Network Policy Enforcement in cloud-native environments

Pedro R. Tomas[1,2][0000−0001−7938−4972], Sofia Silva[2][0009−0008−1162−769X], Marco Neto[2][0000−0002−9534−6279], Jorge Proença[2][0000−0001−9353−0040], Luis Rosa[2][0000−0002−8230−4045], Luis Cordeiro[2][0000−0001−5471−7064], Tarik Taleb[3][0000−0003−1119−1239], and Tiago Cruz[1][0000−0001−9278−6503]

[1] University of Coimbra, DEI, Coimbra, Portugal
{tomas,tjcruz}@dei.uc.pt
[2] OneSource, Coimbra, Portugal
{pedro.tomas,sofia.silva,marco.neto,jorge.proenca,luis.rosa,luis.cordeiro}@onesource.pt
[3] Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum, Germany
tarik.taleb@rub.de

Abstract. The shift towards cloud-native environments has gained significant momentum, and with it, several security and privacy concerns have arisen. One of them is related to the reliable definition and enforcement of network policies in such scenarios. This paper starts by discussing those concerns, reviewing existing technologies and later, introduces a policy orchestrator. Such a proposal addresses the research gap and the notoriously difficult task of ensuring compliance and compatibility with standards. Indeed, the proposed approach supports XACML and JSON-based requests, ensuring interoperability with established standards while also accommodating cloud-native specificities. This paper presents a proof-of-concept of the policy orchestrator in a real-world scenario, demonstrating the usefulness and feasibility of the proposed approach.

Keywords: Cloud-native environments · Policy Enforcement · Policy Orchestrator

1 Introduction

Cloud-native solutions are increasingly adopted over traditional monolithic approaches due to their scalability, elasticity, and adaptability. Nevertheless, ensuring data privacy and security in such complex domains poses several security and privacy challenges, including the correct enforcement of security policies. In this work, we present a novel Policy Orchestrator that aims to ensure a smooth transaction between potential security applications and policy enforcement mechanisms.

The contributions of this work are the following: (i) review of the existing technologies for policy definition and enforcement in cloud-native environments;

(ii) the proposal of an innovative Policy Orchestrator tailored, but not limited, to cloud-native paradigms (including a PoC demonstrating its feasibility).

This paper is divided as follows: Section 3 presents the existing technologies for cloud-native definition and/or enforcement; Section 4 details the Policy Orchestrator, namely by presenting its inputs/outpus and also the PoC develop to validate the feasibility of the proposed approach; Section 5 presents a brief discussion on the topic at hands, in addition to the advantages brought by the proposed approach; Section 6 presents the conclusions of this work.

2 Related Work

Nathaniel et al. [21] investigated the impact of Istio and Nginx API gateways on microservice latency and resource usage in Kubernetes environments. They deployed ExpressJS-based backend services on Kubernetes clusters, integrating Istio for service mesh and Nginx as a traditional ingress controller. Monitoring tools like Prometheus, Grafana, and Kiali were utilized. Load tests via k6 simulated up to 300 requests/sec. Results showed Istio offered lower latency at high request rates (above 175 RPS) but consumed more CPU and RAM compared to Nginx. Their findings recommend Istio for high-traffic scenarios, while Nginx is better suited for lower traffic with minimal resource overhead.

Paul et al. (2024) [25] propose an automated cloud compliance framework integrating Open Policy Agent (OPA) within AWS CI/CD pipelines. They evaluated the usage of AWS CloudFormation templates against predefined security policies using Rego, ensuring policy adherence before deployment. The framework is used to prevent deployments violating compliance policies and preventing misconfigurations. Experimental results demonstrate successful policy validation and automatic remediation, significantly improving the security posture, efficiency, and scalability of cloud infrastructures.

Budigiri et al. [3] analysed Kubernetes network policies' impact on performance and security in multi-tenant 5G edge computing. Focusing on eBPF-based CNI plugins, Calico and Cilium, the authors evaluated latency and throughput using *netperf* on OpenStack-based testbeds. Results show negligible overhead, with Calico outperforming Cilium, particularly for inter-node traffic due to its non-tunneling design. Findings confirm Kubernetes network policies as a scalable, low-overhead solution for secure, low-latency inter-container communication in edge environments.

