A Comparative Study of Authorisation Mechanisms in Kubernetes-Based Service Platforms

ACIG
APPLIED CYBERSECURITY & INTERNET GOVERNANCE

NASK

A Comparative Study of Authorisation Mechanisms in Kubernetes-Based Service Platforms

Francesco Pizzato | Department of Control and Computer Engineering, Politecnico di Torino, Italy | ORCID: 0000-0002-9937-1367

Daniele Bringhenti | Department of Control and Computer Engineering, Politecnico di Torino, Italy | ORCID: 0000-0002-3086-7364

Fulvio Valenza | Department of Control and Computer Engineering, Politecnico di Torino, Italy | ORCID: 0000-0002-8471-3029

Abstract

Kubernetes has become a core substrate for digital service platforms, where multiple teams, tenants, and automation components share the same control plane. In this setting, authorisation is a central security control because it governs API operations that can expose sensitive data, change runtime behaviour, or disrupt availability. Enforcing least privilege in Kubernetes is challenging in practice: the authorisation surface is broad, policies evolve continuously, and automation identities frequently act with privileges that can amplify the impact of misconfiguration or compromise. This paper compares Kubernetes authorisation mechanisms, covering native options (Role-Based Access Control [RBAC], Attribute-Based Access Control [ABAC], and Authorization Webhooks) together with representative open-source approaches that enable more expressive models, namely Open Policy Agent (OPA) and SpiceDB. The analysis is grounded in operational requirements typical of shared clusters, including delegated administration, constrained access to sensitive resources, least-privilege automation, and controlled

Received: 23.02.2026

Accepted: 09.04.2026

Published: 30.05.2026

Cite this article as:
F. Pizzato, D. Bringhenti, F. Valenza, "A comparative study of authorization mechanisms in Kubernetes-based service platforms," ACIG, vol. 5, no. 2, doi: 10.60097/ACIG/220492.

Corresponding author:
Daniele Bringhenti, Department of Control and Computer Engineering, Politecnico di Torino, Italy. Email: daniele.bringhenti@polito.it
0000-0002-3086-7364

administrative operations. Mechanisms are evaluated through a unified framework that captures both security and operational consequences along four dimensions: complexity, granularity, scalability, and performance. The results show that no mechanism dominates across all dimensions. RBAC remains an effective baseline due to tight integration and low-latency enforcement, but it can be difficult to extend to contextual constraints without policy sprawl. ABAC supports conditional rules but is often penalised by operational workflows that make policy evolution costly. Webhook authorisation is flexible but introduces a security-critical external dependency on the API request path. More expressive approaches, such as OPA and SpiceDB, are justified when they replace brittle approximations and support disciplined policy lifecycle management or relationship-driven permissions at scale.

Keywords
access control, Kubernetes, authorisation, multi-tenancy

1. Introduction

Kubernetes has become one of the most widely used orchestration platforms for modern digital services, often supporting not just single applications but entire service chains composed of microservices, shared platform components, and automation agents. In this ecosystem, the Kubernetes API server mediates nearly every action: applying manifests, updating workloads, or reconciling cluster state all pass through it. Authorisation, therefore, is not a peripheral configuration detail but a central control point that determines whether the platform operates safely and predictably. In multi-tenant environments, overly broad permissions can have consequences far beyond their intended scope. Routine operational tasks, such as debugging, patching deployments, retrieving logs, or responding to incidents, can inadvertently expose cross-names pace data, enable privilege escalation, or disrupt other teams' services. Because Kubernetes roles can span multiple API groups and resources, a seemingly narrow permission may grant far-reaching capabilities, turning authorisation errors into latent vulnerabilities that often surface under pressure.

At the same time, excessively restrictive or difficult-to-manage authorisation models fail in a different way. When access control becomes a persistent source of friction, teams tend to compensate by inflating permissions, sharing long-lived credentials, or relying on informal exceptions. These behaviours are rarely malicious; they

are typically driven by deadlines, incident response pressure, or unclear ownership boundaries. Nevertheless, the effect is the gradual erosion of least privilege, reduced audit clarity, and a widening gap between formal governance and actual operational practice. In a platform designed to enable rapid iteration, authorisation must therefore strike a careful balance: it must be protective enough to reduce risk, yet usable enough to avoid pushing teams towards unsafe workarounds.

