Review of Policy-as-code Approaches to Manage Security and Privacy Conditions in Edge and Cloud Computing Ecosystems

Beniamino Di Martino, Salvatore D'Angelo, Gennaro Junior Pezzullo, Dario Branco, Gerardo Pelosi, Alessandro Barenghi, Simone Orlando

Abstract In this position paper, we present an overview of the most popular policy-as-code frameworks for developing and deploying applications in edge and cloud environments. We compare them in terms of features, flexibility, and integration with existing ecosystems, with a focus on defining policies that address security and privacy specifications for data, communications, and environment configurations. Finally, we describe a case study to demonstrate the practical use of these frameworks and their relevance at different stages of the cloud or cloud-edge application development process.

Beniamino Di Martino
University of Campania "Luigi Vanvitelli", Italy and
University of Vienna, Austria
Asia University, Taiwan e-mail: beniamino.dimartino@unicampania.it

Salvatore D'Angelo
Universitá degli Studi della Campania "L. Vanvitelli", Aversa (CE), Italy, e-mail: salvatore.dangelo@unicampania.it

Gennaro Junior Pezzullo
Universitá degli Studi della Campania "L. Vanvitelli", Aversa (CE), Italy, e-mail: gennarojunior.pezzullo@unicampania.it

Dario Branco
Universitá degli Studi della Campania "L. Vanvitelli", Aversa (CE), Italy, e-mail: dario.branco@unicampania.it

Gerardo Pelosi
Politecnico di Milano, Milano, Italy, e-mail: gerardo.pelosi@polimi.it

Alessandro Barenghi
Politecnico di Milano, Milano, Italy, e-mail: alessandro.barenghi@polimi.it

Simone Orlando
Politecnico di Milano, Milano, Italy, e-mail: simone.orlando@polimi.it

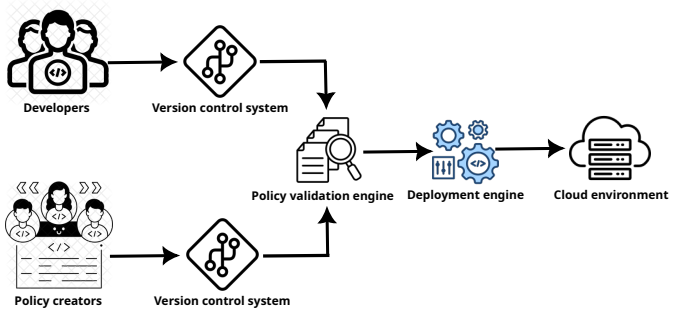

Fig. 1 Workflow for a policy-as-code solution.

1 Introduction

The advent of cloud and cloud-edge computing and the importance acquired by on-demand usage models, such as Software-as-a-Service (SaaS) and Infrastructure-as-a-Service (IaaS) ones, has paved the way for new paradigms in software development and deployment. Indeed, these models have significantly enhanced the scalability and manageability of applications however, they have also introduced new attack surfaces and increased the risks associated with software security vulnerabilities and cybersecurity at large [1, 2].

In this context, ensuring compliance with security requirements has become an increasingly complex task. Historically, rules and guidelines designed to guide users and automated systems within secure boundaries, a.k.a. *policies*, were implemented manually or hard-coded into the application's business logic. This approach made the software development process inefficient and error-prone because strongly pairing the policy and the application logic could limit the re-usability and increase the likelihood that policy-related errors compromise the entire application's functionality.

To address these limitations, one modern approach is to extract policies as separate entities into code artifacts that can be interpreted and executed by tools, called policy engines, designed for compliance verification and acting as reference monitors managing requests. This practice, commonly referred to as policy-as-code (PaC), involves defining and enforcing policy conditions using formal languages to govern operations between system components (see Figure 1). More specifically, policies can be used to specify functional and non-functional aspects of a software system, e.g., infrastructure configurations, security properties, and compliance with standards and regulations. The automation of their application simplifies the process of compliance audits, allowing the guarantees required by industry regulations and internal security standards to be matched more easily. Additionally, since policies are defined in code, they benefit from the available tools for traditional source code management, including version control, testing, and automated deployment. In recent years, the increasing industrial interest in the adoption of PaC solutions [3], has led to the creation of numerous projects that implement this paradigm, many of which are open-source initiatives [4]. These projects provide a rich ecosystem of tools and

frameworks, often integrated with other widely used solutions that are, in turn, employed for application deployment and containerization, enabling organizations to adopt policy-as-code approaches seamlessly.

