UNICO I+D Project
6G-INTEGRATION 02

6G-INTEGRATION-02-E13

Proposal for secure communication between federated elements with valid agreement and a secure registration mechanism for candidates in the federation

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia

Document properties

Document number	6G-INTEGRATION-02-E13
Document title	Proposal for secure communication between federated elements with valid agreement and a secure registration mechanism for candidates in the federation
Document responsible	Irene Rubio, Monica Villalobos, Yaima Fiallo (Capgemini Engineering)
Document editor	Ernesto Correa, Yaima Fiallo, Noel Ruiz, Miguel Juaniz, Jon Iriarte (Capgemini Engineering)
Target dissemination level	Private
Status of the document	In Progress
Version	1.0
Delivery date	31/12/2024
Actual delivery date	31/12/2023 (Preliminary)

Production properties

Reviewers	Noel Ruiz (Capgemini Engineering)

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 Tables

List of Acronyms

HTTP: HyperText Transfer Protocol

HTTPS: HyperText Transfer Protocol Secure

JWT: Jason Web Tokens

MITM: Mitigation of Man-in-the-Middle

NTN: Non-Terrestrial Network

PoC: Proof of concept

RBAC: Role-Based Access Control

SSL: Secure Sockets Layer

TCP: Transmission Control Protocol

TLS: Transport Layer Security

VPNs: Virtual Private Network

1. Introduction

This document gives a general idea of how to achieve a secure configuration between clusters, something crucial to guarantee the integrity, confidentiality and availability of the data and services you manage in a distributed environment. Below is a general guide on best practices for achieving a secure cross-cluster configuration:

1. **Authentication and Authorization:** to achieve this we must use strong authentication mechanisms to validate the identity of the clusters against each other. This will include the use of certificates and tokens.

   We will implement appropriate authorization to ensure that only authorized entities, with a trusted certificate, have access to specific resources in the clusters/can associate with the federation.

2. **Secure Communication:** encrypts communication between clusters using secure protocols such as Transport Layer Security/Secure Sockets Layer (TLS/SSL). This is crucial to protect data being transmitted between clusters.

3. **Secure Networks:** use private networks and Virtual Private Network (VPNs) to connect clusters, rather than relying solely on connections over the public Internet.

   Configure firewall rules to restrict unwanted traffic and allow only necessary connections between clusters.

4. **Credential Management:** carefully manage cluster access credentials. Use secret management solutions and avoid storing credentials in plain text.

5. **Audit and Monitoring:** implement auditing and monitoring systems to detect and respond to unusual activities or unauthorized access attempts.

   Configure alerts for security events and perform periodic log analysis.

6. **Updates and Patches:** keep operating systems, software, and services up to date on each cluster. Apply security patches regularly.

7. **Resource Isolation:** use isolation practices to limit access and impact if a cluster is compromised.

   Implement minimum necessary access policies to reduce the attack surface.

8. **Backup and Recovery:** implement a backup and recovery plan for critical data. Ensure that you can restore clusters to a safe state in the event of data loss or failure.

9. **Documentation:** thoroughly documents security configuration and management procedures. This makes collaboration and problem solving easier.

10. **Security Tests:** perform regular security testing, such as penetration assessments, to identify potential vulnerabilities and weaknesses in your configuration.

11. **Regulatory Compliance:** make sure you comply with the safety regulations and standards applicable to your industry.
12. **Training and Awareness:** provides security training to teams responsible for cluster management and promotes awareness of security best practices.

Starting from these generalities, this document will focus on adapting these concepts to a system implemented with Karmada, the Kubernetes-based solution for federated systems.

2. Security risks in a multi-cluster federation

Security vulnerabilities in a multi-cluster federation refer to potential weaknesses or risks in the security architecture of a system that spans multiple clusters within a federation. A multi-cluster federation typically involves the management and coordination of multiple Kubernetes clusters to work together as a single entity, allowing centralized control and resource management.

Here are some common security considerations and potential vulnerabilities in a multi-cluster federation:

1. **Errors in the Security Configuration of Each Cluster:**
    - Uncertified Components (CRD/CRT)
    - Open Ports (Firewall)
    - Backdoors (SSH → honeypots to detect potential unauthorized accesses)
    - Poor User Permission Management (root/users)
    - Deployment of Unverified Images or Images with Known Vulnerabilities: Using unverified container images or images with known vulnerabilities can expose the system to exploits. (Kyverno)
    - Incorrect Persistent Storage Configuration: Incorrect configurations may allow unauthorized access to sensitive data.
2. **Lack of Authentication and Authorization Management Against the Central Node:**
    - Incorrect Security Management for Adding a Cluster to the Federation: review the configuration of federated clusters to ensure they adhere to security best practices (certificates, tokens).
    - Identity and Access Management: evaluate how identities and accesses are handled within the federated environment. (Roles)
        - Weak or Leaked Passwords: weak passwords can be exploited, and leaked passwords can provide unauthorized access.
        - Incorrect Role-Based Access Control (RBAC) Configuration: incorrect configurations may allow users to obtain more permissions than necessary.
3. **Secure Communication and Networking Between Clusters:**
    - Insecure Communications between Microservices and Cluster Elements: analyse communications between clusters to identify possible weaknesses in encryption and authentication. (Istio, mTLS, Resilience)
    - Use of VPN
4. **Poor Propagation of Federation Rules and Policies for Resource Usage:**

According to [1] it can be summarized as follows:

- Inability to define and propagate a set of federation rules that can be assimilated by the new cluster (e.g., enforce the definition of a series of informative tags useful for selective/restrictive propagation of services) → (use propagation and override rules in Karmada)
- Resource Reliability
- Uneven Resource Utilization: without resource propagation, there's a risk of uneven resource utilization across clusters. Some clusters might be overloaded while others remain underutilized, leading to inefficient resource management.
- Manual Workload Placement: without automated propagation, you may need to manually decide where to deploy workloads in each cluster. This can be time-consuming, error-prone, and may not be optimized for changing resource demands.
- Increased Operational Complexity: manually managing workload distribution across multiple clusters increases operational complexity. Resource propagation simplifies this process and provides a more automated and dynamic approach.
- Difficulty in Scaling: scaling applications horizontally across clusters may be more challenging without resource propagation. This can limit your ability to quickly adapt to changing workloads and scaling requirements.
- Limited High Availability (HA) Options: resource propagation is often crucial for achieving high availability by distributing workloads across clusters. Without it, ensuring HA might require more manual effort and planning.
- Risk of Overloading Clusters: without automated resource management, there is a higher risk of overloading specific clusters, leading to degraded performance or potential failures.
- Inefficient Cluster Utilization: the absence of resource propagation may result in suboptimal cluster utilization, as workloads may not be efficiently distributed based on the available resources and constraints.
- Increased Management Overhead: manually managing deployments without resource propagation can lead to increased management overhead, especially in scenarios where clusters are added or removed dynamically.
- Difficulty in Adhering to Policies: if there are specific policies or constraints for workload placement (such as regulatory requirements or business rules), adhering to them without resource propagation may require additional effort and monitoring.
- Less Dynamic Resource Allocation: resource propagation enables dynamic and automated resource allocation based on the current state of the clusters. Without it, resource allocation may be less dynamic and responsive to changes in demand.

- Complex Scaling Strategies: implementing complex scaling strategies, especially those involving multiple clusters, may become more challenging without the assistance of resource propagation.

5. **Poor or Non-existent Governance Management in the Distribution of Services Within the Federation:**
    - Service Issues → (Service export/import for MULTI-SERVICE GOVERNANCE)
    - Secure Governance: inability to correctly manage the "usage rules" of each of the different federated clusters. For example, one cluster may not allow the execution of a type of service with specific characteristics in a cluster that restricts those characteristics (Example: a cluster of X that does not allow an app owned by Y). (Kyverno)
6. **Monitoring:**
    - Management of Security Updates and Patches
    - State of Resources (Prometheus, Kyverno)
    - Management of Logs and Documentation

2.1. Risk assessment by probability of occurrence

To assess vulnerabilities in a federated system with Karmada, the first step is to identify potential security risks and threats. In this environment, the system consists of a central node that acts as the federation point and dispersed nodes from different providers that can federate with each other. The following table details the potential risks, their probability of occurrence, the associated impact, and the resulting risk level. These risks range from configuration errors to weaknesses in authentication management, insecure communications, inefficient propagation of federated rules, and governance weaknesses. This assessment provides detailed insight to inform decision-making and implementation of security measures in a federated environment with Karmada.

Here's the detailed risk assessment table for a computing cluster within a Non-Terrestrial Network (NTN):

No.	Risk	Probability of Occurrence	Impact	Risk Level
1	Security Configuration Errors in Each Cluster	High	High	High
2	Lack of Authentication and Authorization Management Against the Central Node	Moderate	High	Moderate
3	Insecure Communications Between Microservices and Cluster Elements	Moderate	High	Moderate

4	Poor Propagation of Federation Rules and Policies for Resource Usage	High	High	High
5	Poor or Non-existent Governance Management in the Distribution of Services Within the Federation	Moderate	High	Moderate
6	Inadequate Monitoring	Low	High	Moderate

TABLE 1 RISK ASSESSMENT FOR A COMPUTING CLUSTER WITHIN AN NTN

Notes:

- **Probability of Occurrence:**
  - **High:** Likely to occur frequently.
  - **Moderate:** May occur under certain circumstances.
  - **Low:** Unlikely to occur but not impossible.
- **Impact:**
  - **High:** Severe consequences affecting multiple aspects of the cluster.
  - **Moderate:** Significant consequences but manageable.
  - **Low:** Minimal consequences with easy mitigation.
- **Risk Level:**
  - **High:** Requires immediate attention and mitigation.
  - **Moderate:** Needs careful monitoring and mitigation efforts.
  - **Low:** Monitor but may not require urgent action.