Kubernetes amplifies authorisation challenges because its surface is broad, dynamic, and tightly intertwined with automation. It exposes a wide range of resource types – from workloads and networking primitives to sensitive assets like Secrets and cluster-scoped components that can impact all tenants. Although namespaces are commonly used as units of ownership and multi-tenancy, they do not guarantee strong isolation, and many high-impact operations remain cluster-wide [1]. Moreover, Kubernetes relies heavily on controllers and operators that act continuously through service accounts, which function as privileged, first-class identities. As a result, an effective authorisation strategy must address both human and machine actors, enforce least privilege for automation, define clear delegation boundaries between platform and application teams, and evolve alongside a system in which resources are constantly created and modified.

These characteristics complicate the choice of authorisation mechanisms. Kubernetes provides several native approaches with different trade-offs in expressiveness and operational behaviour. RBAC, the default and most widely adopted model, binds identities to permissions over resources and verbs; it is tightly integrated with the API server and auditing pipeline but limited in expressing context-dependent or content-aware policies. Attribute-Based Access Control (ABAC) enables more conditional logic based on request attributes, yet is often considered cumbersome due to its static configuration model. Authorization Webhooks allow decisions to be delegated to external services, offering maximum flexibility at the cost of additional dependencies and operational complexity. Beyond these built-in mechanisms, ecosystem extensions attempt to address reviewability, organisational modelling, and policy evolution in large or compliance-sensitive clusters. However, adopting such solutions introduces architectural and operational overhead, requiring practitioners to carefully weigh benefits, costs, and scalability trade-offs.

This paper focuses on a practical question: how can Kubernetes authorisation mechanisms be selected and operated to provide

fine-grained control in multi-tenant digital service systems without introducing unsustainable complexity or unacceptable overhead? We approach this question by comparing a set of mechanisms commonly encountered in Kubernetes environments, spanning native options and representative open-source extensions. The comparison is intentionally grounded in realistic operational scenarios. The scenarios reflect common needs in shared clusters, such as delegation within and across namespaces, constrained access to sensitive resources, least-privilege automation, and controlled administrative actions. By using scenarios, the analysis stays close to the way authorisation is actually experienced by platform engineers, security teams, and service owners.

From this perspective, the paper addresses three research questions.

1. The first concerns expressiveness: which mechanisms can naturally represent the fine-grained constraints that arise in practice, including contextual requirements and separation of duties.
2. The second concerns operational viability: what costs each mechanism introduces in terms of policy authoring, deployment and update workflows, auditing, and failure handling.
3. The third concerns system-level trade-offs: what happens when these mechanisms are compared along dimensions that matter in real infrastructures, namely complexity, granularity, scalability, and performance. Complexity captures the cognitive and operational overhead of adopting and maintaining a mechanism. Granularity captures how precisely permissions can be expressed without resorting to broad approximations. Scalability captures how well the mechanism supports growth in identities, services, and tenants. Performance captures decision latency and the likelihood that authorisation becomes a bottleneck.

Consequently, the contribution of this work is three-fold. First, it provides a unified comparison of multiple Kubernetes authorisation mechanisms under a single conceptual lens, emphasising the practical differences that appear when policies are authored, deployed, and maintained over time. Second, it introduces a scenario-driven evaluation approach that connects multi-tenant requirements to concrete authorisation decisions, ensuring that the comparison reflects operational reality, rather than idealised examples. Third, it synthesises the findings into a comparative evaluation matrix and a set of design guidelines aimed at helping platform engineers

and security practitioners select, combine, and evolve authorisation controls as Kubernetes-based service systems grow.

The remainder of this paper is organised as follows: Section 2 presents background about access control paradigms for Kubernetes. Section 3 discusses prior research work on access control and cloud security configuration. Section 4 describes the scenario-driven methodology used to compare the mechanisms. Section 5 presents the comparative analysis across native and open-source approaches and synthesises the main trade-offs. Section 6 discusses implications for digital service systems and provides practical design guidelines for selecting and composing authorisation mechanisms. Section 7 concludes the paper and outlines future work.