Contributions. In this study, we provide a comprehensive overview of the most notable frameworks implementing the PaC paradigm, specifically in the context of edge and cloud computing ecosystems. Moreover, we present a careful comparative analysis of these frameworks, along with a use case scenario, focusing on their functionalities, flexibility, and integration with existing deployment ecosystems. Our goal is to help researchers and practitioners to better understand the main features of the available PaC frameworks to the end of selecting the most suitable solutions for secure and efficient policy management.

Structure of the work. The rest of this paper is structured as follows. Section 2, we provide an overview of cloud computing service models and the Policy-as-Code paradigm. Section 3 presents a comparative analysis of existing Policy-as-Code approaches, highlighting their strengths and limitations. In Section 4, we illustrate a specific use case to demonstrate the practical application of Policy-as-Code frameworks in cloud environments. Finally, Section 5 concludes the paper and offers directions for future work.

2 Background

This section provides an overview of the foundational concepts relevant to our work. Specifically, we outline the key cloud computing service models and introduce the Software-as-Code architecture, which underpins modern cloud-native application development and deployment.

Preliminaries on Cloud Computing Service Models. Cloud computing is a computing paradigm that allows access on-demand to a shared and configurable pool of computing resources. These resources can be provisioned and released quickly with minimal interaction with the service provider [5]. Service providers such as Google, Amazon, or Microsoft offer access to these resources and services under a specific business model.

Cloud computing allows companies to operate more efficiently and reduce costs related to the acquisition and maintenance of software and hardware assets by relying on outsourced services and resources that are managed and owned by a third party. This approach gives the ability to timely increase the IT capacity without making significant investments in setting up new infrastructures [6]. The main service models of cloud computing include Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). SaaS allows users to access software applications hosted and provided by the service provider, often as web-based applications available through a browser. PaaS lets users develop, test, and deploy their applications on the provider's cloud infrastructure offering the flexibility to configure the available operating systems and runtime environments to meet their

specific requirements. IaaS provides users with access to computing resources, including processing power, storage, and networking, which are logically partitioned through virtualization technology. Users are responsible for managing everything above the infrastructure layer, including the operating system, application runtime, and data [7]. Despite its numerous benefits, the main hindrance to the adoption of a cloud computing solution with some degree of outsourcing in the ownership, maintenance, and control of data and resources, is represented by security and privacy concerns. Security vulnerabilities may exist in areas such as virtualization, storage, and networking apparatuses, and these may be amplified by security weaknesses in the employed cloud software stack, as well as in the code migrated to the cloud [8].

Preliminaries on Software-as-Code Architectures. In an enterprise, at the organizational level, a security policy denotes a document that delineates the regulations used to safeguard the confidentiality, integrity, and accessibility of the assets. Security policies are established at various levels, ranging from comprehensive frameworks that outline an enterprise's overarching security objectives and principles to detailed documents addressing specific concerns, such as remote access or Wi-Fi usage. In the following, we will tackle system-specific policies that usually consist of both security objectives and operational rules, that might be suitable to be checked and enforced automatically.

Policy-as-code is an approach where policies are externalized using code artifacts written in a specific formal language, known as policy language, instead of being embedded in business code or managed manually. A policy typically includes a "name" that serves as a label for reference, a "purpose" that explains why the policy is necessary, a "situation" defining the context in which it applies, a "set of rules" highlighting specific controls or prescribed behaviors, and "actions"; that are triggered when a rule is violated [4]. In this framework, policy or authorization engines are software programs delegated to interpret policies expressed as code in a dedicated domain-specific language and apply them to relevant data to make decisions [9]. These decisions generate responses that allow the proper enforcement of the policies. The key advantages of this approach lie in the more expressive, flexible, and manageable policies, as well as in more quick and effective compliance checks, which in turn may also be amenable to be automatically generated with formal guarantees of correctness. Furthermore, policies can be treated like source code, enabling the use of tools and practices such as adding comments or leveraging version control systems (VCS) and testing frameworks [10].