3 Existing technologies

This section presents some of the existing cloud-native tools for policy definition and enforcement.

3.1 Open Agent Policy

Open Policy Agent (OPA) is an open-source policy engine for defining policies in cloud environments [4]. It separates policy definition from enforcement, providing

access control through detailed policies to ensure proper resource access and prevent unauthorized actions. OPA enables uniform policy definitions across multiple services, reducing the risks of misconfigurations and enforcement issues [5].

OPA introduced Rego, a high-level declarative language designed to define policies, being all the policies used in OPA written in this language. Rego was developed under the concept of 'Policy as Code', aiming to make the policy definition process easier, making it achievable for end-users and not only for network experts (as in different policy definition schemas).

The OPA architecture is designed to be flexible and scalable. An end user or application can interact with OPA via its REST API. Figure 1 presents the general workflow steps. The application initiates the request, which is processed by the OPA engine. OPA evaluates this request against the policies present in their local database and returns a decision.

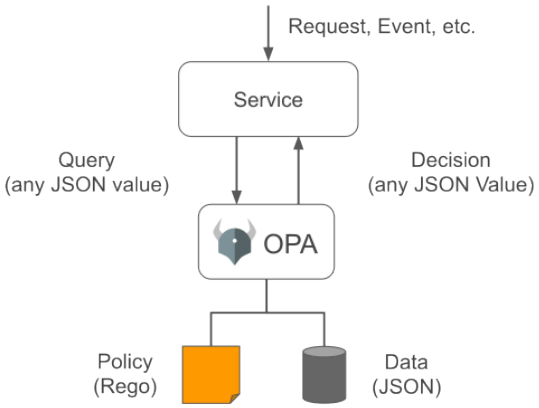

Fig. 1. Open Policy Agent Service Diagram [1].

3.2 Service Mesh

A service mesh is an architectural pattern for connecting distributed microservices, enabling the seamless implementation of service-level features such as the enforcement of network security policies [24] [12]. The main advantage of using a service mesh stems from its ability to effectively manage and optimize traffic flow across various cloud services. The communication between services is end-to-end encrypted, enhancing security across multi-cloud infrastructure [17]. Moreover, an important consideration is that service meshes are language-agnostic, meaning they function independently of the underlying programming language being used by the different components in the scenario [10].

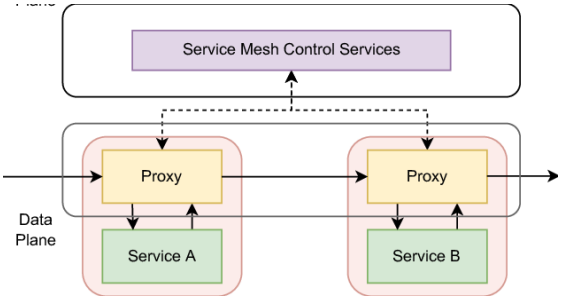

Fig. 2. Service Mesh Architecture.

Envoy proxy is one of the most prominent service mesh implementations. In a service mesh architecture, Envoy is typically deployed as a sidecar proxy with the primary objective of facilitating service-to-service communication within the Kubernetes cluster [12]. The sidecars can be configured to define the specific services that are allowed to communicate with the Envoy proxy. Additionally, configurations can be made to specify the ports and protocols that should be used for communication [18].

3.3 Istio

Istio is an open-source service mesh implementation designed to manage and control traffic within Kubernetes clusters[10]. It provides automatic traffic load balancing and a policy layer for managing access control [9]. Istio is based on the Envoy architecture, meaning it operates as a proxy with the main objective of intercepting and managing all inbound and outbound network traffic between services.

