for communication between kubelet and Container Runtime. The Kubernetes CRI defines the primary gRPC protocol for communication between kubelet node components and the runtime container.

3.1.2. Approaches to the challenge of multi-cluster fleet management

There are currently two Cloud Native Computing Foundation (CNCF) projects that provide modern answers to the problem of managing multiple kubernetes, Karmada and Rancher, they are here to take on the challenge of managing fleets of clusters across the hybrid and multi-cloud landscape.

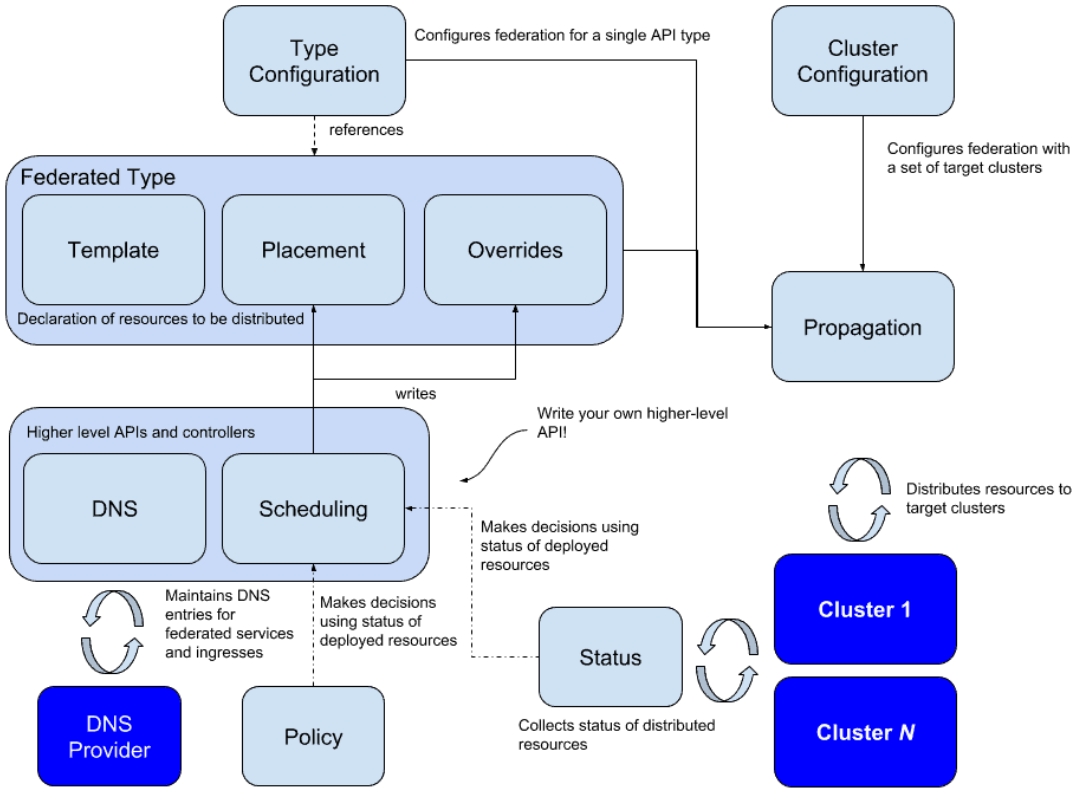


FIGURE 4 CONFIGURATION OF MULTIPLE KUBERNETES CLUSTERS FROM A SINGLE SET OF APIS IN A HOSTING CLUSTER [24]

Karmada: Open, Multi-Cloud, Multi-Cluster Kubernetes Orchestration

Karmada (Kubernetes Armada) is a Kubernetes management system that allows you to run cloud-native applications across multiple Kubernetes clusters and clouds, without application changes. By using native Kubernetes APIs and providing advanced programming capabilities, Karmada enables truly open, multi-cloud Kubernetes. Karmada aims to provide turnkey automation for multi-cluster application management in multi-cloud and hybrid cloud scenarios, with key features such as centralized multi-cloud management, high availability, failover, and traffic scheduling [24].

Karmada Features:

- Resource template: Karmada uses the native Kubernetes API definition for its federated resource template, to facilitate integration with existing tools that already adopt Kubernetes.
- Propagation Policy: Karmada offers a separate propagation (location) policy API to define the scheduling and diffusion requirements of multiple clusters.
- Override Policy: Karmada provides a separate Override Policy API to specialize the automation of cluster-relevant configuration.

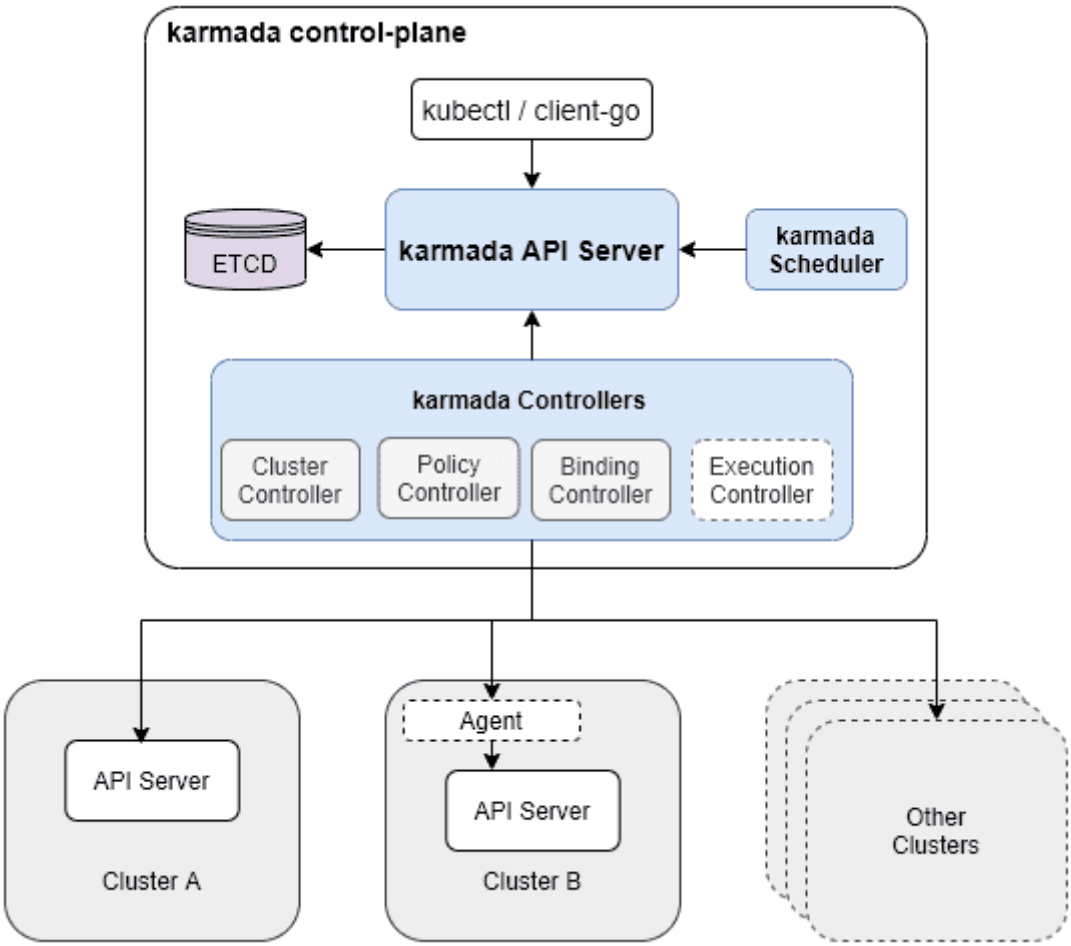


FIGURE 5 PRELIMINARY ARCHITECTURE FEDERATION [24]

4. Proposal of a strategy for the federation of computational nodes

In today's digital era, the volume of data generated and processed has grown exponentially, posing significant challenges in terms of data storage, processing, and security. To address these challenges, it has become increasingly common to rely on distributed systems, such as the federation of computational nodes. In a federation, multiple autonomous systems come together to share resources and capabilities, creating a collaborative network that offers significant benefits in terms of scalability, efficiency and resilience.

However, implementing and managing a federation of computational nodes is a complex task that requires a careful and well-designed strategy. This strategy must address multiple aspects, from system integration and node coordination to security and privacy management. It must also be flexible and adaptable, able to adjust to changes and evolve as needs and technologies change.

In the following proposal, a comprehensive strategy for the federation of compute nodes will be presented, which takes these aspects into account and provides a roadmap for the successful implementation and management of a federation. By implementing this strategy, organisations will be able to maximise the benefits of federation while minimising the risks and overcoming the challenges.