2. Background

Kubernetes is an orchestration platform in which the desired state of services is represented as API objects and continuously reconciled by control-plane components. The system is built around a control plane that exposes a uniform API and a collection of controllers that act on the stored state to drive the cluster towards convergence. In practical terms, this means that most security-relevant actions, i.e. creating and scaling workloads, updating networking configuration, changing storage bindings, or managing sensitive objects, such as Secrets, are ultimately expressed as requests to the Kubernetes API server. This architecture is summarised in Figure 1, which highlights the API server as the gateway to control plane functionality.

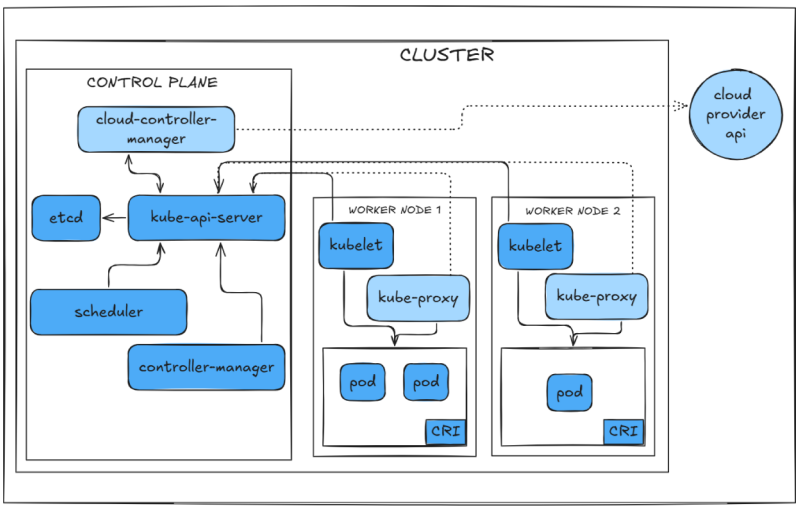

Figure 1. Kubernetes architecture diagram.

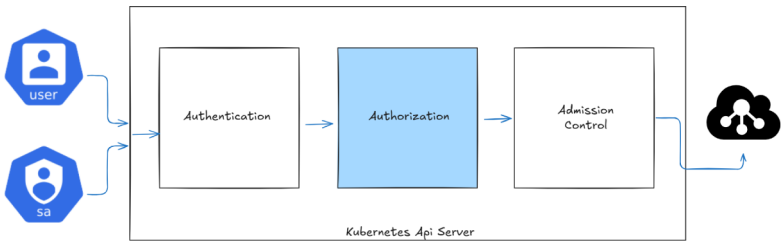

Figure 2. Access control in Kubernetes.

Kubernetes access control is a pipeline of enforcement steps. After a request is received, the API server performs authentication, then authorisation, and finally admission control before persisting the change. This flow is depicted in Figure 2. Authentication establishes the identity (user or service account) associated with the request; authorisation evaluates whether that identity is permitted to perform the requested operation; and admission control performs additional validation (and, in some cases, mutation) of the request prior to persistence [2].

2.1. Authorisation in Kubernetes

After successful authentication, Kubernetes builds an authorisation request that captures the subject identity and groups, the requested verb (e.g. get, create, delete), the API group and resource type, and the target scope (namespace or cluster). Configured authorisers evaluate this request and return an allow or deny decision (some may return 'no opinion'), and when multiple authorisers are enabled, they are evaluated sequentially [3]. As a result, the effective outcome depends not only on individual policies but also on how mechanisms are composed and ordered.

Among the available modes, three are central for fine-grained governance in shared clusters: RBAC, ABAC, and Authorization Webhooks. RBAC, the default in most deployments, is tightly integrated with Kubernetes resources and managed dynamically through standard tooling. ABAC enables conditional rules based on request attributes but is often tied to more static policy workflows. Webhook delegates decisions to an external service via the Subject Access Review API [4], introducing additional flexibility and operational dependencies; these mechanisms can also be combined. In multi-tenant environments, authorisation is further shaped by the distinction between namespace and cluster-scoped resources – since access to cluster-scoped objects can weaken tenant

isolation [5] – and by auditing, which complements authorisation by recording API activity for monitoring and investigation [6].