Istio architecture, present in Figure 3, is composed of a Data and a Control plane [8]. The Istio's **control plane** manages the configuration and operation of Istio components, including traffic management, policy enforcement, and observability. It is responsible for communicating and configuring the proxies (sidecars) deployed next to a certain component [8]. The **data plane** consists of a collection of proxies that manage the network communication between components. In addition, these proxies gather and report telemetry data on all traffic within the service mesh. Initially, Istio supported only the *sidecar mode*, yet has recently added support to the *ambient mode*: in **sidecar mode**, the Envoy proxy is deployed and injected next to the application to be protected as a sidecar; in this mode, each proxy is capable of handling both Layer 4 and Layer 7 traffic. In **ambient mode**, a per-node Layer 4 proxy is used, with the option to integrate a per-namespace Envoy proxy to provide additional Layer 7 functionalities [7]. In contrast to a library-based approach, Istio enables a centralized configuration of sidecar proxies that can be reused across different services with minimal or no modifications.

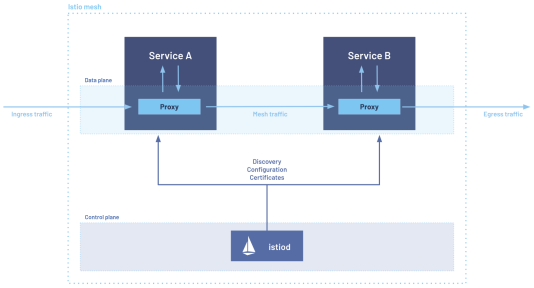

Fig. 3. Istio Architecture Diagram [8].

On the downside, Istio has a substantial learning curve presented by Istio, considering its (initial) complexity. In addition, the use of service mesh introduces an increased demand for computational resources. In some scenarios, its introduction is estimated to nearly double the number of containers, resulting in an increase in computational demand and consequently increased cost, accompanied by a potential decrease in performance [10].

3.4 Calico

Calico is an open-source networking and network security solution for containers owned by *Tigera*. The functionalities provided by Calico are divided into two main strands: the Calico CNI and the Calico network policy suite. The first is a networking engine used in Kubernetes, while the second refers to network policy management capabilities [29].

Calico works in TCP/IP Layer 3, providing support to extensive enterprise-level deployments. Calico offers an easy-to-use interface for defining network policies for Kubernetes objects. It extends traditional network policy features with additional capabilities such as the introduction of specific action rules, including restrictions, permissions, and logging and DNAT traffic flow control [29]. However, the overall complexity of the system may increase [27].

3.5 Cilium

Cilium, similar to Calico, is a cloud-native networking solution and CNI plugin that leverages eBPF to enhance network security and system performance [2]. It offers the option to fully replace kube-proxy with eBPF, making significant advancements in Linux kernel technology. This approach strengthens security, performance and enables the efficient implementation of network and security policies, far surpassing the capabilities of traditional tools [23].

Cilium implements the *Kubernetes NetworkPolicy* to enforce security policies (on layer 4 level) that regulate inter-pod communication. In addition, Cilium provides support for Layer 7, enabling more advanced filtering and policy constructs. By leveraging the use of eBPF, Cilium benefits from its high-performance load

balancing, without depending on the traditional packet proxies, improving latency and enhancing overall performance [11]. One of the significant advantages of this approach lies in its ability to replace kube-proxy, facilitating network observability through Hubble, efficiently managing L3/L4 traffic, and subsequently reducing operational overhead. Ultimately, Cilium combines ease of use, high performance, and strong security, making it a highly effective solution for the networking challenges faced by modern Kubernetes environments [23].

Figure 4 presents the Cilium architecture, which is divided into four components: The **Cilium Agent**, a DaemonSet responsible for configuring eBPF and managing its lifecycle. The **Cilium CNI Plugin** for managing the configuration of network namespaces. The **Hubble** provides observability features for network flow monitoring. The **Cilium Operator** manages cluster-wide operations, including IP address allocation, scaling, and ensuring the high availability of network services.