4.1. Node Identification

Node identification is crucial in the federation of computer systems. It refers to the cataloguing of each network component (computers, servers, mobile devices, routers, etc.) according to their capabilities and roles. The process includes an inventory of the nodes and an assessment of their hardware and software capabilities. This assessment may include factors such as processing power (measured in terms of Central Processing Unit (CPU) speed and number of cores), memory capacity, storage capacity and network capabilities (such as network speed and latency).

In addition, it is important to consider any specialized hardware or software that may be present on a node, as this may affect its suitability for certain tasks. For example, a node equipped with a high-performance GPU may be particularly suited to tasks that require intensive graphics processing or parallelized computation.

The specific roles of the nodes, such as file servers, databases, or compute nodes, are also identified. Finally, each node is assigned a unique identifier for its location and communication in the federated network. This process enables efficient resource management and optimizes network performance.

4.2. Standardization and Federation protocol

The second essential step in the federation of computing nodes is standardization. It is a process that involves the definition and adoption of common rules and procedures to ensure that all nodes in the federation can interact and communicate efficiently and smoothly.

Key elements of standardization include communication protocols which define how data exchanges are performed between the nodes of the federation (such as TCP/IP, UDP, HTTP, FTP, gRPC, REST), standardized Application Programming Interfaces (APIs) for requesting and providing services, standardized data formats such as JSON or XML which ensure that all nodes can interpret and use the data they receive, standardized management systems and security standards to protect the federation against threats and to guarantee confidentiality, integrity and security, standardized data formats such as JSON or XML that ensure that all nodes can interpret and use the data they receive, standardized management systems and security standards to protect the federation against threats and ensure confidentiality, integrity and availability of data and services. This may include the use of encryption technologies, authentication and authorization, access control, monitoring and intrusion detection, and security incident response.

On the other hand, the definition of the federation protocol in the federation of computational nodes dictates the rules for the interaction between nodes in a federated network. It ensures the interoperability of nodes through standardized communication standards and formats, protects data security and privacy through authentication and authorization processes, and coordinates resource management, data synchronization and network fault management.

In general, standardization is a process that requires a careful balance between the need for consistency and flexibility to adapt to changing needs and new technologies. The adoption of open and widely used standards can help facilitate this balance and ensure the long-term interoperability and flexibility of the federation.

4.3. Middleware integration

The integration of middleware in the federation of computational nodes is essential to enable interoperability between different nodes. In [25] there are a big explanation of many different types of middleware depending on the needs and some commercial and cooperative examples, such as:

- Event-based. In event-based middleware, components, applications, and all the other participants interact through events. Each event has a type, as well as a set of typed parameters whose specific values describe the specific change to the producer's state. For example: MiSense, PSWare, TinnyDDS, Hermes, Hydra among others. Each with this own kind of resource discovery, resource, data, event and code management.
- Service-oriented, which has been traditionally used in corporate IT systems. Some examples such as SenseWrap, MUSIC or TinySOA.

- VM-based design provides programming support for a safe execution environment for user applications by virtualizing the infrastructure. Maté, TinyReef and TinyVM.
- Agent-based. Here applications are divided into modular programs to facilitate injection and distribution through the network using mobile agents. Smart messages, Ubiware, Impala, ActorNet and Agilla.
- Tuple-spaces. tuple space middleware facilitates communication and data exchange in a federation of computational nodes, enabling efficient cooperation between nodes and optimal utilization of available resources. LIME, TinyLIME, TS-Mid and TeenyLIME
- Database-oriented. In database-oriented middleware, a sensor network is viewed as a virtual relational database system. An application can query the database using an SQL-like query language, which enables the formulation of complex queries. SINA, COUGAR, IrisNet and TinyDB.
- Application-specific. It is based on implementing an architecture that fine-tune the network or infrastructure based on application requirements. AutoSeC, MiLAN nd TinyCubus.

Middleware provides an abstraction of hardware and software differences, facilitates communication between applications, handles resource management, and provides security and fault management measures. Implementing middleware requires careful selection and configuration of software and may require modification of existing applications.

To integrate middleware into a federation architecture, it is necessary to select a set of middleware that provides the necessary services, configure the middleware on each node to communicate with the middleware on other nodes, and modify the applications to use the middleware services. This process can be complex and require considerable expertise in distributed systems programming but can be facilitated using middleware integration tools and middleware standards such as CORBA, Java RMI or gRPC [26].

4.4. Data management

- Data management is an essential component of any computational node federation strategy to ensure data efficiency, security, and usability. It encompasses several functions and tasks to manage, organize, catalogue, and deliver data across the node federation [6] [27]. In the context of the federation of computational nodes, data management involves several key components:
- Data Modelling: Defines how data is organized, related, manipulated, and stored. It is essential for the understanding and interoperability of data between different nodes and applications.
- Data Integration: This is the task of merging data from different sources and providing users with a unified view. In a federated environment, data may reside in different nodes with

different formats and structures. Data integration techniques, such as schema-based integration or query-based integration, are used to unify this disparate data.

- Data Quality: This is an important aspect of ensuring that data is accurate, consistent, and useful to users. It includes tasks such as data cleansing, deduplication, and validation [28].
- Data Security: This refers to measures taken to protect data from unauthorized access or malicious alteration. It includes access control, encryption, auditing and other security measures [27].
- Data Processing: This involves the manipulation of data to serve the needs of users. It may include query processing, data transformation, data analysis, among others.
- Metadata: Metadata describes the characteristics of the data, such as its origin, structure, owner, access and usage policies. Metadata management is essential to help users discover, understand and use data effectively.
- Data Architecture: This is the way in which data is structured and organized in the federation. This may involve building a federated data catalogue, which provides a unified view of all data available in the federation, regardless of its location or format.
- Data Policies: These are the rules that govern the use and manipulation of data. Data policies can cover aspects such as privacy, security, data quality, data retention, compliance with regulations, etc.

In short, data management in a federated environment requires careful planning and execution and is essential to enable effective and secure operation of the federation. Without effective data management, users may find it difficult to access, understand or trust the data, which can limit the usefulness and value of the federation.

4.5. Security and privacy

Security and privacy are critical aspects in the federation of computational nodes to ensure trust in the system, safeguard data and maintain regulatory compliance. A robust and comprehensive approach to security and privacy will not only protect valuable assets and data, but also facilitate the adoption and use of the federation by users and organizations [5]. It covers issues such as:

- Authorisation and authentication, in a federated environment, authentication is the first line of defense and refers to the process of verifying the identity of a user, system or device.
- Encryption is an essential technique to protect data during transmission and at rest. Strong encryption protocols ensure that data can only be read by those who possess the correct decryption key.
- Data integrity refers to the accuracy and consistency of data throughout its lifecycle. Techniques such as cyclic redundancy checks (CRC) and digital signatures can be used to ensure integrity.

- Federated Identity Management (FIM) allows identities to move across organizational boundaries and security domains. FIM standards, such as SAML, OpenID, OAuth, and WS-Federation, enable cross-organizational authentication and authorization [2] [29] [4].
- Resilience federated systems must be resilient to attacks and be able to recover quickly in case of security incidents. This implies the implementation of redundancy and backup measures, as well as disaster recovery and incident response plans.
- Regulatory Compliance: Federated systems must comply with applicable laws and regulations, such as GDPR or CCPA, which may impose specific requirements in terms of data security and privacy [30].
- Auditing and Event Logging: Logging and monitoring system activities helps to detect unusual or malicious behavior. Auditing solutions can track actions, such as data access, configuration changes, etc., and can be a valuable tool for detecting security incidents.

4.6. Monitoring and maintenance

Monitoring and maintenance in a federation of compute nodes involves several areas, such as monitoring system health, which can include metrics such as CPU utilization, memory utilization, network utilization, storage space, among others. This information is critical for identifying and troubleshooting performance issues, logging and auditing events that occur on nodes and in the federation in general should be logged for further analysis and auditing. This can help identify patterns and trends, which in turn can be used to optimize performance and resolve problems. implementing alerts and notifications Based on monitoring data and logs to act quickly in case of failure. Managing updates and patches Keeping all nodes in the federation up to date with the latest software releases and security patches is critical to ensuring security and performance, disaster recovery and data backup, it is important to have disaster recovery and data backup strategies in place. This may involve replicating data across multiple nodes, creating regular backups and implementing disaster recovery procedures, configuration management, and preventive maintenance involving scheduled checks and repairs to prevent failures and performance degradations [7] [31] [32] [33].