In cloud computing, where resources are provisioned and modified rapidly, PaC enables the timely definition of rules directly in workflows and infrastructure configurations, ensuring governance and compliance without manual oversight. For instance, policy-as-code can be used to specify requirements such as ensuring that storage buckets are not publicly accessible, enforcing encryption for all databases, or restricting the deployment of virtual machines to authorized regions. Modern PaC frameworks such as Terraform [11] or AWS CloudFormation [12], automatically validate policy configurations before deployment.

3 Comparative Analysis of Policy-as-code Approaches

In this section, we present the most widely used PaC frameworks. At a high level of abstraction, these frameworks receive queries from a service that needs to decide whether to allow or deny an external request. Then, they evaluate the query by consulting a set of policies and data they have access to and respond to the service with the decision. Finally, the service is responsible for enforcing the received outcome (see Figure 2). As we will see, for some frameworks, the service that communicates with them can be any custom application, while for others, it must be a specific one, such as Kubernetes [13] or Terraform [11].

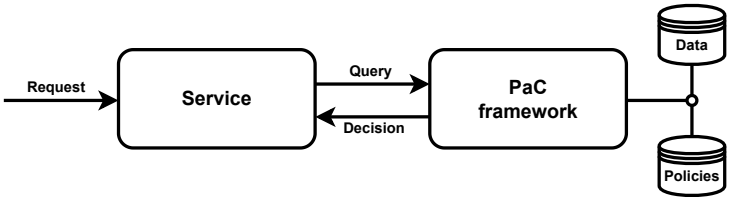

Fig. 2 Standard interaction between a service and a Policy-as-Code (PaC) framework. The service receives a request requiring policy evaluation for acceptance. It queries the PaC framework, which assesses the request by examining applicable policies and available data. The framework then communicates its decision to the service, which is responsible for enforcing it.

In Table 1, we provide a summary and comparison of key aspects of the frameworks discussed. Specifically, we focus on the language used to define policy rules, the different methods available for interacting with the policy engine (e.g., integration into projects through HTTP requests via REST APIs, library function calls, etc.), the availability of command-line tools for interacting with and testing policies, the flexibility and types of input and output, whether the framework can operate as a standalone entity or is tightly coupled with other frameworks (e.g., Terraform [11], Kubernetes [13]), and whether the project is proprietary or open-source.

Open Policy Agent (OPA). It is a domain-agnostic PaC framework created by Styra [14] and classified as a stable project with a graduated maturity level by the Cloud Native Computing Foundation [15]. The domain-agnostic nature of OPA means it does not include built-in binding policies, instead, developers have the flexibility to define custom policies that are applied to generic input requests expressed using the JSON format. Furthermore, the response generated by the OPA engine is not restricted to Boolean values, but it could be any developer-defined JSON value.

OPA's versatility allows it to be integrated into any project, as it is not tied to specific frameworks but serves as a generic engine for defining and evaluating any type of policy. Notably, as reported in the official documentation, it supports various methods for applying policies. It can be run in server mode, enabling interaction via REST APIs through HTTP requests. Additionally, it provides APIs and an SDK for the Go programming language, allowing query evaluation outputs to be retrieved

Table 1 Comparison of Policy-as-Code frameworks and their characteristics.