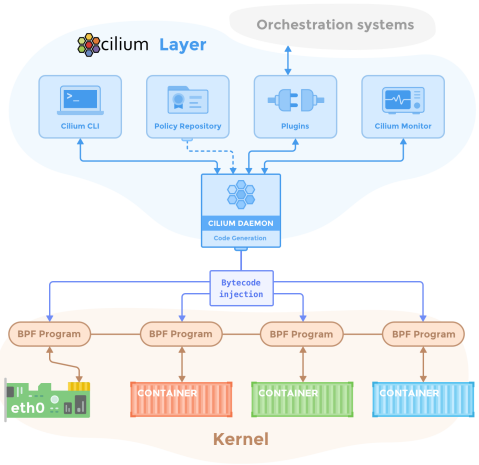

Fig. 4. Cilium architecture [20].

3.6 Kyverno

Kyverno is a cloud-native policy engine for Kubernetes that allows users to define rules to control resources when being deployed in a cluster. It supports policy validation, mutation, generation and cleanup of Kubernetes resources [16]. Kyverno operates as a dynamic admission controller in a Kubernetes cluster. Kyverno handles admission webhook HTTP callbacks from the Kubernetes API server, applying the appropriate policies to either enforce admission policies or deny requests. The Webhook serves as the server responsible for handling Ad-

missionReview requests from the Kubernetes API and forwarding them to the Engine for processing [15].

The Kyverno admission process consists of three essential functional components, which are described below:

1. The **Admission Controller** components are tasked with validating or altering requests as part of the admission control procedure [13].
2. The **AdmissionReview** is a Kubernetes object utilized in the admission control process, encapsulating detailed information regarding the resource request, including the type of request, the identity of the user submitting it, and the contents of the resource itself [14].
3. The **Engine** in Kyverno refers to the core processing unit that evaluates and enforces the policies specified for Kubernetes resources.

Kyverno offers a collection of over 280 pre-built policy templates that can be quickly imported and applied to a Kubernetes cluster. Therefore, saving time and effort for developers by eliminating the need to define them from scratch [6]. On the downside, Kyverno seems to be limited to Kubernetes environments, making it unsuitable for other cloud-native environments. Additionally, being newer than other tools (namely, OPA), may present challenges in terms of maturity and stability [22].

Table 1 presents a brief comparison between the tools previously described.

Table 1: Tools comparison

	OPA	**Istio**	**Calico**	**Cilium**	**Kyverno**
Traffic Management	No	Yes	Yes	Yes	No
CNI	No	No	Yes	Yes	No
Service Mesh	No	Yes	No	No	No
Admission Control	Yes	No	No	No	Yes
Primary Use Case	Policy definition	Service mesh, traffic management	Policy enforcement, CNI	Networking and security with eBPF-based policies	Kubernetes-native policy management

Focus	Policy as code, authorization, and auditing	Microservice communication, observability	Network security, routing, and policies	Advanced networking, security, and observability	Kubernetes resource validation, mutation, and generation
Policy Language	Rego (Declarative policy language)	Custom YAML for policies (Envoy, Istio)	Calico network policies	Cilium network policies	YAML-based policy definitions
Policy Type	Access control, resource management	traffic management, routing, security policies	Network policies	Network policies	Kubernetes policy enforcement
Integration	Kubernetes and other environments	Kubernetes, Istio enabled clusters	Kubernetes and other environments	Cloud-native environments (e.g., Kubernetes)	Kubernetes-native tools, CRDs

Moreover, some of the tools can be combined and used for different scopes. OPA can be used with Istio, separating the policy definition from the policy enforcement, respectively, leveraging the idea of a secure Service Mesh. Calico can be used in standalone Kubernetes environments, leveraging its own CNI to achieve policy enforcement. In a similar form, Cilium can be used in standalone mode, also providing its own CNI implementation, yet with a significant difference in the network routing and management, which is achieved with eBPF entries on each cluster node. Lastly, Kyverno is focused on the policy definition and management aspect, namely by supporting native Kubernetes policies while also providing support for other environments (i.e., CI/CD and/or JSON-based).