4.7. Adaptability

Adaptability in a federation of compute nodes encompasses several areas, including Dynamic Scalability and Load Balancing that interrelate and work together in a federated system. Dynamic Scalability takes care of adjusting the capacity of the system to the level of demand, while Load Balancing ensures that workloads and resources are optimally distributed among the available nodes. Together, these mechanisms allow the federation to operate efficiently and resiliently, accommodating changes and fluctuations in demand and resource supply Fault Tolerance The federation must be resilient to failures at individual nodes or in the network. It should be able to automatically detect failures, reallocate affected tasks and resources to other nodes, and recover from failures without significant service interruption. While Evolvability allows the system to keep up

with new technologies and changing requirements, customization ensures that each node can function in the best possible way within that system. Together, these elements allow the federation to be adaptable, flexible and efficient, able to respond to changes and challenges in the technological environment. These aspects are vital to handle the inclusion or removal of nodes, changes in workloads, and fluctuations in usage patterns dynamically and efficiently [34].

5. Federation Strategy

Strategy Proposal for Node Federation, based on the previously proposed architecture, focuses on leveraging the maximum capacity and flexibility of the Kubernetes cluster. In this regard, admission, control, and priority policies are established to ensure efficient and secure management of distributed resources. By implementing orchestration tools like Kubernetes, nodes can be joined cohesively, allowing for a balanced distribution of workloads and high availability of applications. Furthermore, maintaining federation security at the federated cluster level is achieved through the implementation of a virtual private network and robust security policies, along with authentication and authorization mechanisms. This strategy aims to optimize the management and performance of the Kubernetes cluster, ensuring scalability, availability, and security of applications deployed in the container environment.

5.1. Elements to Federate

The proposed federation strategy focuses on federating nodes within the Kubernetes cluster. Nodes are the virtual or physical machines that constitute the underlying infrastructure of the cluster and upon which containers running applications are executed. Federating nodes involves joining these resources cohesively to maximize their computing and storage capacity, allowing for unified and efficient cluster management.

Federating nodes aims to achieve the following objectives:

1. **Efficient Resource Utilization:** Federating nodes enables the balanced distribution of workloads among different cluster nodes, maximizing the utilization of available resources and avoiding overload on individual nodes.
2. **High Availability and Fault Tolerance:** Federating nodes allows for the configuration of replication and redundancy policies to ensure continuous availability of applications even in the event of node failure. This ensures service continuity and minimizes downtime.
3. **Unified Management:** Federating nodes enables managing and administering the Kubernetes cluster as a single cohesive entity. This simplifies administration, updating, and monitoring operations, reducing complexity and operational costs.
4. **Flexibility and Portability:** Federating nodes enables the creation of a flexible and portable infrastructure environment, where applications can consistently run on any cluster node, regardless of its physical location or specific characteristics.

5.2. Purpose of Federation

Federating nodes or machines under the Kubernetes architecture is carried out to efficiently and scalably lift services and computations according to different use cases. This capability allows for initiating and managing specific services according to the needs of each application or set of applications deployed in the cluster. By federating nodes, computing, storage, and network resources can be dynamically and adaptively allocated, ensuring optimal performance and high availability for running services.

For example, for services requiring intensive processing capacity, nodes with higher CPU and memory resources can be allocated. For applications needing high storage performance, nodes with high-performance storage disks can be allocated.

5.3. Policies within the Federation Strategy

Several policies are established to guide the federation process and ensure a consistent and secure environment:

Admission Policies

Admission policies are fundamental to ensuring the integrity, security, and consistency of the federated environment. These policies address specific aspects affecting the adherence of new nodes to the cluster:

- **Installation of Monitoring Tools:** It is mandatory for nodes wishing to join the federation to install designated monitoring tools. These tools provide visibility and performance tracking of the cluster, which is crucial for early detection of issues and informed decision-making regarding resource management and performance optimization.
- **Implementation of a Virtual Private Network (VPN):** All nodes must configure and use a VPN to establish a secure connection to the cluster's private network. The VPN ensures encrypted communication between federated nodes and the cluster network, thus protecting sensitive data and ensuring the confidentiality and integrity of transmitted information.
- **Acceptance of Communication, Security, and Protocol Policies:** Nodes must explicitly accept the established communication, security, and protocol policies for federation. This includes communication policies regulating interaction between nodes and applications, security policies specifying security standards and practices to follow, and protocols defining communication methods and data transmission.
- **Configuration of Network Parameters:** Nodes must correctly configure network parameters, such as IP addresses, subnets, and routing, to ensure proper connectivity within the federated cluster. This ensures network consistency and integrity, avoiding IP address conflicts and routing issues that could affect communication between nodes and applications.

Priority Policies

Within the federation strategy, priority policies are established to efficiently manage resources and ensure optimal operation of applications in critical situations. These policies are expanded to address different aspects related to resource prioritization and response to emergencies:

- **Installation of High-Priority VPN:** It is established that VPN installation has a high priority for certain critical or security-sensitive applications. This ensures that applications handling sensitive data or requiring a secure connection have priority access to a VPN to protect the confidentiality and integrity of transmitted information.
- **Allocation of Resources in Emergency Situations:** A priority policy is defined to allocate additional resources to applications in emergency situations. For example, in cases of unexpected demand spikes or critical system failures, prioritizing the allocation of additional resources, such as CPU, memory, or storage, ensures service continuity and minimizes impact on end users.
- **Incident Management:** A priority protocol is established for incident management, defining the sequence of actions to be taken in case of detecting issues or vulnerabilities in applications or the cluster. This includes allocating resources and resolving issues on a priority basis to minimize downtime and restore functionality of affected applications as soon as possible.
- **Prioritization of Critical Tasks:** A priority policy is defined to identify and prioritize critical tasks requiring immediate attention. This includes addressing security issues, applying security patches, and mitigating vulnerabilities to protect the cluster and applications against potential threats and attacks.

Control Policies

Control policies aim to ensure the security, integrity, and compliance of established standards in the federated environment. These policies are expanded to address various aspects related to application security and cluster management:

- **Application Security:** Control policies are established to ensure the security of applications deployed in the cluster. This includes implementing security measures across different layers, such as user and application authentication and authorization, data encryption at rest and in transit, and role-based access control policies to limit user and application privileges.
- **Verification of Security Policy Compliance:** A continuous verification process is defined to ensure compliance with established security policies in the cluster. This includes conducting periodic audits, actively monitoring system activity, and analyzing logs to detect and mitigate potential vulnerabilities and security risks.
- **Identity and Access Management:** Control policies are established to properly manage identities and access in the cluster. This includes implementing user and group management practices, multi-factor authentication, and applying granular access policies to ensure that only authorized users can access resources and sensitive data.

- **Updates and Security Patches:** A process for managing updates and security patches is defined to ensure that the cluster and applications are protected against the latest threats and vulnerabilities. This includes timely application of security patches, conducting compatibility testing, and monitoring potential impacts on system performance and stability.

5.4. Security

It is necessary to implement security measures both at the federated cluster level and at the application level to protect data and ensure the integrity and confidentiality of communications. The following security aspects necessary at each level are detailed below:

At the federated cluster level (globally)

- VPN: Allows for creating a secure communication tunnel between federated nodes and the cluster infrastructure. This ensures that all communications between nodes, as well as internal cluster network traffic, are encrypted and protected against unauthorized access from external networks. The VPN establishes an additional layer of security by hiding transmitted information over the public network and ensuring that only authorized nodes within the cluster can access and communicate with each other securely.
- Protocol Types: Using secure communication protocols throughout the federated cluster environment ensures authentication, encryption, and integrity of communications. For example, the Transport Layer Security (TLS) protocol secures connections between different cluster components, such as the API server, nodes, and deployed services. Using TLS ensures that communications are encrypted and protected against possible eavesdropping and data manipulation attacks. Additionally, by implementing role-based access control (RBAC) policies, access permissions of users and applications within the cluster can be controlled and managed, ensuring that they only have access to authorized resources and data according to their assigned roles and privileges.
- Network Security: Implementing additional security measures at the network level helps protect incoming and outgoing traffic of the cluster. This includes using firewalls and access control lists to restrict and filter network traffic, as well as network segmentation to isolate and protect different components and workloads within the cluster.

At the application level (locally):

- Transport Security: Implementing security measures in data transport helps protect communications between applications and services deployed in the cluster. This includes using secure protocols, such as HTTPS, to ensure that communications are encrypted and protected against unauthorized access and possible eavesdropping and data manipulation attacks. Additionally, if network services are configured to use secure connections and valid SSL/TLS certificates, the authenticity and integrity of communications are ensured.
- Application Security: Application-level security practices help protect services against potential vulnerabilities and security risks. This includes input data validation to prevent code

injection attacks, such as SQL injection and XSS (Cross-Site Scripting), as well as implementing proper access controls to protect sensitive resources and data from unauthorized access.

- Audit and Monitoring: Audit and monitoring systems help detect and respond efficiently to potential threats and malicious activities in applications deployed in the cluster. This includes configuring audit logs to record important events and activities, as well as implementing real-time monitoring systems to detect anomalies and alert about potential intrusions and security breaches. Additionally, incident response procedures can be established to investigate and mitigate potential attacks and security breaches in a timely and efficient manner.