This assessment provides a more nuanced view of the potential risks in a NTN computing cluster, considering both the likelihood of occurrence and the potential impact on the system. Adjustments can be made based on the specific context and characteristics of the NTN environment.

2.2. Risks, preventive and/or mitigating actions

The next table show the risk evaluation considering preventive actions.

Risk	Preventive/Mitigation	Possible apps/pluggins
Security Configuration Errors in Each Cluster – Uncertified cluster	Use trusted certificates – mTLS	Istio
Security Configuration Errors in Each Cluster – open ports	Config firewall – Periodical port scan	Linux firewall

Security Configuration Errors in Each Cluster – SSH unallowed access	Use HoneyPot	SSH Honeypots
Security Configuration Errors in Each Cluster – Deployment of Unverified Images or Images with Known Vulnerabilities	Check container image signatures and attestations for software supply chain security	Kyverno
Security Configuration Errors in Each Cluster – Incorrect Persistent Storage Configuration	Data encryption, Use RBAC, namespace and network policies	Configure K8s native
Lack of Authentication and Authorization Management Against the Central Node - Incorrect Security Management for Adding a Cluster to the Federation	Validate node + registration token + Authentication mechanism	Custom development + Karama registration using token
Lack of Authentication and Authorization Management Against the Central Node - Identity and Access Management	Registration token – RBAC	Keycloak
Secure Communication and Networking Between Clusters - Insecure Communications between Microservices and Cluster Elements	Use VPN - use mTLS	OpenVPN, Istio
Secure Communication and Networking Between Clusters – Insecure networks	Use VPN	OpenVPN,
Poor Propagation of Federation Rules and Policies for Resource Usage - Inability to define and propagate a set of federation rules that can be assimilated by the new cluster	Consider propagation and propagation rules	Use and configure Karmada Propagation
Poor Propagation of Federation Rules and Policies for Resource Usage - Resource Reliability	Resource Monitoring	Prometheus, Grafana, Kyverno

TABLE 2 RISK EVALUATION CONSIDERING PREVENTIVE ACTIONS

2.3. Proposal of minimum qualifying conditions and self-checking mechanisms

Minimum Qualification Conditions:

- Security Requirements: defines the minimum conditions that federated clusters must meet to be considered secure. This may include:
    - Firewall requirements
    - Certification requirements
    - Encryption requirements (e.g., etcd, encryption, mTLS)
    - Minimum propagation rules and tags
    - Use of tools like Karmada, ISTIO for mTLS and Kyverno

Self-Verification Mechanisms:

- Verification Scripts: provides scripts or self-verification tools that administrators can run periodically to assess the security status of the federated system.
    - Example: Kyverno

Additional Considerations:

- Updates and Patches: ensure the inclusion of best practices for the timely application of updates and security patches in federated clusters.
- Education and Awareness: include recommendations for ongoing training and security awareness for system administrators and end-users.

3. Software communications security

3.1. Secure transport layer connection (TLS/SSL) (mTLS)

Implementing TLS/SSL at the transport layer plays a crucial role in ensuring secure communications in software systems. These technologies are instrumental in maintaining confidentiality, integrity, and mutual authentication between communication endpoints. Mutual TLS (mTLS), an extension of TLS/SSL, further strengthens security by requiring both parties in the communication process to authenticate each other, thereby creating a trusted and secure channel. In the context of service meshes like Istio, mTLS becomes particularly significant.

Istio, a service mesh framework, uses mTLS to secure communication between microservices. Istio's mTLS implementation ensures that all communication across the service mesh is cryptographically secure and authenticated. By default, Istio operates in a 'PERMISSIVE' mode, which allows services to accept both plaintext and mTLS traffic. However, it can be configured to 'STRICT' mode, in which services must use mTLS exclusively, or 'DISABLE' mode, where all traffic is plaintext. [2] [3]

3.2. Secure device authentication (certificates)

An integrated approach for secure communication between federated elements is proposed, focusing on the secure authentication of devices by means of certificates and the development of an application to validate these certificates.

The basis of our proposal lies in the implementation of digital certificates. Devices present a digital certificate containing their public key and a digital signature, which is verified by the authentication server to provide additional security. These certificates, issued by a trusted certificate authority (CA), ensure the identity of the devices within the federation. Each device has a unique certificate, which guarantees its authenticity and the integrity of the communication. This approach prevents man-in-the-middle attacks and ensures that only authenticated devices can participate in the federation. They facilitate mutual authentication and secure connections between two devices via public key infrastructure (PKI), ensuring the authenticity of machine-to-machine (M2M) communications. [4]

Certificate-Based Authentication (CBA) uses digital certificates, obtained through cryptography, to identify users, machines or devices before granting access to networks, applications or other resources, ensuring that only authorised devices are connected to the organisation's network. Certificate-based methods are recommended for IoT devices, as they support security measures such as multi-factor authentication and TLS/SSL, which supports certificate-based two-way authentication. [5]

We will incorporate the use of tokens generated by karmadactl, which will have a lifetime of one day. These tokens will provide an additional method of authentication and authorisation for federated

operations. The temporality of the token ensures that, in case of compromise, unauthorised access is limited in time. [1]

We will develop a specialised application that will allow registered users to verify and manage certificates on their devices. This application will facilitate the verification of the validity of certificates, including checking the digital signature and verifying against the CA's Certificate Revocation List (CRL). In addition, the application will allow users to update or revoke certificates when necessary.

The combination of secure device authentication with certificates, a dedicated application for managing these certificates, and the use of karmadactl's temporary tokens provides a robust framework to ensure secure and efficient communication between federated devices. This approach not only improves security but also the scalability and management of device federation.

3.3. Encryption strategy (etcd)

An encryption strategy is a comprehensive plan designed to protect the confidentiality, integrity and accessibility of data in a computer system. In the context of a Kubernetes cluster, this involves securing data communication and storage through the use of encryption technologies. This is generally achieved through the use of TLS/SSL certificates, which are issued and managed by Certificate Authorities (CAs). These certificates are used to encrypt and authenticate communication between the different components of the cluster, such as nodes, services and users. An effective encryption strategy is crucial to maintain security and privacy in distributed and multi-user environments such as Kubernetes. [6] [7] [8] [9]

This set of scripts is used to generate certificates using the OpenSSL command, facilitating secure communication in environments using the TLS/SSL protocol. Key components include:

- Certification Authorities (CAs): Three CAs are generated: **front-proxy-ca**, **server-ca**, y **etcd/ca**, trusted entities that issue certificates for other entities in the cluster.
- Leaf Certificate Generation: Specific scripts (generate_leaf.sh) are used to issue leaf certificates for individual entities such as web servers or services.
- Subject Alternative Name (SAN): Leaf certificates may have their SAN modified to specify applicable hostnames or IP addresses to which certificates apply. Possibility of changing the SAN in leaf certificates.
- Separate CAs and Reusability: The scripts for generating CAs and leaf certificates are separate, allowing the reuse of CAs by modifying the SAN of leaf certificates.
- Etcd/ca: Use is made of a CA etcd/ca and related scripts for etcd, important for secure communication within the Kubernetes cluster.
- Exclusion of Third Party Provided Etcd: In some cases, certain scripts (generate_etcd.sh and csr_config/etcd) may be ignored if a third party provided etcd is used.

In summary, these scripts and configurations are designed to simplify the generation and management of security certificates in a Kubernetes environment. Secure certificate generation is essential to ensure authentication and encrypted communication within a Kubernetes cluster. It is recommended to follow the instructions provided in the documentation or comments within the scripts to ensure a correct and secure implementation.

3.4. VLAN network isolation

The implementation of network segmentation using VLANs (Virtual Local Area Networks) is a strategic approach to improve security, performance and network management in a context of secure communication between federating elements with valid agreement and secure federation candidate registration mechanism. Key technical aspects of this strategy are detailed below: [10]

- VLAN Basics for Network Isolation: VLANs are logical groupings of devices connected to the same network, regardless of their physical location. They allow logical partitioning within a single switch, creating multiple virtual LANs where physical switch segmentation is not possible. Each VLAN is assigned a unique number, allowing network administrators to organise and separate network traffic.
- Advantages of VLANs in Network Management: VLANs improve network efficiency, scalability and security, as well as simplify network management. They are useful for logically separating devices by function, creating isolated guest networks, prioritising critical traffic, and optimising large-scale networks. They provide a number of benefits, such as logical network segmentation, improved network security, increased operational efficiency, improved throughput and reduced latency, reduced hardware costs and requirements, simplified device management, broadcast troubleshooting and broadcast domain reduction, and simplified network topology.
- Implementing VLANs for Secure Communication in Federations: In a federated context, segmenting the network into VLANs or subnets can limit the exposure and impact of potential threats. This is especially relevant for federations that require a high degree of security and control over communication between members. For example, in a federation involving collaboration between multiple organisations, each organisation could have its own VLAN, ensuring that network traffic is confined to its specific environment. This not only improves security by limiting access to sensitive resources, but also facilitates the management of network traffic and reduces the possibility of interference or cross-cyber-attacks.

The use of VLANs in network segmentation is an effective technique for improving the security and efficiency of communication in federated environments. It provides a way to segment and manage network traffic, ensuring the integrity and confidentiality of communications between federated elements.