	OPA	**Gatekeeper**	**Sentinel**	**CrossGuard**	**Kyverno**	**Cedar**
Language (Type)	Rego (2)	Rego (2), YAML (2)	Sentinel (2)	Python (0), TypeScript (0), Rego (2) (Preview)	YAML (2)	Cedar (2)
Integration	REST API, Wasm, Go	Kubernetes webhooks	Consul, Nomad, Terraform, Vault	Pulumi	Kubernetes, Rest API, Go	Rust, Go, Cedar, Java, Wasm
CLI Tool	Yes	Yes	Yes	Yes	Yes	Yes
Input Type	JSON	JSON	HCL	Python, Typescript, Go, C#, YAML	YAML, JSON	Cedar, JSON
Output Type	JSON	JSON	Allow/Deny	Allow/Deny	YAML, JSON	Allow/Deny
Domain Agnostic	Yes	No	No	No	Yes (Extension)	Yes
Standalone	Yes	No	Yes	No	Yes	No
Open Source	Yes	Yes	No	Yes	Yes	Yes

as native Go types. Finally, it offers a mechanism to compile policies into the WebAssembly (Wasm) binary instructions for stack-based virtual machines and evaluate them in the Wasm runtime [16].

Policies are written using the Rego declarative language. It is a domain-specific language (DSL) for work with structured data inspired by Datalog and designed by the same authors of OPA. According to its grammar representation, expressed in the official documentation in extended Backus-Naur form (EBNF), it can be classified as a context-free language (Type-2) [17] thus, decidability is not an issue and it can be always successfully and efficiently parsed with only a push-down automaton. It is possible to load policies into the engine either remotely via APIs or locally using bundles, which are used to package data and policies together. OPA also supports digitally signing these bundles using public-key cryptography techniques, ensuring the authentication and integrity of policy updates. OPA includes a powerful command-line tool that offers a wide range of commands for benchmarking, prototyping, managing, and testing policies. It also incorporates a REPL interactive shell that facilitates and accelerates the development and testing phases, as well as running quick experiments.

Gatekeeper. It is a Webhook implementing the logic of an admission controller to decouple policy decisions from the Kubernetes' API Server [18]. It is internally

based on Open Policy Agent to evaluate policies, but it does not directly expose OPA functionalities to the end user. In this scenario, the place of the service shown in Figure 2 is occupied by Kubernetes [13], and the query-response communication is handled by Kubernetes admission webhooks HTTP callbacks.

Every resource creation, update, or deletion request sent to the Kubernetes API is intercepted by Gatekeeper and validated according to the defined policies. Gatekeeper policies are characterized using two Kubernetes resources: ConstraintTemplates and Constraints, both expressed in YAML files. ConstraintTemplates define the structure and logic of policies, where the Rego code is written, and allow developers to define new constraints. On the other hand, Constraints use ConstraintTemplates to enforce policies on specific resources. In addition, Gatekeeper offers an extensible community library of policies, including both validation and mutation policies. The former is useful to validate cluster resources, while the latter can mutate resources within the cluster. Finally, in order to simplify policy evaluation, Gatekeeper provides developers with a command-line tool called Gator, which supports both manual and automated policy testing.

Hashicorp Sentinel. It is a language and framework for policies built to be embedded in existing software to enable fine-grained, logic-based policy decisions. Sentinel is an enterprise-only feature of HashiCorp Consul, Nomad, Terraform, and Vault. Its policies are written in Sentinel Language, a JSON-compatible language that is easy to read and write, and that can be classified as context-free (Type-2). The Sentinel engine, which is embedded in the HashiCorp products, is used to validate policies. It evaluates policies against the input data and returns a boolean value, which is used to determine whether the policy passes or fails.

CrossGuard. It is a policy-as-code framework developed by the cloud engineering platform Pulumi. It is designed to provide a unified policy enforcement mechanism across multiple cloud providers. CrossGuard policies can be written in Python (Type-1 language, decidability not an issue, parsed with a Turing Machine) or TypeScript (Type-0 language, decidability is an issue, a Turing Machine might be used), and are evaluated by the Pulumi engine. Preview support for the Rego language is also available. The command-line tool used to evaluate the policies is the Pulumi CLI so, CrossGuard policies are evaluated against the Pulumi program, which is a declarative description of the cloud infrastructure. The policies are validated at the time of deployment and can be used to enforce constraints on the infrastructure, such as resource limits, naming conventions, and security settings.