5.5. Communications within the Federation

Within the federation strategy, it is crucial to establish effective methods of communication for both application deployment and monitoring, as well as to facilitate interaction among federated nodes. The communication approaches used are detailed below:

Communication for Deployment and Monitoring

- **Automated Application Deployment:** Configuration management and deployment tools, developed by the team, are utilized to automate the application deployment process on the federated nodes of the cluster. This allows defining and managing application packages (called "charts") containing all the necessary resources to deploy an application on Kubernetes, including containers, services, configurations, and dependencies. This will facilitate consistent and reproducible deployment of applications in the cluster, reducing the possibility of errors and improving operational efficiency.
- **Centralized Configuration Management (under development):** A centralized configuration repository is established to store and manage configuration files for applications deployed in the cluster. Configuration management tools, developed by the team, are used to maintain application configuration consistently and versioned across the Kubernetes environment. This will ensure that all instances of applications deployed on federated nodes are configured consistently and can be easily audited and reverted if needed.
- **Real-Time Monitoring:** Implementation of real-time monitoring systems is proposed for future work to collect metrics and monitor the status of nodes and applications deployed in the cluster; such as CPU and memory usage, network latency, and application performance. This will provide complete visibility into the operational status of the cluster and allow for the identification and resolution of issues before they affect service availability and performance.

6. Proof of concept that exemplifies the proposed strategy

A Proof of Concept (PoC) is a demonstration or small-scale project that serves to validate and showcase the feasibility of a proposed strategy or concept. It is typically used to test and verify whether an idea or approach will work in practice before committing to full-scale implementation. PoC allows stakeholders to see firsthand how the proposed strategy would function in a real-world scenario and assess its potential for success. It helps to reduce risks and uncertainties associated with a new strategy or technology and informs decision-making about whether to proceed with a larger-scale project.

The following figure shows the proposed of PoC to be carried out.

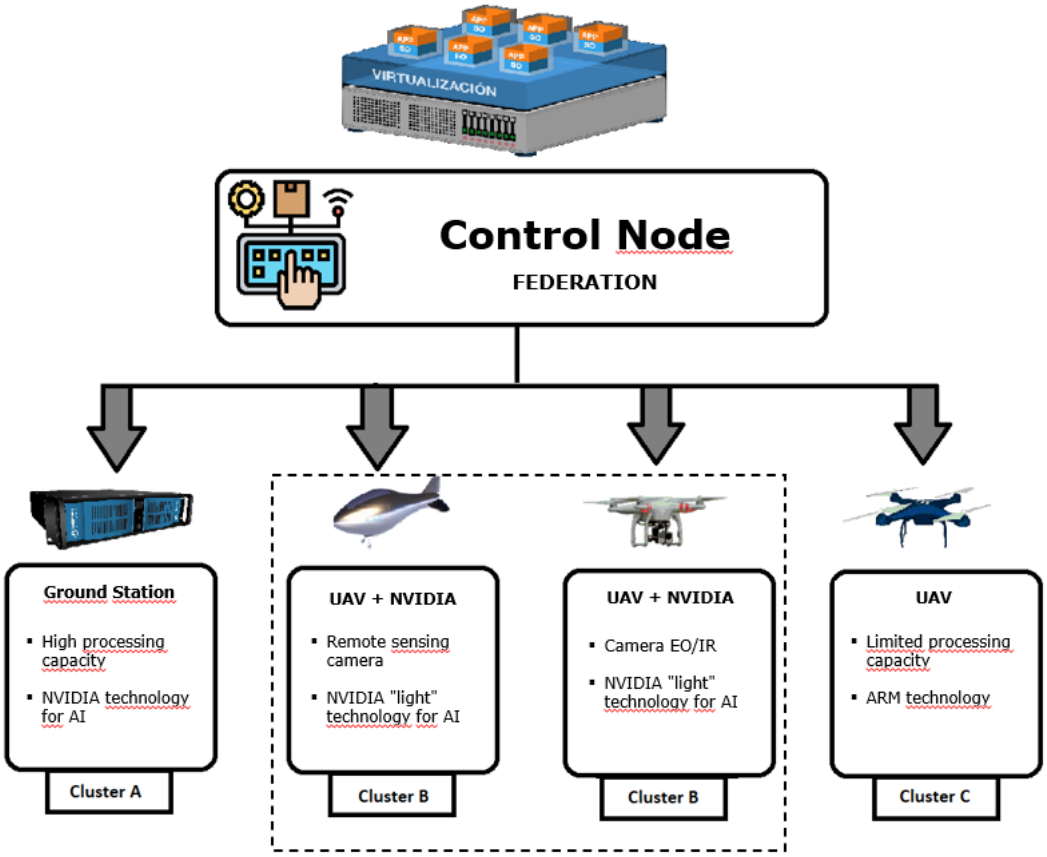


FIGURE 6 PROPOSAL POC

Cluster Federation

In this process, the necessary steps to federate a cluster on a specific platform are detailed. Federating a cluster involves integrating a local cluster into a broader network, allowing for sharing resources and capabilities with other clusters within the same federation. To carry out this process efficiently, a series of steps are required, including user validation, policy configuration, cluster resource specification, and software requirements compliance.

1. **User VPN Request** (as shown in the next figure)

- Request a user on the platform (to be completed in the security PoC).
- It is verified whether the user is authorized on the platform.
- If the user is not authorized:
  - The user is requested to provide their name.
  - The username and VPN credentials are received and logged.

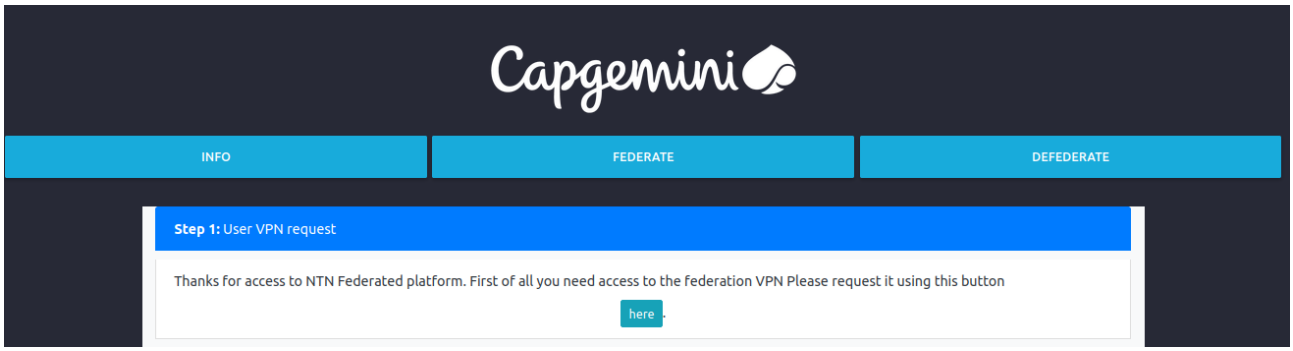


FIGURE 7 USER VALIDATION

2. **Software Requirements** (as shown in the following figure)
   - The necessary software is requested to be installed on the system:
     - The presence of a Kubernetes cluster is confirmed, following the provided instructions.
     - It is ensured that karmadactl is correctly installed as per the instructions.
     - The necessary software for VPN connectivity is verified to be correctly configured and operational.

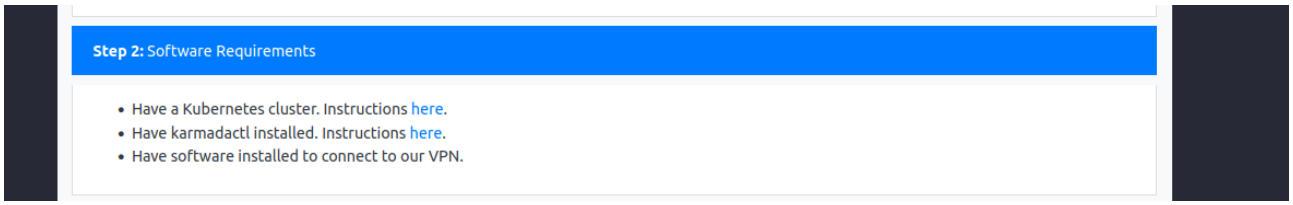


FIGURE 8 SOFTWARE REQUIREMENTS

3. **Completing the Policy Form** (as shown in the next figure)
   - The required details are entered into the policy form:
   - The cluster owner is specified.
   - A list of restricted owners obtained from the database is displayed.
   - The security level for services is chosen, opting between HTTP or HTTPS.
   - The priority level for available services is selected, with options such as "Emergency Only," "Medium and Emergency," or "All Services."
   - The maximum allowed size for services in terms of volume and memory is set.