4 Proposed Approach

All the aforementioned tools employ different methods for defining policies, which, besides complexity, can pose an interoperability challenge for applications responsible for securing a Kubernetes environment (e.g., NIDS). To mitigate such an issue, this work presents a policy orchestrator that serves both as a Policy Decision Point (PDP) and a bridge between diverse access control systems, ensuring consistent policy enforcement across different technology stacks.

Designed for versatility and scalability, the policy orchestrator supports access control requests in both Extensible Access Control Markup Language (XACML) and JavaScript Object Notation (JSON) formats through a dedicated northbound interface. XACML is supported considering its role within legacy systems that rely on XACML for fine-grained policy control, namely its position

as a standard for network policy definition [19]. In addition, JSON is supported, aiming to provide support for modern microservices and web applications where JSON is used considering its flexibility and simplicity [26].

The Policy Orchestrator supports authentication mechanisms, which ensure that only authorized entities interact with it. External security frameworks or applications must first register with the orchestrator, providing identity and access details. Upon human-based approval, the orchestrator issues a JWT token, serving as proof of authorization. The orchestrator also implements RBAC to manage registered entities and prevent cross-environment tampering. When an entity requests policy enforcement, it must include the JWT token in the Authorization header. The orchestrator validates the token's signature, expiration, and permissions, processing or rejecting the request accordingly.

The southbound interface is designed to be a plug&play component, being prepared to interact with all of the previous mentioned technologies, therefore ensuring a continuous and seamless integration between different systems.

Moreover, the core processing unit of the orchestrator is empowered with AI-based mechanisms designed to improve security and network management. These mechanisms serve two major tasks: (i) intelligent policy recommendations, where the system continuously analyses the current network status, topology, and traffic patterns to suggest optimal access control policies; and (ii) provide usage anomaly detection aiming to detect malicious usages coming from end users via its northbound interface.

4.1 Policy Orchestrator Architecture

Figure 5 presents the policy orchestrator architecture, which comprises several key components that collaborate to ensure seamless policy enforcement across diverse systems.

The first component is the **client or requester**, which represents any system that submits requests to the orchestrator via its northbound interface. These requests can be formatted in either XACML or JSON.

The second component is the **orchestrator** itself, which is composed of: the northbound interface, which is the point of contact between the external security applications and the orchestrator; the AI-based processing unit - whose primary function is to process the incoming requests; and, the southbound interface, which is used to communicate with the different Policy Enforcement Points (PEP) and ensure the correct policy enforcement.

The third component is the **PEP**, responsible for enforcing the policy decisions made by the orchestrator, ensuring that access control rules are consistently applied.

4.2 REST API

The northbound interface of the policy orchestrator exposes a RESTful API that handles incoming requests in both XACML and JSON formats. This API is de-

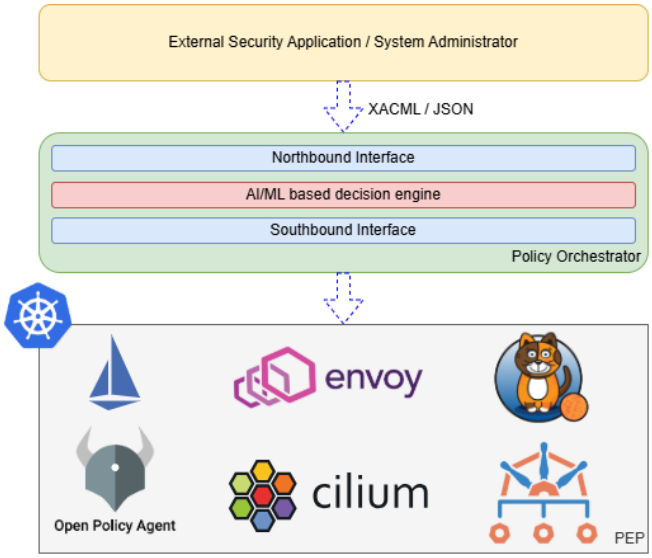

Fig. 5. Policy Orchestrator architecture.

signed to facilitate seamless interaction between clients and the orchestrator.The REST API provides the following endpoints for policy management:

Table 2. API Endpoints for Policy Orchestrator

Method	Endpoint	Description
POST	`/policy/json`	Accepts and processes policy requests in JSON format
POST	`/policy/xacml`	Accepts and processes policy requests in XACML format
GET	`/policy/json/id`	Retrieves the policy specified by ID, if specified, all stored policies, otherwise (in JSON format)
GET	`/policy/xacml/id`	Retrieves the policy specified by ID, if specified, all stored policies, otherwise (in XACML format)

4.3 Proof of Concept

To validate the feasibility of the proposed approach, a scenario was designed to demonstrate the dynamic enforcement of network security policies based on user behaviour, ensuring that unauthorized or malicious users are prevented from accessing critical services in a cloud-native environment.

This scenario replicates a real-world cloud-native system where two users are attempting to access a web page. While performing network traffic analysis, the external security framework detects malicious behaviour from one of the users, who tries to circumvent the web page security using SQL Injection commands.

The sequential diagram present in Figure 6 presents the potential steps taken in such a situation.

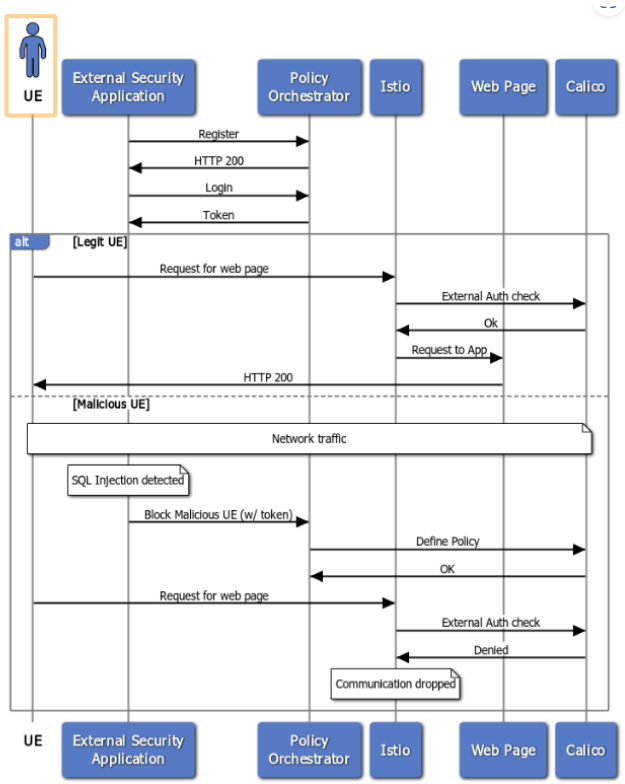

Fig. 6. Workflow of dynamic policy enforcement using the HSPF as an external security framework, the proposed policy orchestrator, Calico for network policy enforcement, and Istio for service mesh traffic control.

In detail, the Holistic Security and Privacy Framework (HSPF) [28] was used to perform the network traffic anomaly detection, acting as the external security framework. Initially, the HSPF registered with the policy orchestrator and obtained a JWT token for authentication. Once registered, the HSPF continuously analysed the network traffic. In this case, a malicious activity was detected, resulting in the HSPF sendinga request to the orchestrator to enforce a blocking policy against the malicious user.

Upon receiving this request, the orchestrator initiated the policy enforcement process by communicating with Calico to define a network policy containing relevant information about the detected threat. Calico then applied the policy, restricting the user's access to the web page and confirming the enforcement to the orchestrator.

Since all traffic is routed through Istio, and Calico was used as an External Authorization Provider for Istio, Istio continually queried Calico to verify if the incoming traffic could be delivered to the desired component or if an action should be taken (for instance, to drop the traffic). When the blocked user tried to reach the web page again, and Istio queried Calico to determine if the traffic should be allowed, Calico informed about the presence of a blocking rule, leading Istio to drop the communication, thus, effectively preventing the user from accessing the target web page.