Kyverno. It is an open-source framework explicitly designed for Kubernetes. Policies in Kyverno are managed as native Kubernetes resources and written in YAML. In particular, it can validate, mutate, generate, and clean up any Kubernetes resource. In addition, there is also support for JMESPath and the Common Expressions Language (CEL) to handle articulated logic easily and efficiently. It can be considered domain agnostic thanks to a sub-project called "Kyverno JSON" which adapted the main project to allow its policies to be applied on JSON payloads outside Kubernetes environments, thus extending its usability. YAML is chosen as the policy language to eliminate the need for the developer to learn a new language. It has a proprietary

command-line tool and a web-based user interface, called Kyverno Policy Reporter, that provides report management.

Cedar. It is a flexible, extensible, scalable policy-based access control language developed by AWS. Cedar supports standard authorization models such as role-based access control (RBAC) and attribute-based access control (ABAC). When a user attempts to act on a resource, the application requests authorization from the Cedar policy engine, which evaluates the applicable policies and returns an allow or deny decision. Cedar is designed to be integrated into custom applications, allowing developers to outsource authorization logic from the application code. In addition, AWS has made Cedar open-source, allowing the community to contribute to its development and adapt it to different application needs.

4 Use Case: E-Health Cloud Edge Architecture

The proposed case study is an architecture distributed on three levels, dropped into an e-Health scenario. The goal is to support doctors and patients in securely acquiring, processing, and visualizing health data. The design choice of dividing the entire system into three levels (see Figure 3) stems from the need to strategically distribute functions and workloads and, at the same time, ensure personalized access to information [19].

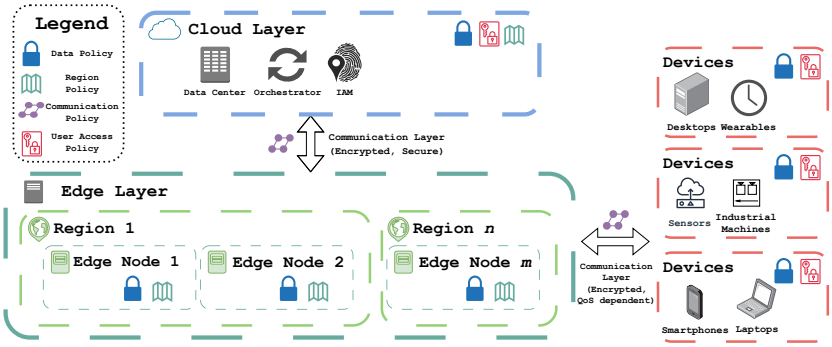

Fig. 3 e-Helth Cloud-edge Architecture.

The first level, the device level, is dedicated to real-time data collection through wearable devices and, more generally, sensors for monitoring vital parameters or environmental conditions. Deciding to decentralize data collection makes continuous monitoring possible even in remote contexts not directly connected to the infrastructure. The edge layer, on the other hand, is critical for the aggregation and pre-processing of collected data. If the number of devices is large enough, there is no guarantee that they are all from the same provider. As a result, heterogeneity of

devices can lead to different formatting or data capture for each IoT component. In this context, the edge layer unifies the data before sending it to the cloud. Here, in the last layer, learning algorithms are processed for predictive analysis and visualization of results. Resource management in the cloud is also essential to manage differentiated access to data (e.g., patients can view only their personal information, while physicians access more complex data about the patients under their care). Encryption mechanisms protect the transmission of data between the different layers. This approach is essential to ensure that patient information remains hidden during transfer and processing, thus guaranteeing the minimization of privacy compromises. From an implementation perspective, on the other hand, this architecture lends itself particularly well to being developed via microservices using an orchestrator such as Kubernetes or Nomad. These tools are essential since they dynamically allow modular system management to add new edge nodes to the cluster [20]. Based on the different levels and the different roles of the actors, we can define different policies to ensure the security and privacy of the data. In particular, we can define the following policies:

- **User Access Policy**: at the user access level, it is possible to define an access control policy for data shared between doctor and patient. In particular, the doctor must have access to patients' data, while patients should have access only to their own data. To enforce this policy, some of the frameworks presented can be used, particularly those that allow policy verification through a REST endpoint, as access occurs from web or mobile applications where the user is already authenticated and accesses shared resources. Therefore, we can use OPA, which can be run as a host daemon or sidecar container to verify the policy through a REST endpoint; Kyverno, specifically the KyvernoJSON subproject, executed as a web application with REST API for policy verification; finally, Cedar-Agent, an HTTP server for managing and verifying policies based on HTTP requests.
- **Data Policy**: for storing sensitive data, such as medical ones, encryption must be enforced. Therefore, we can define a policy where every stateful microservice must have encrypted storage. The verification of this policy occurs when deploying the various microservices for the machine learning model calculation on the various edge nodes, so the control must be carried out during the deployment phase. To implement this policy, if we have a Kubernetes cluster available, we can use Gatekeeper, Kyverno, and Pulumi; if the cluster is managed with Nomad, we can use Sentinel. With all these tools, we can define a policy that verifies that every deployed stateful microservice has associated encrypted storage.
- **Communication Policy**: even in the case of communication between microservices, it is important to ensure data security. In particular, data must be encrypted during communication between the different levels of the application. Since the microservices are distributed across edge and cloud nodes and managed with a microservice orchestrator, such as Kubernetes or Nomad, we can define a policy that verifies that every communication between microservices occurs through secure communication protocols, such as HTTPS. To implement this policy, we can use OPA, Gatekeeper, Kyverno, Pulumi, and Sentinel, as all these tools allow

for defining policies for communication between different microservices, with the only difference being that Sentinel supports only Nomad clusters.

- **Region Policy**: to ensure compliance with privacy regulations, such as GDPR, it is important that patient data is stored in specific geographic regions. In particular, patient data must be stored in European geographic regions. The application of the policy occurs during the deployment phase of the microservices. To implement this policy, we can use OPA, Gatekeeper, Kyverno, Pulumi, and Sentinel, as all these tools allow defining policies to enforce deployment in specific regions of the microservices. In scenarios where edge nodes are added and removed dynamically, we can also define policies for the dynamic addition and removal of edge nodes in a Kubernetes or Nomad cluster depending on the geographic location.

5 Concluding Remarks

In this work, we have presented a comprehensive review of the principal Policy-as-Code frameworks available for privacy and security management in cloud-edge environments. The adoption of Policy-as-Code has several advantages, such as the possibility to automate policy enforcement and compliance verification, reducing human errors. The integration with orchestration tools, such as Kubernetes and Nomad, allow to ensure the coherence of the infrastructure. Moreover, it simplifies the monitoring and auditing of policies, allowing the organizations to be compliant with the regulations and the internal security standards. The analysis of the frameworks has shown that every solution has its strengths and limitations. Some of them are more flexible but require the knowledge of a specific language, others are more user-friendly but have limitations in terms of flexibility or independence from the tools used for the deployment. Despite the benefits, the adoption of Policy-as-Code has some challenges, like the need for specific languages, the complexity of multi-cloud and multi-framework, and the difficulty of integration into legacy systems.

For future work, we are working on a methodology and a tool to support the code decomposition, parallelization, and cloud-edge deployment making direct use of code annotation, primitives, and skeleton [21, 22]. We plan to extend this methodology and tool to support the Policy-as-Code frameworks, allowing the developers to associate a policy to code blocks realizing the business-logic, during the code decomposition, and enabling the automatic enforcement of these policies during deployment.

Acknowledgements This work was supported in part by project SERICS (PE00000014) under the Italian NRRP MUR program funded by the EU - NGEU.