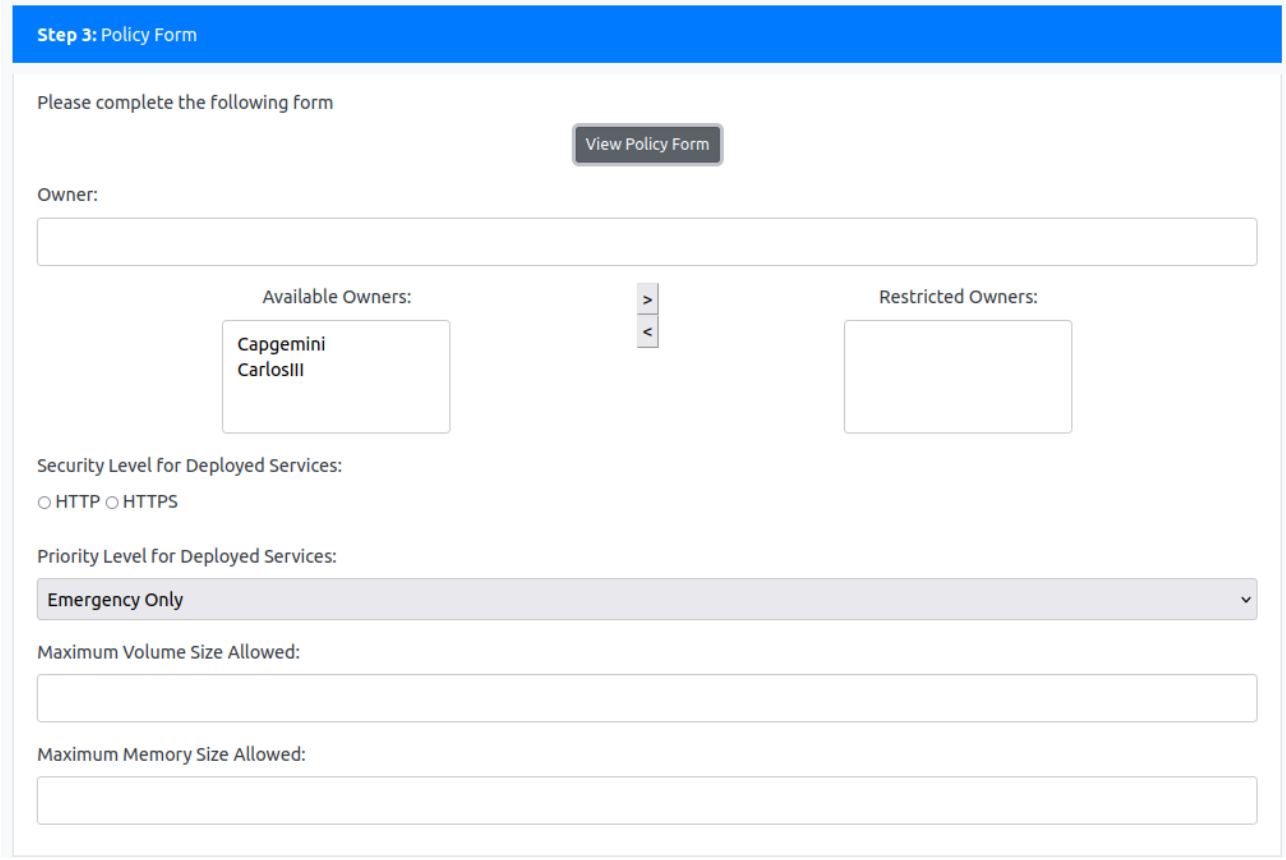


FIGURE 9 POLICY FORM

4. **Completing the Resource Form** (as shown in the next figure)
    - The resources that the cluster will share are detailed:
        - The cluster architecture is chosen from options such as arm64, x86_64, arm32, or arm_hf.
        - The number of Central Processing Units (CPUs) available is indicated.
        - The amount of available RAM memory is specified, expressed in megabytes (Mb), gigabytes (Gb), terabytes (Tb), or petabytes (Pb).
        - The size of the hard disk drive (HDD) volume is established, also in Mb, Gb, Tb, or Pb.
        - It is determined whether the cluster will have a GPU, and if so, the type of GPU is detailed, including brand, model, capacity, and features.
        - The availability of peripherals is indicated, and if applicable, they are specified, adding brand, model, and characteristics.

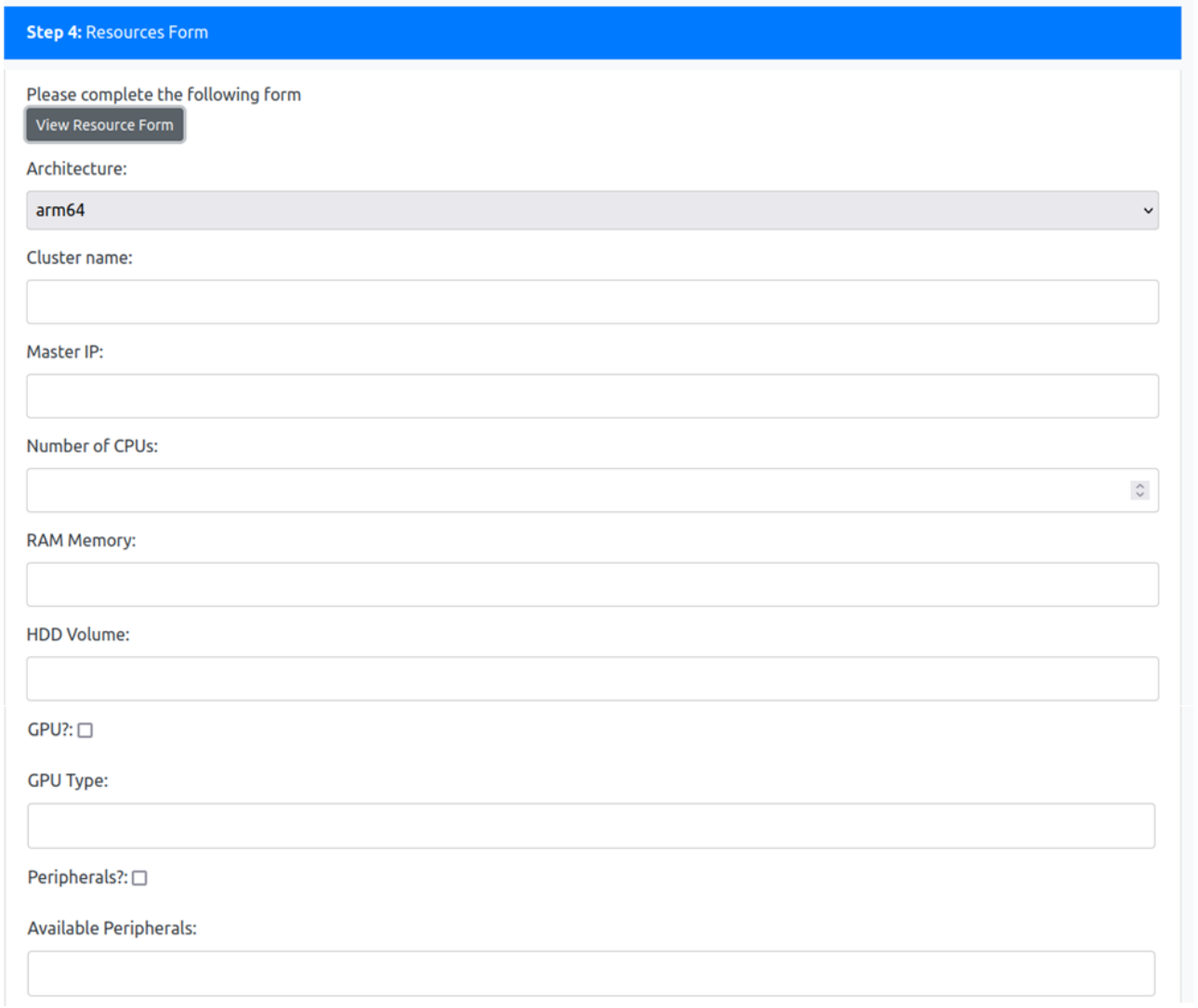
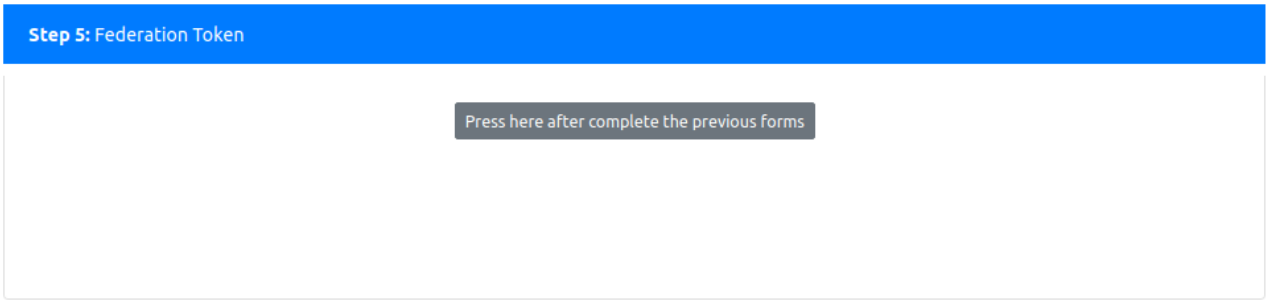


FIGURE 10 RESOURCES FORM

5. **Adding the Cluster to the Federation** (as shown in the next figures)
    - The process of adding the cluster to the federation is executed:
        - A join declaration is made along with a token, which is a unique authorization code.
        - The user is prompted to provide the join phrase with the included token to complete the federation process.