3.5. Virtual Private Networks

Using a VPN can provide an additional layer of security at the link level. It encrypts all network traffic between network nodes, ensuring secure communication even on a public network.

Istio [11] allows, as we have seen before, to deploy a service mesh to orchestrate the service and resource sharing between different Kubernetes clusters. With it, it will be possible to deploy various clusters under the Istio management to share resources and workload.

To interconnect two clusters, Istio, in its simplest way, will need a control plane in each cluster. When a service in one cluster wants to communicate with another service in the same cluster, this communication will be secured by Kubernetes mechanisms. However, when a service wants to communicate with a service deployed on another cluster, it must be done through the Istio gateway of the other cluster. This communication is secured in the transport layer by the Istio mechanisms how we have already seen.

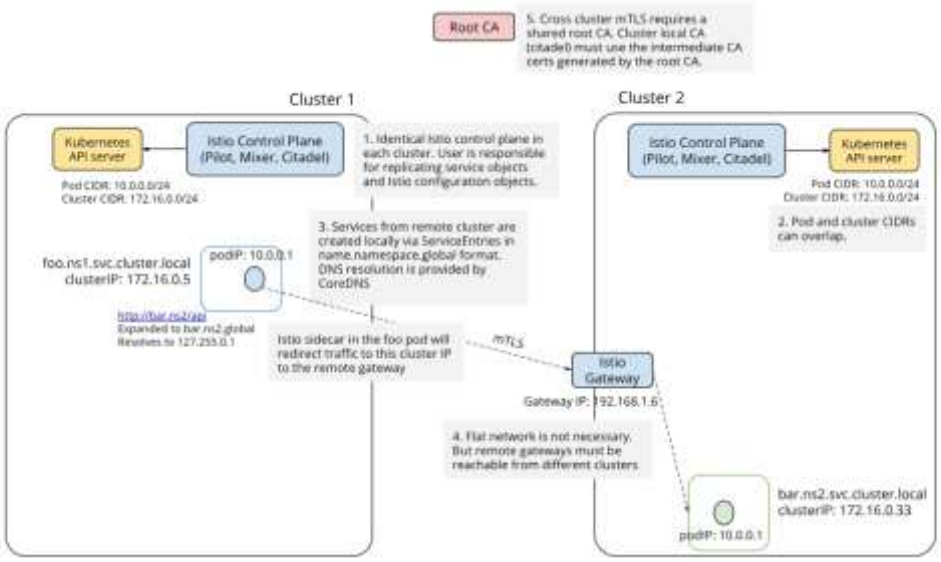

FIGURE 1. ISTIO MESH SPANNING MULTIPLE KUBERNETES CLUSTERS USING MULTIPLE ISTIO CONTROL PLANES AND GATEWAY TO REACH REMOTE PODS. [11]

It may be possible to offer an additional security layer on the link level through a VPN, which encrypts all the network traffic between the nodes ensuring the communications even in a public domain.

To provide this layer, we could, whenever possible, deploy a VPN between clusters. And in the case where all clusters have VPN connectivity it would no longer be necessary to deploy two control planes simplifying the deployment on the Istio side. In this case, an Istio deployment with remote connectivity would be enough. This would communicate with the remote plane through a VPN connection and manage resource sharing between cluster services through the VPN as well.

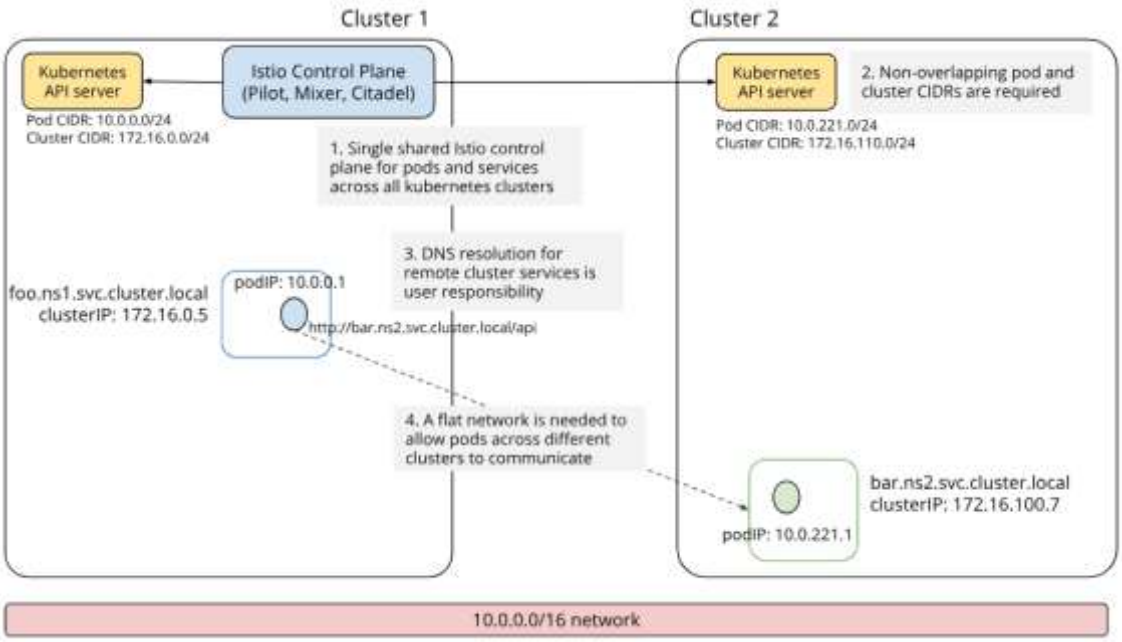

FIGURE 2. ISTIO MESH SPANNING MULTIPLE KUBERNETES CLUSTER WITH DIRECT NETWORK ACCESS TO REMOTE PODS OVER VPN. [11]

A compromise would then appear in which on the one hand we have a deployment with security at the transport level between clusters and on the other, an added layer of security at the link level but which would add complexity in its deployment and in the Istio's one since it would only require a control plane, but the user would be responsible for more develops and configurations.

In conclusion, we could say that in a development environment, the deployment of VPN connectivity would not be necessary. But it would have to be considered in a production environment where critical data is exposed publicly.

3.6. Evaluation of the need to use IPsec (secure tunnel)

Setting up secure tunnels, such as SSH or IPsec, can protect traffic at the link layer. These tunnels can provide encryption and authentication for communication between two points.

We have already evaluated the link and transport layer. But it is possible to add security in the network and application layers too.

For the first one, IPsec appears [12], a set of rules and protocols which add encryption and authentication to IP. With this protocol it is possible to add security when the data passes through the public Internet network, encrypt the application data, quickly authenticate the data if they come from a known sender and set encrypted circuits between two connection nodes which encrypts all the data that passes through them.

IPsec tunnels.

The IPsec encryption is a software function which encodes the data to secure its content from no authorized parts. The data is encrypted by a encryption key and a decryption key is needed to recover it. It allows many encryption types as AES, Blowfish, triple DES, ChaCha and DES-CBC.

IPsec is shown as a general protocol of the network layer. In this project we study the appliable security to an environment in which the security is controlled by frameworks which manage the security and modify this framework or add functionalities as securing the communication with IPsec tunnels is not simple and it exceeds this project scope.

In the application layer case, the main security protocol is SSH whose function is to provide the remote access to a server through a secure channel in which all the data is secured. There is an option which allows to implement security to this layer without having to get into the guts of the frameworks or adding complex functionality.

The honeypots. [13]

Honeypots are mechanisms that simulate the real software that we want to protect, and they expose them to possible attackers as bait. When attackers see an exposed service and try to attack it, honeypots receive attack data and help to prevent attacks. Honeypots could be added to Kubernetes using Vcluster.

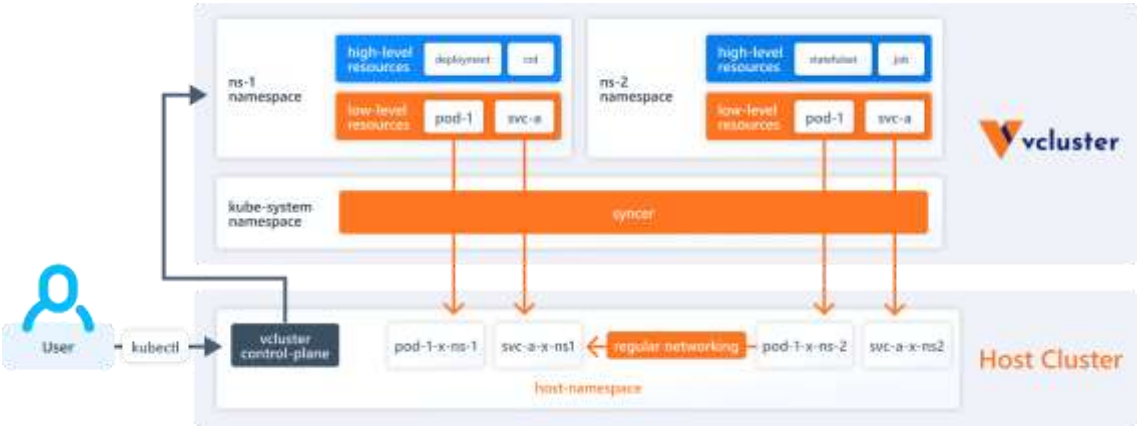

FIGURE 3. WHAT IS A VCLUSTER - ARCHITECTURE. [13]

Vcluster allows you to create virtual clusters on top of real clusters. Several of them can be run and are in separate environments from which the host cluster cannot be reached. As the virtual cluster is running within its own isolated place, it allows us to expose very sensitive items within it as the attacker does not actually see the sensitive information, but a trap.