5 Discussion

Different solutions exist to perform network policy definition and/or enforcement over cloud-native environments (i.e., OPA, Istio, Calico, Cilium, and Kyverno), namely in Kubernetes environments. Some may be used as standalone solutions, while others decouple the policy definition from the policy enforcement, naturally requiring two approaches to perform both parts. Beyond its advantages and disadvantages, all these solutions are useless if a manager or responsible security application is not capable of communicating with them. To such purpose, in this work, a policy orchestrator is presented that aims to facilitate the communication between components identifying the need for the definition of a traffic policy and the tools that actually enable such process.

A key advantage of this approach is its ability to integrate legacy and modern access control mechanisms within a single orchestration layer. While legacy systems rely on XACML for fine-grained access control, modern microservices and cloud-native applications benefit from the simplicity of JSON-based policies. A PoC is provided, aiming to validate the feasibility of the proposed approach, representing a realistic situation in an E2E perspective, that is, from the identification of the attack, going through the definition of the network traffic policy up to the respective policy enforcement and consequent pratical application.

6 Conclusion

This paper explored the challenges of policy enforcement in cloud-native environments, highlighting the characteristics of existing tools for policy definition and/or enforcement. To address the gap between potential security frameworks or system administrators and such tools, this paper introduced a policy orchestrator designed to streamline the policy definition and/or enforcement process.

The proposed policy orchestrator provides a flexible, scalable, and interoperable solution for managing policies in both JSON and XACML formats, ensuring seamless policy enforcement across different tools. Offering a RESTful API, enhanced with an authorization mechanism, simplifies policy administration while enhancing security and compliance in distributed environments.

An immediate step of future work is the continuous development to support all the identified technologies and, as well as the continuous development and fine-tuning of the existent AI/ML algorithms, but also the exploration of other

AI-based mechanisms to support real-time policy evaluation, adaptive decision-making, and policy recommendation systems. Additionally, further performance evaluations are envisioned to assess the impact, feasibility and advantages of the proposed approach.

Acknowledgments. This work was supported in part by the EU's HE research and innovation programme HORIZON-JU-SNS-2022 under the RIGOUROUS project (Grant No. 101095933) and by the EU's European Union's HE Research and Innovation programme HORIZON-CL3-2022-CS-01-01 under the MIRANDA project (Grant No. 101168144). The views expressed in this contribution are those of the authors and do not necessarily represent the project nor the Commission.

References

1. Agent, O.P.: Open policy agent service diagram (2025), https://www.openpolicyagent.org/docs/latest/images/opa-service.svg, accessed: 2025-03-12
2. Budigiri, G., Baumann, C., Muhlberg, J., Truyen, E., Joosen, W.: Network policies in kubernetes: Performance evaluation and security analysis. pp. 407–412 (06 2021). https://doi.org/10.1109/EuCNC/6GSummit51104.2021.9482526
3. Budigiri, G., Baumann, C., Mühlberg, J.T., Truyen, E., Joosen, W.: Network policies in kubernetes: Performance evaluation and security analysis. In: 2021 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit). pp. 407–412 (2021). https://doi.org/10.1109/EuCNC/6GSummit51104.2021.9482526
4. Caracciolo, M.: Policy as Code, How to Automate Cloud Compliance Verification with Open-Source Tools. Ph.D. thesis, Politecnico di Torino (2024), https://webthesis.biblio.polito.it/26908/
5. Devoteam: Take control of your policy enforcement with open policy agent (opa) (2025), https://www.devoteam.com/expert-view/take-control-of-your-policy-enforcement-with-open-policy-agent-opa/, accessed: 2025-03-07
6. Devoteam: Unlocking kubernetes security and compliance with kyverno: A distributed cloud technology to consider (2025), https://www.devoteam.com/expert-view/unlocking-kubernetes-security-and-compliance-with-kyverno-a-distributed-cloud-technology-to-consider/, accessed: 2025-03-11
7. Istio: Data plane modes in istio (2025), https://istio.io/latest/docs/overview/dataplane-modes/, accessed: 2025-03-07
8. Istio: Istio architecture diagram (2025), https://istio.io/latest/docs/ops/deployment/architecture/, accessed: 2025-03-12
9. de Jesus Silva, J.M.: Segurança zero trust para microserviços em sistemas escaláveis (06 2024)
10. Jösch, R.M.: Managing microservices with a service mesh: An implementation of a service mesh with kubernetes and istio (2020), accessed: 2025-03-07
11. Ksolves: Cilium vs. calico: A deeper look into kubernetes networking (2024), https://www.ksolves.com/blog/devops/cilium-vs-calico-a-deeper-look-into-kubernetes-networking, accessed: 2025-03-24
12. Kuikka, S.: Kubernetes networking: Comparative insights into api gateways and service mesh implementations (03 2024)