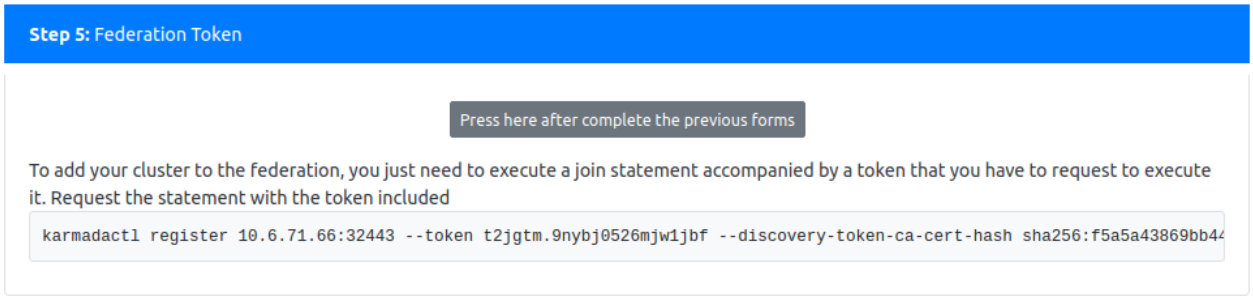


FIGURE 11 VALIDATION TOKEN

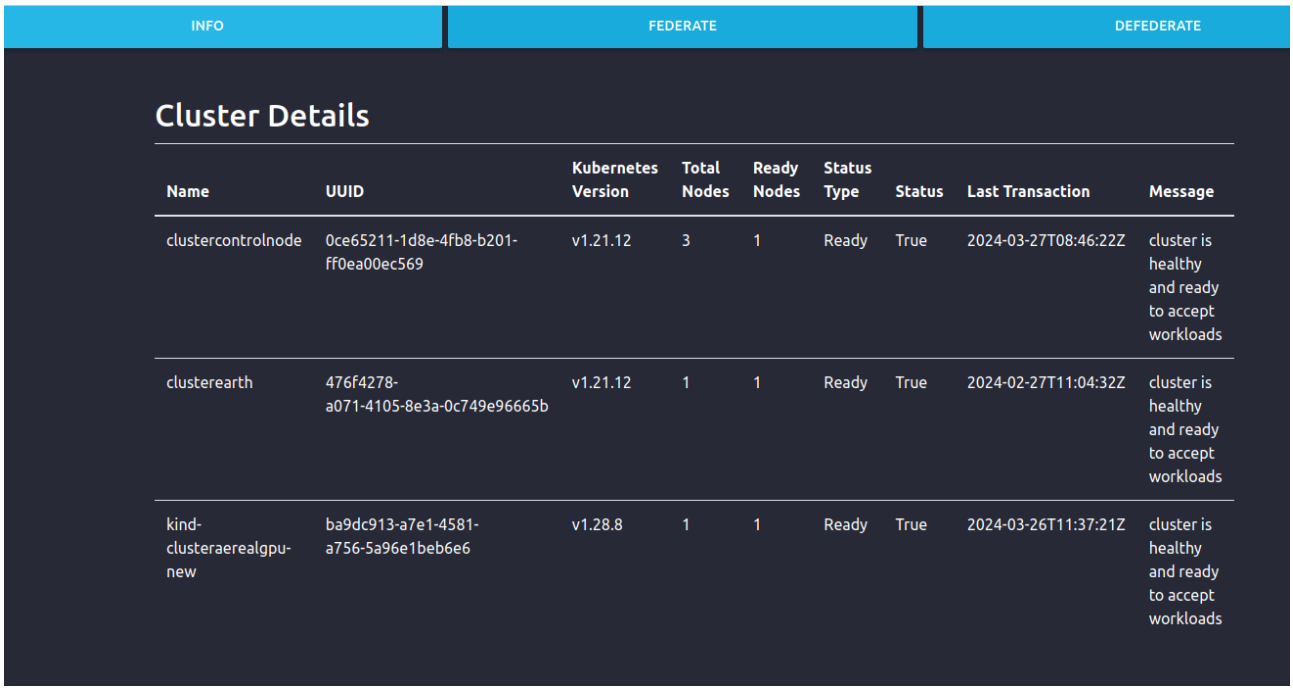


FIGURE 12 CLUSTER DETAILS

6.1. Analysis of Practical Experience

From a theoretical perspective:

The main challenge lies in designing federation policies, as several crucial factors must be considered to ensure the proper federation of nodes and the formulation of an effective processing distribution strategy within the network. These policies must guarantee data coherence and consistency among federated nodes, facilitating efficient communication and precise synchronization between them.

Furthermore, it is essential to design load balance and processing distribution policies that optimize resource utilization, minimize latency, and maximize the performance of deployed services. Security policies must also be robust and unified, ensuring consistent and secure access control and identity management across the entire federated network.

From a practical perspective:

Karmada offers significant advantages in terms of management and scalability for very complex deployments, but its disadvantages related to complexity, learning curve, and lack of support and documentation can be major obstacles. For small or medium environments, these disadvantages may outweigh the benefits, while in large infrastructures, Karmada's potential may justify the additional effort required for its implementation and management.

Advantages

- Potential in Complex Deployments: Karmada has great potential for managing very complex deployments with many clusters, especially in environments running thousands of replicas. Notably, Karmada proposes a policy and resource propagation system that facilitates service-level scaling, particularly when there is a high volume of requests.

  In summary, for very large infrastructures, Karmada can simplify the management and organization of distributed resources, allowing for better scalability and operational efficiency.

- Automated Cluster Organization: Karmada's ability to delegate organization to an automated system is beneficial in very large and relatively heterogeneous environments, as it requires the existence of a Kubernetes system for each cluster.

  This reduces administrative burden and the risk of human error in managing multiple clusters.

Disadvantages

- High Conceptual and Implementation Complexity: Karmada is complex both conceptually and in implementation, making it difficult to adopt in smaller or less sophisticated environments.

  It requires a team with a high level of knowledge and experience, which can increase training costs and implementation time.

- Slow Learning Curve: The learning curve for Karmada is very slow due to its complexity and the lack of accessible learning resources.

  Impact: teams by taking a long time to become productive, delaying the expected benefits of its implementation.

- Limited Documentation and Lack of Community Support: Karmada's novelty means there are few people working with it and few forums where help can be found, forcing users to rely on official documentation.

  This makes it difficult to solve problems and nearly impossible to get support beyond community forums, slowing down implementation and problem resolution.

- Version Incompatibilities: New versions of Karmada are not always compatible with each other, and several bugs have arisen during its development.

- Uninformative Logs: Karmada's logs do not contain much information, making troubleshooting and system monitoring difficult. Administrators may have trouble diagnosing and solving problems effectively.
- Cluster Naming and Management: The need for unique cluster names and the difficulty in changing these names add complexity to management. Additionally, up to version 1.9, Karmada did not allow hyphens in cluster names. Often, the solution was to create a new cluster and an additional context on the same host.
- Dependence on KIND Tool: Although Karmada's requirements specify the need to use this tool, it has been discovered that it is not necessary, though recommended. The need to install it along with GO increases the required disk space, often resulting in some duplication.

  It is recommended to assign a unique name to clusters during their initial creation using Kubeadm.
- Dashboard in Development: The dashboard, while proposing an interesting visualization, is in a very early phase, which presents two problems:

  1. The community code has bugs that need to be patched manually.
  2. Being an open-source tool maintained by the community and highly dependent on Karmada's evolution as a tool, its development is quite outdated.
- Manual Configuration of Images and API Server Issues: An error in version 1.9 of Karmada requires manual configuration of images, and the API server does not correctly display the pods deployed by namespace.

Kubernetes is a heavy system, so it is necessary to adapt deployment solutions to the available hardware capacity. Although the system's heaviness and resource consumption by network plug-ins represent significant disadvantages, the flexibility and resource optimization through the selection of suitable tools provide a clear advantage in terms of performance and efficiency.