Inside these Vclusters a fake SSH server could be installed, for example, so that whoever wants to attack through an SSH connection would connect to these virtual clusters, and if a high activity honeypot is deployed, we can trap the attackers by simulating a real connection and analyze their behavior to protect the real assets.

However, we come back to the same point: in our opinion, implementing these honeypots would exceed the scope of the project, it has been seen that it is possible to use and implement them, so they may be considered for future projects or deployments in production environments.

3.7. Monitoring the network

It is also important to implement network monitoring systems to detect suspicious activities or intrusions. This may include monitoring network traffic and detecting attempts at unauthorized access.

Here two concepts appear, the SIEM and the XDR. SIEM, Security Information and Event Management is an integral solution designed to collect, analyze, and correlate security events data in an IT infrastructure. XDR, Extended Detection and Response, extends the detection capabilities and response beyond a single point to address multiple attack vectors.

There are platforms which bring together both systems and protect workloads across on-premises, virtualized machines, containerized software, and cloud-based environment. For instance, Wazuh [14] is a free and open-source security platform.

It's been proven to work in many different use cases like in configuration assessment. Configuration assessment is a process that verifies whether endpoints adhere to a set of predefined rules regarding configuration settings and approved application usage. Done by comparing the current configuration against established industry standard and organizational policies to identify vulnerabilities and misconfigurations. This example is attached to single endpoint security, but it is possible to focus on threat intelligence, security operations and cloud security.

Log data analysis, included into the threat intelligence, involves examining and extracting valuable insights from log files created by different systems. With such insights, we can detect attacks like unauthorized attempts, denial of service, or brute force attacks. In addition, as a future line it would be possible to add AI to that log analysis to predict and avoid attacks. It only needs a SIEM agent deployed in the endpoint which we want to monitor to read the logs written, filter them and communicate with the central part of the system to act consequently. Then, inside the security operations, the incident response reacts to the previous analysis.

With the monitoring we have the threats detected and analyzed but the incident response oversees effectively handling a security incident and restore normal business operations as quickly as possible.

And as an example of cloud security, something in what we are interested in this project, Wazuh offers the possibility of being integrated with container platforms like Docker and Kubernetes and actively monitors container runtime events, application log, and overall container health. Extracting the data directly from the container platforms, it is possible to maintain a whole deployment healthy automatically.

It is also important take care about software updates and patches, keeping devices and systems up to date with the latest security patches addresses known vulnerabilities and ensures a more secure environment.

For that purpose, we can think in any policies management platform as could it be Kyverno. Kyverno is an open-source platform which allow us to set policies to the Kubernetes deployment. Between all

the possible policies that can be applied, there is a category of software Supply Chain Security [15]. That's exactly what we want to do, monitor the software supply chain to avoid vulnerabilities.

Some actions that can be performed with a framework like Kyverno is to perform periodic vulnerability scans on images. Initial scans as part of the build process are necessary, but as new vulnerabilities are discovered the scans must be refreshed.

We can verify CycloneDX SBOM too. SBOM provides details on the composition of a given container image. Having it could be important to ensure that images are built using verified processes.

In general, it is possible to check that the images have been developed following some security standards and that has been signed using valid certifications to avoid the system to update some image to a new version in which malware is injected.

3.8. Multi-cluster security compliance Governance

Multi-cluster: refers to the existence of more than one cluster or set of resources. In the context of technologies like Kubernetes, a cluster is a group of servers working together to run applications and services.

Security Compliance: "Security" refers to protecting a system against threats and ensuring that information is secure. "Compliance" involves adhering to established rules, regulations, or standards. In this case, "Security Compliance" means ensuring that clusters comply with established security standards.

Governance: This term is related to control and management. In the context of technology, "governance" involves establishing rules, policies, and procedures to manage and direct the behavior of systems.

So, "Multi-cluster Security Compliance Governance" means, in simple terms, establishing rules and standards to ensure that multiple sets of resources (clusters) work securely and comply with established security norms. This ensures that even if you have several groups of servers working together, they all follow secure practices and follow security requirements.

Software for multicluster Governance: Kyverno

Kyverno is a policy engine designed specifically for Kubernetes, and its integration into Karmada for multi-cluster governance provides a robust and efficient approach to policy management in federated environments. Policy Engine for Kubernetes. Kyverno is a Cloud Native Computing Foundation project that serves as a policy engine for Kubernetes. Kyverno policies are managed as Kubernetes resources, and do not require learning a new language. This allows the use of familiar tools such as kubectl, git and kustomize for policy management. Kyverno can validate, mutate, generate and clean up Kubernetes resources, as well as verify image and artifact signatures to help secure the software supply chain. The Kyverno CLI can be used to test policies and validate resources as part of a CI/CD pipeline. [15]

3.9. Proposal at transport/link layer level

Federated networks present unique challenges for secure communication due to their distributed and dynamic nature, where nodes frequently vary in configuration, policies, and roles. To address these complexities, it proposes a comprehensive hybrid framework that combines encryption, authentication, and isolation strategies across the transport and link layers. This approach ensures robust protection of both data and control communication while maintaining scalability and adaptability to evolving network conditions.

Core Strategies and Implementation

1. VPN between nodes (Link Layer): VPNs operate at the link layer to establish secure, encrypted tunnels for data exchange between nodes. By encapsulating traffic within secure tunnels, VPNs provide protection against interception and tampering at a foundational network level. Key elements of our VPN implementation include:
    - Authentication Mechanisms: X.509 certificates enable mutual authentication, ensuring that only authorized devices participate in secure connections
    - Encryption Standards: Advanced Encryption Standard with 256-bit keys delivers high-grade protection against brute-force attacks
    - Dynamic Node Management: VPN configurations dynamically adapt to the addition or removal of nodes, ensuring continuous scalability without disrupting secure channels
    - Performance Optimization: by fine-tuning encryption mechanisms, the VPN minimizes latency and overhead while maintaining strong security
2. Transmission Control Protocol (TCP) communication for control messages (Transport Layer): communication of control messages between the central node and federated nodes is critical for network coordination and workload distribution. This is achieved through secure implementations at the transport layer, with the following mechanisms:
    - Session Security with TLS: TLS encrypts control messages over TCP connections, ensuring end-to-end confidentiality and integrity
    - Message Authentication: Hash-based Message Authentication Code provides an additional layer of trust by verifying the authenticity and integrity of messages
    - Mitigation of Man-in-the-Middle (MITM) Attacks: certificate pinning validates the identities of nodes, defending against MITM threats during communication
    - Reliability via TCP: TCP ensures reliable delivery of control messages and integrates seamlessly with TLS to enhance security

Layer-Specific Security Contributions

- Link Layer (VPN): by securing traffic at the link layer, VPNs ensure that the entire communication channel is encrypted and isolated from unauthorized access, even before transport protocols like TCP come into play. This creates a foundational layer of security that protects against eavesdropping and tampering
- Transport Layer (TCP with TLS): adding TLS to TCP enhances the transport layer by securing individual sessions, encrypting control messages, and ensuring integrity and authenticity. This layer-specific security is essential for real-time coordination in federated networks

Technical Foundations and Risk Management

- Risk Analysis and Threat Modeling: the framework incorporates assessments of risks, including eavesdropping, MITM attacks, and denial-of-service (DoS) threats, tailoring security measures to address these challenges effectively
- Protocol Selection: protocols like IPsec for VPNs and TLS for TCP are chosen for their proven reliability, scalability, and balance of computational overhead with security needs
- Layered Defense: integrating link-layer VPN security with transport-layer TLS ensures a defense-in-depth strategy, providing robust protection across all communication layers
- This multi-layered secure communication framework offers a scalable and adaptable solution for federated networks. By combining the foundational security of VPNs with the advanced session-specific protection of TCP with TLS, the proposed approach effectively addresses the dynamic challenges of distributed systems while maintaining high security standards

The implementation of secure communication at the transport/link layer is a foundational element of federated network security. By combining VPNs for encrypted node-to-node communication and TLS-secured TCP for control message exchanges, our approach addresses the critical security needs of federated environments. This strategy not only mitigates risks effectively but also ensures scalability and adaptability in the face of evolving threats and network dynamics.

3.10. Proposal at application and session level

At the application layer, layer 7 of the OSI model, an authentication and authorization system are essential to ensure that the secure access to resources within a cluster. In a Kubernetes environment, such a system can use tokens to control deployment permissions and resource access. These tokens play a vital role in access control and identity management, ensuring only authorized users or services can interact with the protected resources and the cluster.

Types of security:

1. Authentication and authorization
    - Authentication: authentication is the process by which the identity of the user or system requesting access to Kubernetes resources or cluster is verified. The token can be a unique string to identify the user or service that is willing to interact with the cluster

- Authorization: after the authentication, the same token is used to determine the permissions of the user or service within the cluster and check if it can perform the requested action. For instance, if the user or service can deploy, read or update resources

2. Information Security:
   - The tokens provide an identity-based security method, as they are tied to the identity of the user or service that is interacting with the system
   - Tokens can also be used in a RBAC, (Role-Based Access Control), context. Roles assign permissions based on roles, and these are mapped to job permissions or responsibilities
   - Hybrid solutions can also be implemented
   - Token based authentication can be implemented using mechanisms such as OAuth, bare Jason Web Tokens (JWT) or bearer tokens. In our case, we chose bearer tokens.