13. Kyverno: Admission controllers (2023), https://kyverno.io/docs/introduction/admission-controllers/, accessed: 2025-03-24
14. Kyverno: Admissionreview - writing policies with jmespath (2023), https://kyverno.io/docs/writing-policies/jmespath/admissionreview, accessed: 2025-03-24
15. Kyverno: How kyverno works (2023), https://kyverno.io/docs/introduction/how-kyverno-works/, accessed: 2025-03-24
16. Kyverno: Introduction to kyverno (2023), https://kyverno.io/docs/introduction/, accessed: 2025-03-24
17. Luca, C.: Architecture of multi-cloud kubernetes environments (03 2024)
18. Malviya, R., MOHAMMED, N.: Leveraging istio for advanced traffic management and securityin generative ai applications on kubernetes cluster (11 2024). https://doi.org/10.5281/zenodo.14199369
19. Mawla, T., Gupta, M., Sandhu, R.: Specification and enforcement of activity dependency policies using xacml (2024), https://arxiv.org/abs/2403.10092
20. Medium: Diagram illustrating a concept (2025), https://miro.medium.com/v2/resize:fit:720/format:webp/0*K0swBsKe9-Rlivex.png, accessed: 2025-03-12
21. Nathaniel, L., Perdana, G.V., Hadiana, M.R., Negara, R.M., Hertiana, S.N.: Istio api gateway impact to reduce microservice latency and resource usage on kubernetes. In: 2023 International Seminar on Intelligent Technology and Its Applications (ISITIA). pp. 43–47 (2023). https://doi.org/10.1109/ISITIA59021.2023.10221035
22. Nirmata: Kubernetes policy comparison: Kyverno vs opa gatekeeper (2025), https://nirmata.com/2025/02/07/kubernetes-policy-comparison-kyverno-vs-opa-gatekeeper/, accessed: 2025-03-11
23. Oberoi, S.: Cilium: A comprehensive guide to networking, security, and observability in kubernetes (2025), https://medium.com/@simardeep.oberoi/cilium-a-comprehensive-guide-to-networking-security-and-observability-in-kubernetes-41e11fa69d15, accessed: 2025-03-10
24. Patharlagadda, P.P.: Kubernetes traffic management using istio. Journal of Media Management pp. 1–4 (02 2022). https://doi.org/10.47363/JMM/2022(4)E101
25. Paul, A., Manoj, R., S, U.: Amazon web services cloud compliance automation with open policy agent. In: 2024 International Conference on Expert Clouds and Applications (ICOECA). pp. 313–317 (2024). https://doi.org/10.1109/ICOECA62351.2024.00063
26. Peng, D., Cao, L., Xu, W.: Using json for data exchanging in web service applications **7** (12 2015)
27. Team, K.: Comparing kubernetes container network interface (cni) providers (2025), https://kubevious.io/blog/post/comparing-kubernetes-container-network-interface-cni-providers, accessed: 2025-03-10
28. Tomas, P.R., Felix, P., Rosa, L., Gomes, A.S., Cordeiro, L.: A novel approach for continual and federated network anomaly detection. In: Arai, K. (ed.) Proceedings of the Future Technologies Conference (FTC) 2024, Volume 4. pp. 212–225. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-73128-0_14
29. Xgrid: Implementing kubernetes network policies with calico (2025), https://xgrid.medium.com/implementing-kubernetes-network-policies-with-calico-83c37bb822a0, accessed: 2025-03-10