Identified Problems

- System Heaviness: The system is heavy and uses many resources, making it unsuitable for low-capacity hardware. This can lead to a significant performance drop and potential operational failures in limited hardware.
- Resource Consumption by Network Plug-ins: The deployment of network plug-ins like Calico in low-performance systems consumes the remaining resources, further reducing the system's operational capacity, making low-capacity hardware even less efficient.

Alternatives

- K3s or Kubernetes Lite: Evaluating alternatives like K3s for lightweight systems shows a flexible and adaptable approach for different hardware capacities. This alternative is ideal for devices like the Nvidia Xavier but costly for devices like the Raspberry Pi.

- Use of Docker Swarm for Lightweight Devices: The decision to use Docker Swarm for devices like the Raspberry Pi due to its lower resource consumption is a pragmatic choice.

Custom Solution

To address the monitoring challenges and facilitate deployments in the federation clusters, a custom system is proposed. This system allows each cluster to be deployed using the Kubernetes API, providing a more direct and efficient solution. This solution is proposed as it greatly eases the learning curve and allows the project objectives to be met without compromising functionalities or leaving aspects unaddressed.

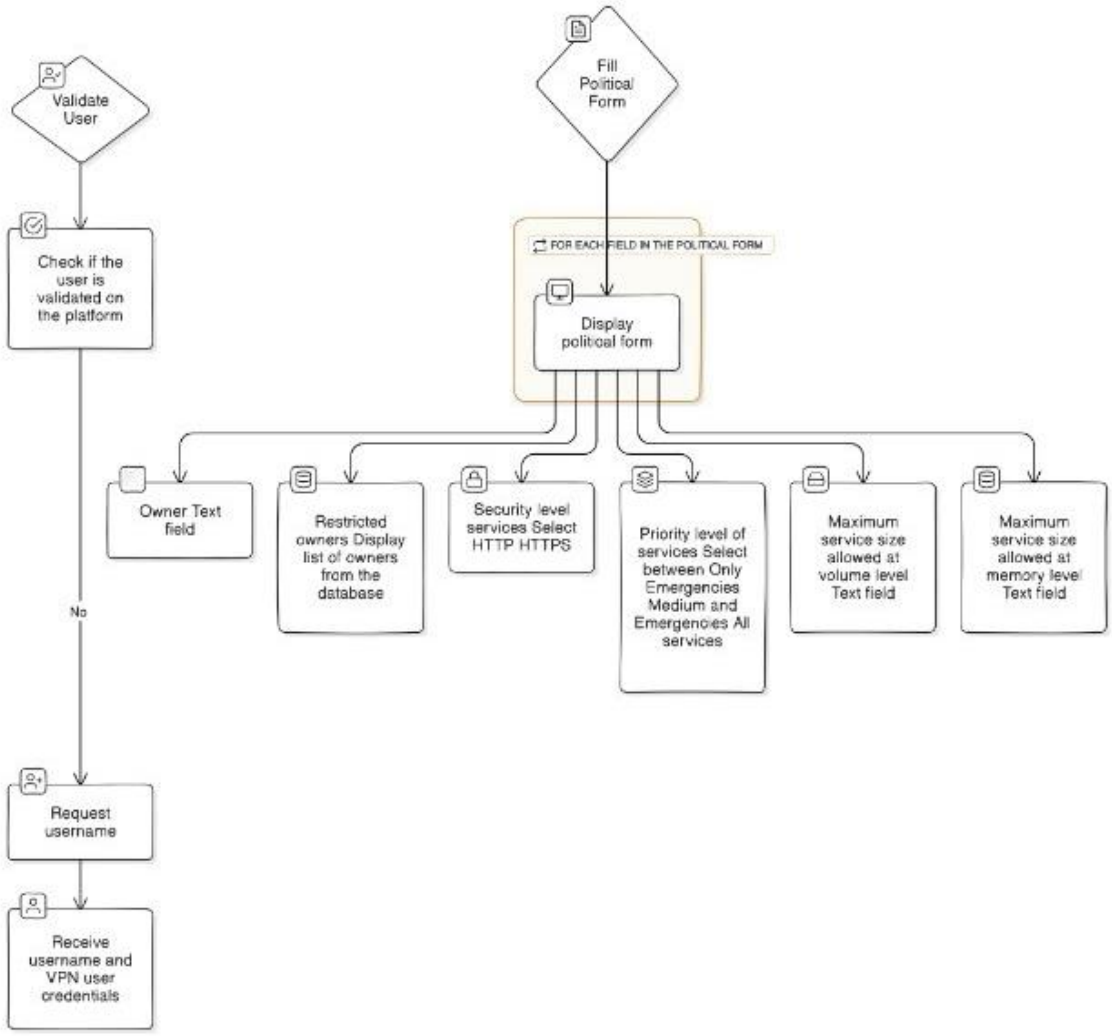
Additionally, the option to deploy Karmada is included to deepen the study, as it could be a useful reference for future deployments of federations that may reach a high number of clusters and wish to leverage Karmada's propagation rules for automatic service scaling, regardless of the associated learning curve. This ensures the study covers both small environments and large cluster infrastructures.

7. Conclusions

In conclusion, the adoption of a federation strategy in Kubernetes clusters offers organizations a comprehensive approach to resource management, performance optimization, security reinforcement, and operational efficiency, ultimately leading to a more robust and scalable containerized infrastructure.

- **Resource Optimization and Performance:** The proposed federation strategy enables resource management optimization and enhances Kubernetes cluster performance by evenly distributing workloads among federated nodes. By establishing admission, control, and priority policies, we ensure efficient resource allocation and high availability of deployed applications, resulting in improved performance and greater scalability of the container environment.
- **Security and Compliance:** Security is a fundamental priority in the federation strategy, with the implementation of security measures at both the federated cluster and application levels to protect data and ensure the integrity and confidentiality of communications. Established control policies and security practices ensure compliance with security standards and mitigate potential vulnerabilities and security risks.
- **Operational Efficiency and Management Simplification:** Federating Kubernetes cluster nodes enables unified and simplified container environment management, reducing operational complexity and associated costs. Deployment automation, centralized configuration management, and real-time monitoring provide greater operational efficiency and facilitate proactive issue detection and resolution, enhancing user experience and customer satisfaction.

The following figure shows a summary of the federation strategy.


Validate User
Fill Political Form
Check if the user is validated on the platform
FOR EACH FIELD IN THE POLITICAL FORM
Display political form
Owner Text field
Restricted owners Display list of owners from the database
Security level services Select HTTP HTTPS
Priority level of services Select between Only Emergencies Medium and Emergencies All services
Maximum service size allowed at volume level Text field
Maximum service size allowed at memory level Text field
No
Request username
Receive username and VPN user credentials