Bearer tokens are part of the OAuth 2.0 protocol and are used to simplify the token-based authentication process. They have the following advantages:

- **Simplicity**: Bearer tokens are straightforward to implement and use. The client simply includes the token in the authorization header when making requests, without needing to handle the full OAuth flow
- **Protocol-Agnostic usage:** Bearer tokens can be used outside the full OAuth framework. They are not tied to the entire OAuth flow, making them flexible for standalone authentication systems
- **Focus on Resources Access:** Bearer tokens are designed specifically for accessing protected resources. They eliminate the overhead of managing multiple token types or additional OAuth configurations

OSI model layers:

The use of tokens, either for authentication or authorization, can be found in the following layers of the OSI model.

1. Layer 7 (Application layer)

   Token-based authentication and authorization play a big role in layer 7 of the OSI model. This is because these processes involve applications and services that manage access over the network. Bearer tokens are used as part of the HyperText Transfer Protocol (HTTP) to manage access to resources. They are transmitted as part of the authorization header in HTTP requests, and this allows the application layer to manage user access to resources.

   Kubernetes's own API service verifies whether the user or service has the necessary permissions to perform specific actions on cluster resources, communicating with the client user or service using tokens.

2. Layer 6 (Presentation layer)

If TLS/SSL is used to encode the communications, which is usual in Kubernetes, the bearer token can be a part of the header of the HTTP request. At this layer, TLS/SSL encoding protects the confidentiality of the token and other credentials during the HTTP transmission. Bearer tokens are often used in the JWT format, which is Base64-encoded.

On the receiving end, the resource server decodes the JWT to extract claims and retrieve the authorization and authentication information. If the token is encrypted, the presentation layer handles its decryption before passing to the application for validation. The token's signature is verified using cryptographic techniques to ensure that the token has not been modified or corrupted.

3. Layer 4 (Transport layer)

   Even though bearer tokens themselves do not operate directly in the transport layer, the HTTP requests that contain those tokens are transmitted using transport protocols like TCP, which ensures reliable communications and ensure that the tokens reach their destination intact. While the process of authentication and authorization is primarily relevant to the application layer, it relies on the transport layer to facilitate secure and dependable communication.

HTTP and **HTTPS (HyperText Transfer Protocol Secure)** operate mainly in the **Application layer** of the OSI model.

In this layer, data is processed to be sent from applications (e.g., web browsers, servers, etc.) over the network. HTTP and HTTPS are used so that a client, usually a web browser, and a web server can transmit information to each other.

HTTP is an insecure protocol that sends plain data with no encryption. This means that any data transmitted can be intercepted and read by third parties if the network is compromised. Regarding bearer tokens, sending them using HTTP protocol means that the token and the information within it can be visible to anyone intercepting the traffic. Anyone in possession of the token can impersonate the user or application that owns it and access the resources that the bearer of the token can handle.

HTTPS uses an additional security protocol, usually **SSL/TLS (Secure Sockets Layer / Transport Layer Security),** to cypher communication between the client and the server, providing confidentiality, authentication and data integrity. Using this protocol over HTTP ensures that the bearer token is secure during transmission and cannot be intercepted or read by attackers. This protocol also ensures that the client is communicating with the intended server by using TLS certificates. HTTPS also prevents the modification or corruption of the transmitted data, ensuring the bearer token remains unaltered.

As application protocols, **both HTTP and HTTPS operate in layer 7**. This layer is responsible for directly interacting with the user applications and services.

In the case of **HTTPS, SSL/TLS encryption operates in layer 6**, Presentation layer, which handles encoding, encryption and compression of data before it is transmitted over the network.

HTTPS also involves the TCP (Transmission Control Protocol) in layer 4, Transport layer, to ensure reliable transmission of data.

HTTP has certain advantages over HTTPS in specific scenarios:

- **Speed:** HTTP is faster than HTTPS because it does not involve the overhead of encryption and decryption processes associated with HTTPS. This can be beneficial in situations where speed is critical, and security is less of a concern.
- **Simplicity:** HTTP is simpler to set up and use since it does not require SSL/TLS certificates or their management. This can be an advantage for small-scale applications, testing environments, or non-sensitive data exchanges.
- **Compatibility:** Older systems, browsers, or devices that do not fully support modern TLS standards can still work with HTTP. This can be relevant for legacy systems or restricted environments.
- **Reduced Resource Usage**: Since HTTP does not encrypt data, it consumes fewer computational resources, making it less demanding on servers, especially those with limited processing power.
- **No Certificate Costs**: HTTPS requires obtaining and maintaining SSL/TLS certificates, which can involve costs and administrative efforts. HTTP eliminates this need.

HTTPSs main advantage is the additional security that it offers:

- **Data Encryption:** HTTPS encrypts the communication between the client and the server using protocols like SSL/TLS. This ensures that sensitive data cannot be intercepted or read by third parties during transmission.
- **Data Integrity:** By using HTTPS, data cannot be modified or corrupted during transmission without being detected. This protects against man-in-the-middle (MITM) attacks.
- **Authentication:** HTTPS ensures that users are communicating with the intended website through SSL/TLS certificates, reducing the risk of phishing and fake websites.

The following table highlights the main advantages of using the HTTP and HTTPS protocols.

HTTP	HTTPS
Speed	Data Encryption
Simplicity	Data Integrity
Compatibility	Authentication
Reduced usage resource	

No certification cost	

TABLE 3 ADVANTAGES OF HTTP AND HTTPS

The owner of each cluster can decide if HTTPS or HTTPS and HTTP protocols are available.

3.11. Proposal at governance level

In federated environments, where clusters are interconnected and share resources between different stakeholders, it is crucial to have a personalized policy management system that allows cluster owners to control access, resource allocation, and prioritization in a detailed manner. This management not only ensures the protection of the clusters against unauthorized access but also facilitates the optimization of available resources according to the system's needs and priorities.

Restricting Access to Clusters

One of the main features of personalized policy management is enabling cluster owners to define and control who can access their resources and under what conditions. Through access control policies, the participation of unwanted users is restricted, preventing unauthorized nodes or services from interacting with the cluster. These policies can be based on different criteria, such as user identity, the role they play within the network, or the characteristics of the nodes attempting to access.

Cluster owners have the ability to modify, add, or remove these policies based on security needs or the evolution of the network, ensuring that only trusted actors can access the deployed services and resources.

Limiting and Prioritizing Resource Usage

In addition to access control, owners can efficiently manage resource usage by implementing policies that limit consumption. These policies allow for setting limits on the use of CPU, memory, and other resources, ensuring that the systems are not overloaded with unnecessary workloads.

In situations of high demand, resources are allocated based on the priority assigned to each service or node. For example, emergency priority policies can ensure that the most critical services (such as those related to security or system availability) always have access to the necessary resources, even if it means temporarily suspending lower-priority services.

Security of Deployed Services

The security of the services deployed in the clusters is also an integral part of personalized policy management. Owners can configure security policies that include the use of authentication, authorization, and encryption mechanisms to protect deployed services, preventing unauthorized access and ensuring the confidentiality and integrity of data in transit.

This includes, among other things, service isolation through segmented networks and the application of RBAC policies, which limit interactions between different services according to their authorization level.

Prioritization in Emergency Levels

Finally, one of the most important features of this policy management approach is the ability to manage priority levels in emergency situations. During high load spikes or system failure scenarios, policies allow critical services to receive the necessary resources and attention to maintain their operation, while lower-priority services may be suspended or run with limited resources.

This type of management is crucial to ensure that the system remains operational and secure at all times, even under adverse conditions.

It can be concluded that personalized policy management in federated clusters provides owners with detailed control over access, resource usage, and the security of deployed services. By allowing precise configuration of access and prioritization policies, this approach ensures that resources are used efficiently, security is maintained at high levels, and essential services receive the necessary resources during emergencies, all without compromising system performance or security.

Choice of Customized Policy Management Over Complex Orchestration Solutions

The decision to opt for customized policy management instead of integrating complex orchestration systems like Karmada and Kyverno is based on several operational and technical factors specific to the needs of NTN (Non-Terrestrial Networks) federated networks, where resources are limited and efficiency is crucial.

- Resource Savings and Operational Efficiency

  Using platforms like Karmada and Kyverno can result in high resource consumption, which is not ideal for NTN clusters handling small payment loads. These orchestration solutions require significant processing power to manage policies and cluster federation, which can be excessive in scenarios where resources are limited. By implementing customized policy management, resource usage is optimized by focusing only on the essentials, without the additional overhead imposed by systems like Karmada.

- Flexibility in Managing Small Payment Loads

  For clusters with small payment loads, Kubernetes, along with complex orchestration solutions, may not be the most suitable option, as it may not efficiently support the load or may result in unnecessarily complex management. In such cases, Docker or Docker Swarm offer a lighter, more appropriate solution, enabling more efficient and cost-effective container management. These technologies are ideal for networks where simplicity and lightness are essential to maintain optimal performance without sacrificing flexibility.

- Reduction of Operational Complexity

  Integrating Karmada with its governance system in a federated environment adds excessive complexity for both operators and users of the federation. This additional complexity not only makes daily network operations more difficult but also can create obstacles in resource management and decision-making efficiency. By opting for a customized policy management approach, this complexity is reduced, allowing for more direct and manageable administration without the operational difficulties introduced by systems like Karmada.

In summary, the choice to use customized policy management instead of complex orchestration systems responds to the need for resource optimization, operational flexibility, and simplicity in managing NTN federated networks. This approach not only ensures more efficient use of limited resources but also facilitates network operation without adding unnecessary layers of complexity, offering a more effective and suitable solution for the specific environment of these networks.