References

1. P. K. Chouhan, F. Yao, and S. Sezer, "Software as a service: Understanding security issues," in *2015 Science and Information Conference (SAI)*, 2015, pp. 162–170.
2. F. K. Parast, C. Sindhav, S. Nikam, H. I. Yekta, K. B. Kent, and S. Hakak, "Cloud computing security: A survey of service-based models," *Comput. Secur.*, vol. 114, p. 102580, 2022.
3. Styra, "The State of Policy as Code Report: Insights as Organizations Scale Authorization Policies," https://www.styra.com/resources/reports/policy-as-code/, accessed: 2024-12-11.
4. J. Ray, *Policy as Code*. O'Reilly Media, Inc., 2024. [Online]. Available: https://www.oreilly.com/library/view/policy-as-code/9781098139179/
5. P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Special Publication 800-145, 2011. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-145
6. M.-G. Avram, "Advantages and challenges of adopting cloud computing from an enterprise perspective," *Procedia Technology*, vol. 12, pp. 529–534, 2014.
7. A. Rashid and A. Chaturvedi, "Cloud computing characteristics and services: a brief review," *International Journal of Computer Sciences and Engineering*, vol. 7, no. 2, pp. 421–426, 2019.
8. S. Sengupta, V. S. Kaulgud, and V. S. Sharma, "Cloud Computing Security-Trends and Research Directions," in *World Congress on Services, SERVICES 2011, Washington, DC, USA, July 4-9, 2011*. IEEE Computer Society, 2011, pp. 524–531.
9. J. W. Cutler, C. Disselkoen, A. Eline, S. He, K. Headley, M. Hicks, K. Hietala, E. Ioannidis, J. H. Kastner, A. Mamat, D. McAdams, M. McCutchen, N. Rungta, E. Torlak, and A. Wells, "Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization (Extended Version)," *CoRR*, vol. abs/2403.04651, 2024.
10. Styra, *What is Policy as Code? Definition and Benefits*, November 2022, accessed: 2024-12-16. [Online]. Available: https://www.styra.com/blog/what-is-policy-as-code-definition-and-benefits/
11. *Terraform*, HashiCorp, 2024, accessed: 2024-12-16. [Online]. Available: https://www.terraform.io/
12. *AWS CloudFormation*, Amazon Web Services, 2024, accessed: 2024-12-16. [Online]. Available: https://aws.amazon.com/cloudformation/
13. *Kubernetes*, Cloud Native Computing Foundation, accessed: 2024-01-24. [Online]. Available: https://kubernetes.io/
14. Styra, *Open Policy Agent*, accessed: 2025-01-17. [Online]. Available: https://www.openpolicyagent.org/
15. Cloud Native Computing Foundation, *Graduated and Incubating Projects*, accessed: 2025-01-17. [Online]. Available: https://www.cncf.io/projects/
16. Styra, *Integrating OPA*, accessed: 2025-01-17. [Online]. Available: https://www.openpolicyagent.org/docs/latest/integration/
17. ——, *Rego Grammar*, accessed: 2025-01-17. [Online]. Available: https://www.openpolicyagent.org/docs/latest/policy-reference/
18. OPA, *Gatekeeper - Policy Controller for Kubernetes*, accessed: 2025-01-23. [Online]. Available: https://open-policy-agent.github.io/gatekeeper/website/docs/
19. B. Di Martino, G. J. Pezzullo, V. Bombace, L.-H. Li, and K.-C. Li, "On exploiting and implementing collaborative virtual and augmented reality in a cloud continuum scenario," *Future Internet*, vol. 16, no. 11, p. 393, 2024.
20. G. J. Pezzullo, B. Di Martino, and M. Bubak, "Container-based platform for computational medicine," in *International Conference on Advanced Information Networking and Applications*. Springer, 2022, pp. 131–140.
21. B. Di Martino, S. Venticinque, A. Esposito, and S. D'Angelo, "A methodology based on computational patterns for offloading of big data applications on cloud-edge platforms," *Future Internet*, vol. 12, no. 2, 2020.
22. B. Di Martino, A. Esposito, S. D'Angelo, S. A. Maisto, and S. Nacchia, "A compiler for agnostic programming and deployment of big data analytics on multiple platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, p. 1920 – 1931, 2020.