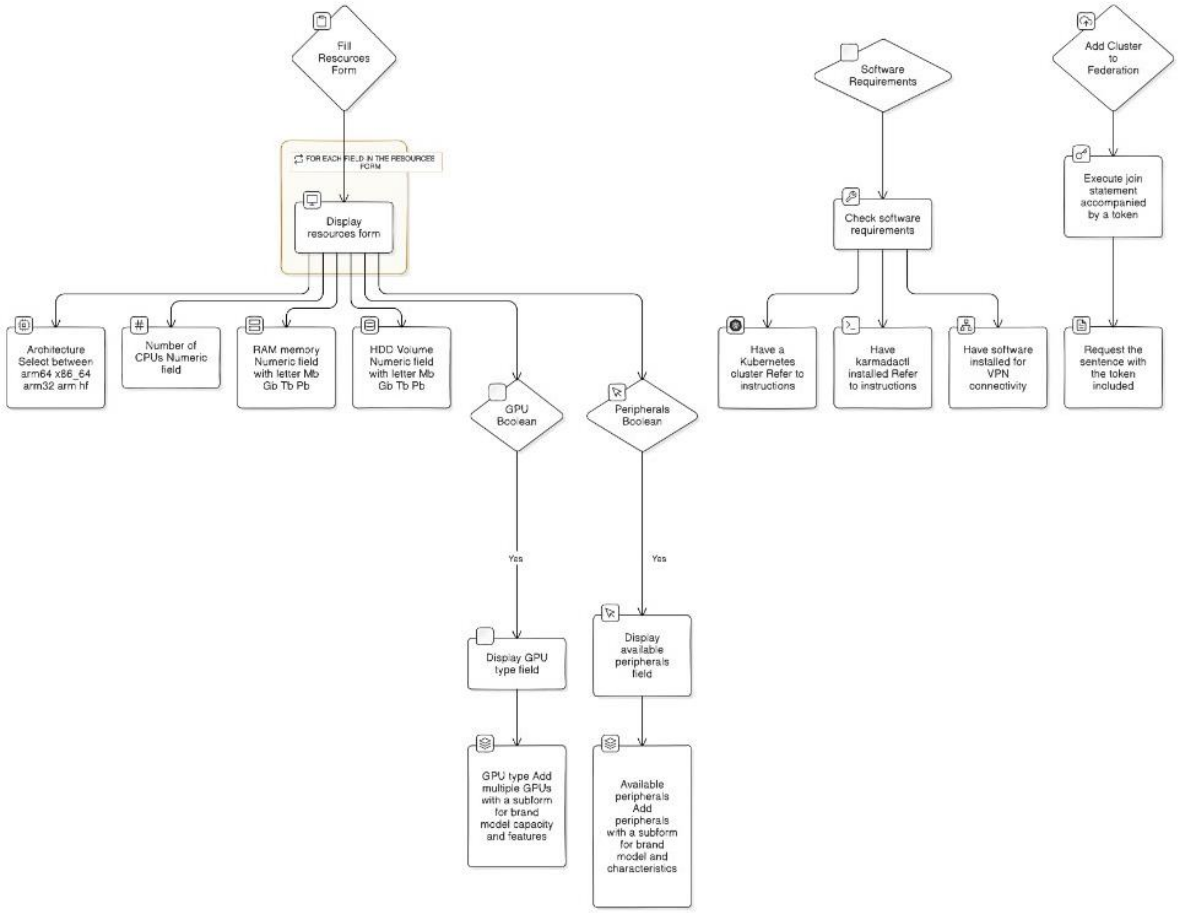


FIGURE 13 FEDERATING CLUSTER

8.References

[1] J. Estublier, H. Verjus, P.-Y. C. 2. R. de la Chimie and B. P. 5. G. C. France, "Building Software Federations," [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=28100264b796590bf4750d348c328d6844efdd97.

[2] A. Buecker, P. Ashley and N. Readshaw, "Federated Identity and Trust Management," ibm.com, 2008. [Online]. Available: https://www.redbooks.ibm.com/redpapers/pdfs/redp3678.pdf.

[3] S. S. Y. Shim, G. Bhalla and V. Pendyala, "Federated identity management," 2005. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1556498.

[4] P. Goldstein, "SAML vs. OAuth: What is Federated Identity Management?," 13 01 2020. [Online]. Available: https://fedtechmagazine.com/article/2020/01/federated-identity-management-saml-vs-oauth-perfcon.

[5] D. Rountree, Federated Identity Primer, Rockland, MA, Estados Unidos de América: Syngress Media, 2012.

[6] M. Tamer Özsu, "Distributed Database Systems," 2002. [Online]. Available: https://www.researchgate.net/publication/228806512_Distributed_Database_Systems.

[7] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," 1990. [Online]. Available: https://dl.acm.org/doi/epdf/10.1145/96602.96604.

[8] M. Gómez y Sonnenberger, Middleware and Cloud Computing, 2012.

[9] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai and M. Kunze, "Cloud Federation," 2011. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=07903e3ab11d69b1c38620bbaaba34fd75e26642.

[10] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016. [Online]. Available: http://arxiv.org/abs/1610.05492https://arxiv.org/abs/1610.05492.

[11] J. Li, "Content federation: The next stage of composability," 18 May 2023. [Online]. Available: https://hygraph.com/blog/content-federation-and-composable-architecture.

[12] Redhat, "What is lightweight directory access protocol (LDAP) authentication?," 3 June 2022. [Online]. Available: https://www.redhat.com/en/topics/security/what-is-ldap-authentication.

[13] G. T. D. G. Y. L. Xiaofeng Cao, "Edge Federation: Towards an Integrated Service," 27 Mar 2020. [Online]. Available: https://arxiv.org/pdf/1902.09055.

[14] M. D. P. N. P. J. L. H. V. P. Dinh C. Nguyen, "Federated Learning for Internet of Things: A Comprehensive Survey," 2021. [Online]. Available: https://arxiv.org/pdf/2104.07914.pdf.

[15] N. S. ,. X. Y. ,. S. C. ,. E. B. ,. M. C. ,. J. J. W. K. ,. N. M. ,. M. N. ,. C. E. O. G. R. ,. R. S. K. S. ,. Z. Y. RAED KONTAR, "The Internet of Federated Things (IoFT): A Vision for the Future and In-depth Survey of Data-driven Approaches for Federated Learning," 2021. [Online]. Available: https://arxiv.org/pdf/2111.05326.pdf.

[16] D. Heimbigner and D. McLeod, "A federated architecture for information management," 1985. [Online]. Available: https://dl.acm.org/doi/epdf/10.1145/4229.4233.

[17] D. McLeod and D. Heimbigner, "A federated architecture for database systems," 1980. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/1500518.1500561.

[18] M. T. Ozsu and P. Valduriez, Principles of distributed database systems: United States edition, Pearson, 1999.

[19] S. E. Sampson and C. M. Froehle, "Foundations and Implications of a Proposed Unified Services Theory," 2006. [Online]. Available: https://www.researchgate.net/publication/227627129_Foundations_and_Implications_of_a_Proposed_Unified_Services_Theory.

[20] J. A. Ruiz-de-Azua, N. Garzaniti, A. Golkar, A. Calveras and A. Camps, "Towards Federated Satellite Systems and internet of satellites: The Federation Deployment Control Protocol," 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/5/982.

[21] A. Golkar and I. Lluch i Cruz, "The Federated Satellite Systems paradigm: Concept and business case evaluation," 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576515000624.

[22] K. N. K. S. Rick Kazman, "Understanding Patterns for System of Systems Integration," *Carnegie Mellon University Research Showcase,* 2013.

[23] "Learn Kubernetes Basics," p. https://kubernetes.io/es/.

[24] K. W. David Eads, "Karmada and Open Cluster Management: two new approaches to the multicluster fleet management challenge," *CNCF projects are the foundation of cloud native computing,* p. https://www.cncf.io/, 2022.

[25] M. A. Razzaque, M. Milojevic-Jevric, A. Palade and S. Clarke, "Middleware for internet of things: A survey," 2016. [Online]. Available: https://www.researchgate.net/publication/283651343_Middleware_for_Internet_of_Things_A_Survey.

[26] "gRPC. Introduction to gPRD," [Online]. Available: https://grpc.io/docs/what-is-grpc/introduction/.

[27] Q. Zhu, C. Rieger and T. Basar, "A hierarchical security architecture for cyber-physical systems," 2011. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6016081.

[28] K. Goel, F. Emamjome and A. ter Hofstede, "Data governance for managing data quality in process mining," 2021. [Online]. Available: https://aisel.aisnet.org/icis2021/governance/governance/9/.

[29] U. Fragoso-Rodriguez, M. Laurent-Maknavicius and J. Incera-Dieguez, ""Federated Identity Architectures"," [Online]. Available: http://www-public.imtbs-tsp.eu/~lauren_m/articles/mlaurent-mcis06.pdf..

[30] K. Bernadini, "CCPA vs GDPR," 11 04 2020. [Online]. Available: https://www.gdpreu.org/ccpa-vs-gdpr/.

[31] Ł. Kufel, "Tools for Distributed Systems Monitoring," 2016. [Online]. Available: https://www.researchgate.net/publication/311863266_Tools_for_Distributed_Systems_Monitoring.

[32] H. Fleischmann, J. Kohl, J. Franke, A. Reidt, M. Duchon and H. Krcmar, "Improving maintenance processes with distributed monitoring systems," 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7819189.

[33] N. Mehta, "How to accurately monitor distributed systems?," 2018. [Online]. Available: https://www.catchpoint.com/blog/monitoring-distributed-systems.

[34] J. D. T. K. G. B. George Coulouris, Distributed Systems: Concepts and Design, Addison Wesley, 2005.

[35] Q. J. Ning Zhang, "An Overview of Data Governance," 2016. [Online]. Available: https://www.researchgate.net/profile/Zhang-Ning-25/publication/321899578_An_Overview_of_Data_Governance/links/5a3867a8aca272a6ec1e8864/An-Overview-of-Data-Governance.pdf.

[36] W. Stallings and L. Brown, Computer security: Principles and practice, global edition -- (perpetual access), Londres, Inglaterra: Pearson Education, 2023.

[37] D. Pöhn and P. Hillmann, "Reference service model for federated identity management," 2021. [Online]. Available: http://arxiv.org/abs/2108.06701https://arxiv.org/abs/2108.06701.

[38] T. Mikito, "Distributed systems for fun and profit," 2013. [Online]. Available: http://book.mixu.net/distsys/.

[39] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: A comprehensive survey," 2015. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6994333.

[40] C. D. Alwis, A. Kalla, Q.-V. Pham, P. Kumar, K. Dev, W.-J. Hwang and M. Liyanage, "Survey on 6G frontiers: Trends, applications, requirements, technologies and future research," 2021. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9397776.

[41] E. F. Silva, D. C. Muchaluat-Saade and N. C. Fernandes, "ACROSS: A generic framework for attribute-based access control with distributed policies for virtual organizations," 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17316060.