4. Extension of the Proof of Concept (PoC)

In federated systems, where multiple entities collaborate and access distributed resources, security at the application layer is essential to preserve the integrity, confidentiality, and availability of information. In this context, securing communications and managing access centrally are key challenges that require robust and efficient solutions.

This PoC will implement a security strategy at the application layer based on the following main components: VPN, an identity and access management system using Keycloak, which includes security token generation.

1. **VPN**: a VPN will be established to create encrypted communication tunnels between federated participants, protecting data transmissions from interception and man-in-the-middle attacks. This will ensure that all communications between federated nodes travel through a secure and private channel, adding a layer of encryption that makes intrusion attempts more difficult.
2. **Authentication and Authorization**: these processes will be centralized using Keycloak, an identity and access management solution. This system will manage users, passwords, and permissions, allowing credential-based authentication (username and password) and access delegation through JWT (JSON Web Tokens). Keycloak will simplify the federated authentication process and provide additional features such as single sign-on and multi-factor authentication.
3. **Token Generation**: Access to the application's resources and services will be controlled through a token generation system, ensuring that each request comes from an authenticated and authorized user. This will enable granular permission management and ensure that each transaction is linked to a verified identity.

This PoC will validate the feasibility of integrating these three components into a coherent security architecture, providing encrypted communications and centralized identity and access management. Performance, scalability, and robustness metrics of the solution will be evaluated, with the aim of creating a foundation for future expansions that can handle a larger number of federated participants and more complex access levels.

Keycloak endpoints:

- Login:
  - Base URL: http://10.6.71.74:30955/ui/
  - METHOD: GET
  - PURPOSE: The URL for the users to login.
  - Once the user gets to this endpoint a login screen in shown:

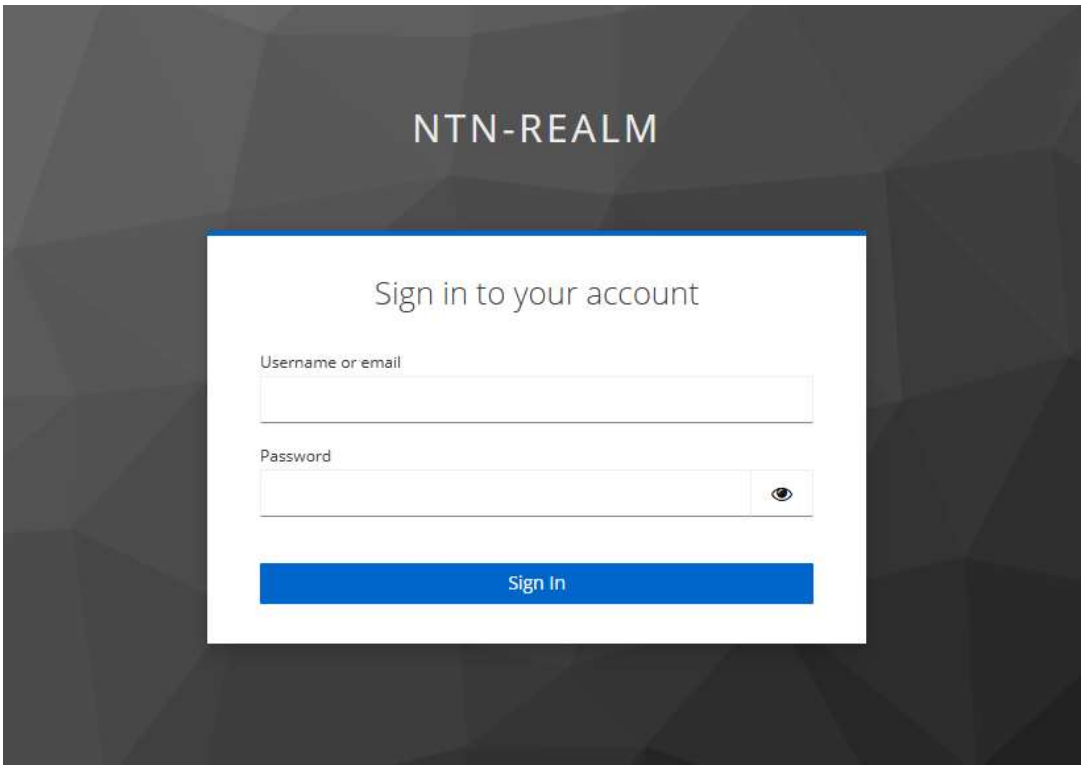

FIGURE 4 NTN FEDERATION LOGIN SCREEN

If the user's credentials are not correct an error is shown in screen and the user does not get logged in:

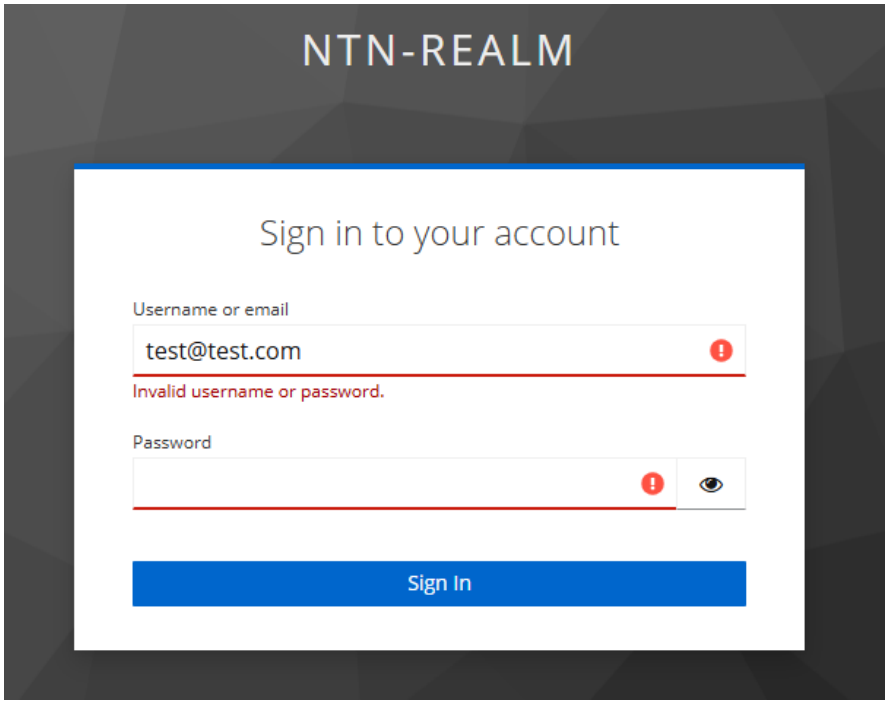

FIGURE 5 LOGIN ERROR MESSAGE SCREEN ON NTN FEDERATION

After the user has filled the fields with the correct data and is logged in, Keycloak redirects the user to the main page. At the same time the session is initiated in Keycloak's administration console.

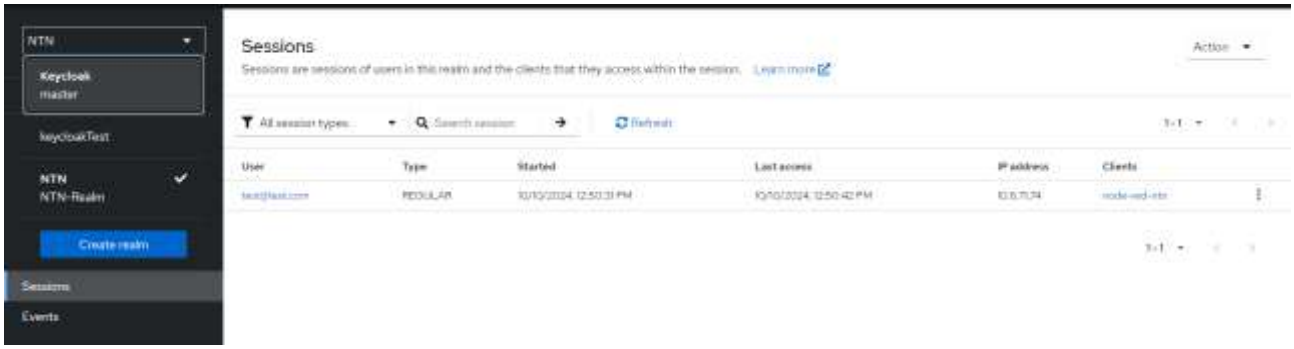

FIGURE 6 KEYCLOAK'S ADMINISTRATION CONSOLE

This indicates that the user is correctly logged in and Keycloak will filter and allow the user to the resources and permissions that the user is able to use.

If the user logs in correctly is directed to the dashboard:

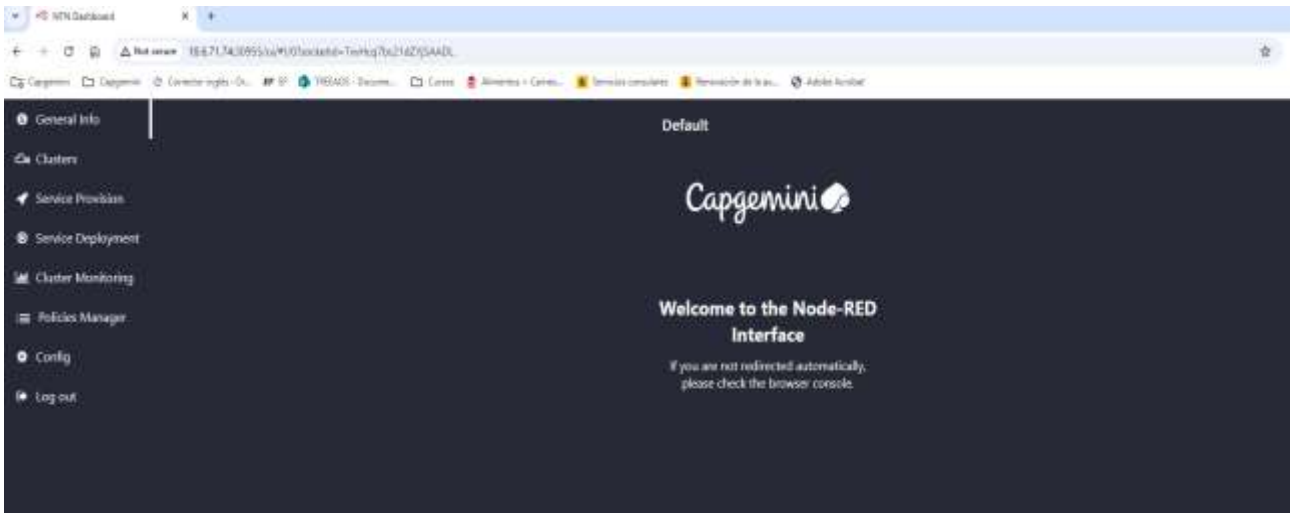

FIGURE 7 NTN DEFERATION DASHBOARD

Also, three items are stored in the browser's sessionStorage:

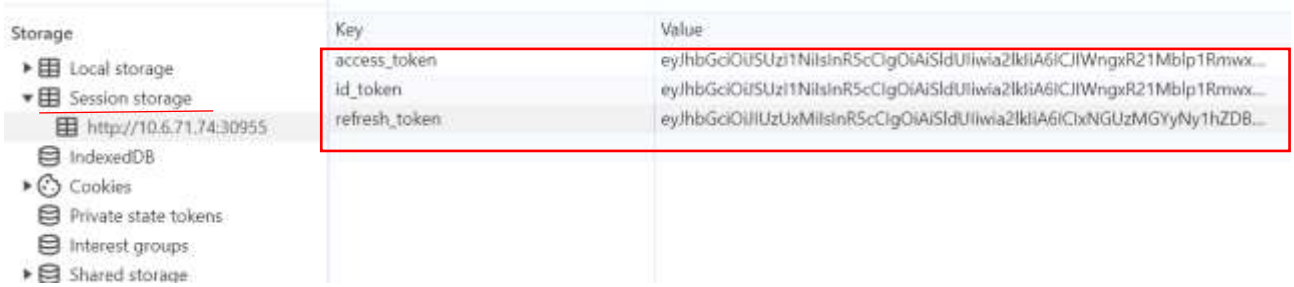

FIGURE 8 ITEMS STORED IN BROWSER SESSION STORAGE

These three items are JWT tokens and are defined and sent by Keycloak and include the user's session, permissions and resources:

These are the decoded tokens:

access_token:
This access token includes personal data (like email and name), user roles and permissions, as well as metadata about when the token was issued and when it will expire. It is used by systems to verify the user's identity and authorization status during API calls or interactions.


```
{exp: 1728557732, iat: 1728557432, auth_time: 1728557431, jti: 'd7712c89-8bfc-49be-bc88-5753ed553626', iss: 'http://10.6.71.74:30904/realms/NTN-Realm', …}
  acr: "1"
  allowed-origins: Array(2)
    0: "http://10.6.71.74:30955"
    1: "http://10.6.71.74:30955/*"
    length: 2
    [[Prototype]]: Array(0)
  aud: "account"
  auth_time: 1728557431
  azp: "node-red-ntn"
  email: "test@test.com"
  email_verified: true
  exp: 1728557732
  family_name: "password:test"
  given_name: "test"
  iat: 1728557432
  iss: "http://10.6.71.74:30904/realms/NTN-Realm"
  jti: "d7712c89-8bfc-49be-bc88-5753ed553626"
  name: "test password:test"
  preferred_username: "test@test.com"
  realm_access:
    roles: (3) ['default-roles-ntn-realm', 'offline_access', 'uma_authorization']
    [[Prototype]]: Object
  resource_access:
    account: {roles: Array(3)}
    [[Prototype]]: Object
  scope: "openid profile email"
  sid: "6ffffe17-e2a0-4f4d-a63f-7f4eba55b1bc"
  sub: "b6f037b9-1a4b-4e37-b400-781d30113d49"
  typ: "Bearer"
  [[Prototype]]: Object
```


These are the following properties of access_token:

- **`acr`**: Authentication Context Class Reference. This typically indicates the level of authentication assurance, "1" being a standard level
- **`allowed-origins`**: An array specifying the origins (URLs) from which this token is valid, such as:

  http://10.6.71.74:30955 ,http://10.6.71.74:30955/*
- **`aud`**: Audience, which is the intended recipient of this token, here specified as `"account"`.
- **`auth_time`**: Authentication time, represented as a Unix timestamp (1728557431), indicating when the user was authenticated

- **azp**: Authorized party, which is the client that the token is issued for—in this case, "node-red-ntn"
- **email**: The user's email address: "test@test.com"
- **email_verified**: Indicates whether the user's email is verified: true.
- **exp**: Expiration time, another Unix timestamp (1728557732), indicating when this token will expire
- **family_name** and **given_name**: User's family and given names, but here it seems to be set as "password:test" and "test"
- **iat**: Issued at time, the Unix timestamp (1728557432) when this token was issued
- **iss**: Issuer, the URL of the Keycloak instance that issued this token: "http://10.6.71.74:30904/realms/NTN-Realm"
- **jti**: JWT ID, a unique identifier for this token ("d7712c89-8bfc-49be-bc88-5753ed553626")
- **name**: The user's full name, formatted as "test password:test"
- **preferred_username**: The user's preferred username, which is their email: "test@test.com"
- **realm_access**: Lists the roles assigned to the user at the realm level:

  default-roles-ntn-realm, offline_access, uma_authorization
- **resource_access**: The user's roles associated with specific resources. Here, for the "account" resource, there is a set of roles
- **scope**: The scopes that the token is authorized for, such as "openid profile email".
- **sid**: Session ID, representing the ID of the session associated with this token: "6ffffe17-e2a0-4f4d-a63f-7f4eba55b1bc"
- **sub**: Subject, which is the user's unique identifier ("b6f037b9-1a4b-4e37-b400-781d30113d49")
- **typ**: The type of token, here indicated as "Bearer"

Id_token:
This token is primarily used by the client (like "node-red-ntn") to confirm the identity of the user after they successfully authenticate. It contains personal information (e.g., name, email) and metadata

(e.g., token expiration, session ID). This token is used in OpenID Connect for authenticating a user and delivering identity information.


```
{exp: 1728557732, iat: 1728557432, auth_time: 1728557431, jti: '2f764ffb-6263-4add-a3a7-9ce593884491', iss: 'http://10.6.71.74:30904/realms/NTN-Realm', …}
  acr: "1"
  at_hash: "NqHSbC3_tkCFweHJ6xLvNg"
  aud: "node-red-ntn"
  auth_time: 1728557431
  azp: "node-red-ntn"
  email: "test@test.com"
  email_verified: true
  exp: 1728557732
  family_name: "password:test"
  given_name: "test"
  iat: 1728557432
  iss: "http://10.6.71.74:30904/realms/NTN-Realm"
  jti: "2f764ffb-6263-4add-a3a7-9ce593884491"
  name: "test password:test"
  preferred_username: "test@test.com"
  sid: "6ffffe17-e2a0-4f4d-a63f-7f4eba55b1bc"
  sub: "b6f037b9-1a4b-4e37-b400-781d30113d49"
  typ: "ID"
▶ [[Prototype]]: Object
```


These are the following properties of id_token:

acr: Authentication Context Class Reference. This indicates the level of assurance for authentication (e.g., `"1"` for basic authentication).

- **at_hash**: Access Token Hash, a hash of the access token. It allows a client to verify that the access token, issued alongside this `id_token`, is legitimate. Here, it is `"NqHSbC3_tkCFweHJ6xLvNg"`
- **aud**: Audience, indicating the intended recipient of the token. In this case, the audience is the client `"node-red-ntn"`
- **auth_time**: Authentication time, represented as a Unix timestamp (`1728557431`), indicating when the user was authenticated
- **azp**: Authorized party, specifying the client for which this token was issued. Here, it is `"node-red-ntn"`
- **email**: The user's email address: "test@test.com"
- **email_verified**: Indicates whether the user's email has been verified. In this case, it is `true`.
- **exp**: Expiration time, which is a Unix timestamp (`1728557732`), indicating when this token will expire
- **family_name** and **given_name**: The user's family and given names. It looks like `"password:test"` is used as the family name and `"test"` as the given name, though this might be placeholder data
- **iat**: Issued at time, a Unix timestamp (1728557432), showing when this token was issued.
- **iss**: Issuer, the URL of the Keycloak server that issued this token. Here, it is "http://10.6.71.74:30904/realms/NTN-Realm", indicating the realm `"NTN-Realm"`

- **jti**: JWT ID, a unique identifier for this token: "2f764ffb-6263-4add-a3a7-9ce593884491"
- **name**: The user's full name, again using "test password:test"
- **preferred_username**: The user's preferred username, which in this case is their email: "test@test.com"
- **sid**: Session ID, indicating the session associated with this token: "6ffffe17-e2a0-4f4d-a63f-7f4eba55b1bc"
- **sub**: Subject, a unique identifier for the user ("b6f037b9-1a4b-4e37-b400-781d30113d49")
- **typ**: Token type, which is "ID", indicating that this is an id_token

Refresh_token:

The refresh token's purpose is to allow the client (in this case, "node-red-ntn") to request new access tokens when the original access token expires, without forcing the user to log in again. It has a longer expiration time than access tokens and helps maintain user sessions in a more seamless manner.


```
{exp: 1728558032, iat: 1728557432, jti: '7bf4a71f-bd01-40ac-90b8-3f1357ff783b', iss: 'http://10.6.71.74:30904/realms/NTN-Realm', aud: 'http://10.6.71.74:30904/realms/NTN-Realm', …}
  aud: "http://10.6.71.74:30904/realms/NTN-Realm"
  azp: "node-red-ntn"
  exp: 1728558032
  iat: 1728557432
  iss: "http://10.6.71.74:30904/realms/NTN-Realm"
  jti: "7bf4a71f-bd01-40ac-90b8-3f1357ff783b"
  scope: "openid basic roles acr profile web-origins email"
  sid: "6ffffe17-e2a0-4f4d-a63f-7f4eba55b1bc"
  sub: "b6f037b9-1a4b-4e37-b400-781d30113d49"
  typ: "Refresh"
  [[Prototype]]: Object
```


These are the following properties of id_token:

- **aud**: Audience, indicating the intended recipient of this token. In this case, it's the Keycloak server: http://10.6.71.74:30904/realms/NTN-Realm
- **azp**: Authorized party, specifying the client for which the token was issued. Here, the client is "node-red-ntn"
- **exp**: Expiration time, represented as a Unix timestamp (1728558032). This indicates when the refresh token will expire and no longer be valid
- **iat**: Issued at time, another Unix timestamp (1728557432), showing when the token was issued
- **iss**: Issuer, which is the URL of the Keycloak server that issued this token: "http://10.6.71.74:30904/realms/NTN-Realm"

- **jti**: JWT ID, a unique identifier for this token: `"7bf4a71f-bd01-40ac-90b8-3f1357ff783b"`
- **scope**: The scopes granted to this token, specifying the permissions and information the client can access. Here, the token includes scopes like:
- `"openid"`: Enables OpenID Connect authentication
- `"basic"`: Basic user info
- `"roles"`: User roles
- `"acr"`: Authentication Context Class Reference
- `"profile"`: Access to the user's profile information
- `"web-origins"`: The allowed web origins for cross-origin requests
- `"email"`: Access to the user's email address
- **sid**: Session ID, representing the session associated with this refresh token: `"6ffffe17-e2a0-4f4d-a63f-7f4eba55b1bc"`
- **sub**: Subject, the unique identifier for the user who owns this token: `"b6f037b9-1a4b-4e37-b400-781d30113d49"`
- **typ**: Token type, here identified as `"Refresh"`, meaning this is a refresh token

Logout

There is a logout button in the main page:

FIGURE 9 LOGOUT BUTTON IN THE MAIN PAGE

Clicking on it triggers a script that sends a POST request to the keycloak logout endpoint at http://10.6.71.74:30904/realms/NTN-Realm/protocol/openid-connect/logout?id_token_hint='+id_token+'&post_logout_redirect_uri=http%3A%2F%2F10.6.71.74%3A30955%2Fui'

and removes access_token, id_token and refresh_token from sessionStorage from the user's browser. Also, the session is deleted from Keycloak. Then the user is redirected to the screen with the login form.

5.1. Analysis of Practical Experience

From a practical perspective:

Keycloak is an open-source identity and access management solution that provides features like single sign-on (SSO), user authentication, authorization, and user management. It supports various protocols like OAuth 2.0, OpenID Connect, and SAML 2.0. In this case it's using OAuth 2.0 protocol. It also offers significant advantages in terms of management and flexibility for very complex or specific authorization managements, but its disadvantages related to learning curve and the need of detailed administration. For small environments, these disadvantages may outweigh the benefits, while in large infrastructures, Keycloak's potential may justify the additional effort required for its implementation and management.

Advantages

- Flexibility: The main advantage of implementing Keycloak derives from its flexibility and the many different options that it provides, making it very customizable and adaptable to various authentication and authorization scenarios. This allows all the members of the federation to tailor and meet their specific needs.
- Fine grain roles and authorization system: Keycloak's fine-grained roles and authorization system provides precise control over user access and permissions, allowing administrators to define specific roles, scopes, and access policies at a granular level. This flexibility enables organizations to tailor permissions to individual users, groups, or applications, ensuring that each user only has access to the necessary resources, minimizing security risks.
- Automation of authorization over resources This feature, known as Keycloak Authorization Services, allows you to automate and manage fine-grained access control to resources without needing to implement custom authorization logic in the federation

  Scalability Keycloak's features make it highly scalable, whether you're serving a small user base or many federated members with thousands of users
- Documentation comprehensive documentation for Keycloak can be found on the official site, which includes everything from getting started to advanced configuration and customization. Since it's a widely used tool across many industries and organizations it has a large base of supporters that use blogs, discussion fora and Keycloak's Github repository that makes it easy to find help and solve possible problems that could arise
- Ubiquity: Keycloak can be used from a web browser with the access to the VPN and the user's email and password credentials, both for users and administrators.

Disadvantages

- Steep Learning Curve: The learning curve for Keycloak is slow due to its complexity and the number of options that it gives to the administrators of the users and the resources, especially for users unfamiliar with identity and access management (IAM) concepts. The complexity can be managed by starting small, using Keycloak's excellent documentation, and taking advantage of community resources.
- Latency: Keycloak is used as a central service for handling authentication and authorization requests so it can introduce some latency in applications, especially if it is not optimized or placed geographically far from the application.
- Customization Complexity: While Keycloak offers extensive customization options, achieving complex custom flows may require custom development. This can be a time-consuming task and may require diving into Keycloak's Service Provider Interfaces to extend its default capabilities.

In summary, the next table shows Keycloak's advantages and disadvantages.

ADVANTAGES	DISADVANTAGES
Flexibility	Steep learning curve
Fine grain roles and authorization system	Latency
Scalability	Customization complexity
Ubiquity	
Automation of authorization over resources	
Documentation	

TABLE 4 KEYCLOAK'S ADVANTAGES AND DISADVANTAGES

5. Conclusions

This work presents a comprehensive proposal and a PoC designed to address the main security challenges in federated systems through an application-layer strategy. This strategy includes the implementation of VPN tunnels to protect communications, the use of Keycloak as a centralized identity and access management system, and security token generation, forming a robust and adaptable architecture that ensures both the integrity and confidentiality of interactions between federation participants.

The integration of Keycloak proves to be an effective and flexible solution for managing detailed roles and permissions, enabling granular access customization and adapting to the specific needs of each federated entity. Although it requires initial setup and administrative effort, this tool provides scalability and robustness, which is especially valuable in large-scale, distributed infrastructures.

The PoC results show that this is a viable and effective security architecture for protecting federated systems and facilitating secure collaboration among participants. This centralized application-layer security approach establishes a solid foundation for future expansions and improvements, enabling scalable adaptation as the number of participants and access requirements within the federation grow.

6. References

[1] "Karmada Documentation," 2023. [Online]. Available: https://karmada.io/docs/.

[2] ISTIO. [Online]. Available: https://istio.io/latest/docs/ops/configuration/traffic-management/tls-configuration/#:~:text=By%20default%2C%20the%20sidecar%20will,configured%20using%20a%20PeerAuthentication%20resource.

[3] ISTIO. [Online]. Available: https://istio.io/latest/docs/tasks/security/authentication/mtls-migration/#:~:text=Istio%20automatically%20configures%20workload%20sidecars,plaintext%20and%20mutual%20TLS%20traffic.

[4] C. Crane, "What Is a Device Certificate? Device Certificates Explained," October 2021. [Online]. Available: https://www.thesslstore.com/blog/what-is-a-device-certificate-device-certificates-explained/#:~:text=A%20device%20certificate%20is%20a,infrastructure%2C%20or%20PKI%20for%20short.

[5] M. Nelson, "How to use PKI to secure the IoT.," 24 July 2016. [Online]. Available: https://www.digicert.com/blog/using-pki-to-secure-the-iot#:~:text=Experts%20recommend%20certificate,device%20authentication.

[6] Kubernetes., "Generate Certificates Manually," [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/certificates/.

[7] Kubernetes, "Manage TLS Certificates in a Cluster," [Online]. Available: https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/.

[8] Kubernetes, "Encrypting Confidential Data at Rest," [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/.

[9] A. Arampatzis, "Encryption and etcd: The key to securing Kubernetes," 20 April 2021. [Online]. Available: https://resources.infosecinstitute.com/topics/cryptography/encryption-and-etcd-the-key-to-securing-kubernetes/.

[10] M. Basan, "What is a VLAN? Ultimate Guide to How VLANs Work," 3 July 2023. [Online]. Available: https://www.esecurityplanet.com/networks/what-is-a-vlan/.

[11] ISTIO. [Online]. Available: https://istio.io/v1.1/docs/concepts/multicluster-deployments/.

[12] A. W. Servicies, "What is IPsec," [Online]. Available: https://aws.amazon.com/es/what-is/ipsec/#:~:text=IPSec%20es%20un%20conjunto%20de,el%20protocolo%20sea%20m%C3%A1s%20seguro.

[13] Sysdig.com, "Building honeypots with vcluster and Falco: Episode I," [Online]. Available: https://sysdig.com/blog/how-to-honeypot-vcluster-falco/.

[14] Wazuh. [Online]. Available: https://documentation.wazuh.com/current/getting-started/index.html.

[15] Kyverno, "Kyverno Policy types," [Online]. Available: https://kyverno.io/policies/?policytypes=Software%2520Supply%2520Chain%2520Security.