2.2. Access control paradigms

To compare authorisation mechanisms meaningfully, they must be interpreted through established access control paradigms. In fact, a paradigm often predicts how a mechanism will behave as the number of services, tenants, and identities grows, and how easily policy can evolve without destabilising operations.

Role-based access control assigns permissions to roles and binds roles to subjects. In Kubernetes, this is realised through four primary API objects: Role, Cluster Role, Role Binding, and Cluster Role Binding [7]. The practical strengths of this approach are its transparency and low-latency evaluation: policies are stored in the cluster, evaluated locally by the API server, and can be audited alongside other configuration. However, RBAC policies are fundamentally structured around enumerating permissible verbs and resources. When organisations need conditions that depend on context or resource content (e.g. constraints tied to labels, tenant identity, or environmental state), RBAC tends to require approximation through additional roles, namespace conventions, or external guardrails. This can lead to role proliferation and to policies that are correct but hard to interpret.

Attribute-Based Access Control generalises authorisation decisions by evaluating attributes associated with subjects, objects, requested operations, and potentially the environment. A standard reference point for ABAC is NISTSP800-162, which defines ABAC and describes the functional components and considerations for its application [8]. In the Kubernetes context, ABAC can express conditional rules that would be awkward in pure RBAC. The recurring practical challenge is not expressiveness but lifecycle management: when ABAC policies are treated as static configuration rather than cluster-managed resources, policy changes can become tightly coupled to control-plane configuration workflows.

Beyond native mechanisms, the cloud-native ecosystem increasingly adopts ‘policy-as-code’ practices in which policies are authored, reviewed, tested, and deployed using software engineering workflows. Open Policy Agent (OPA) is a widely used example in this category, with a dedicated policy language (Rego) and a documented model for expressing decisions over structured inputs [9]. Although policy engines can be applied at different points of the

Kubernetes pipeline (authorisation time, admission time, or both), their common promise is governance discipline: policies can be version-controlled, validated through automated tests, and deployed consistently across environments. The common risk is that policy evaluation becomes part of the critical path for platform control, which raises questions of availability, latency, and safe failure behaviour.

A further paradigm that has gained attention for large-scale systems is relationship-based access control, in which access is derived from relationships among subjects and objects (e.g. membership, ownership, delegation). This family of approaches is strongly influenced by Zanzibar, Google's global authorisation system, which formalises a scalable model for representing permissions and evaluating access decisions at high volumes [10]. In practice, these models emphasise a capability that becomes important for digital service systems: efficiently answering not only 'can subject *s* access object *o*?', but also listing and filtering questions, such as 'which objects are accessible to this subject?'

3. Related Work

Prior research on access control and cloud security configuration highlights usability and policy-drift problems that are particularly relevant in Kubernetes, where authorisation rules are embedded in continuously evolving workflows driven by both humans and controllers. Even when an authorisation paradigm is sound in theory, effective permissions can drift as policies are updated, composed with other controls (e.g. admission and policy-as-code), and interpreted across multiple tooling layers. This has motivated work that treats security policy not as a static specification but as an evolving artifact requiring systematic analysis and verification.

In the Kubernetes context, several approaches encode RBAC and related policy layers into formal reasoning frameworks: Event-Calculus models have been used to detect redundancies and semantic conflicts [11], while other work translates RBAC and admission policies into first-order logic and applies SMT solving to check invariants and uncover conflicts, privilege-escalation paths, and unreachable denies [12]. These efforts emphasise policy correctness by modelling authorisation as analysable logic rather than opaque configuration, but they generally focus on verifying specific policy instances and stacks rather than comparing authorisation mechanisms as alternative design choices. Moreover, on the

networking side, policy verification for Kubernetes Network Policies has been studied to address mismatches between traditional network verification tools and Kubernetes's higher-level abstractions. Existing approaches range from scalable verification techniques that check intended connectivity properties over large policy sets, to symbolic methods that detect inconsistencies, conflicts, and unintended reachability [13–15].

As cloud environments became dominant, cloud misconfiguration became a primary cause of severe security incidents. Large-scale studies of misconfigured storage services (e.g. exposed object storage buckets) illustrate how small configuration mistakes can enable broad data exposure [16]. Broader discussions of cloud breaches caused by misconfiguration show that incidents frequently stem from control-plane and configuration errors, such as incorrect access rules, overly permissive policies, or unintended exposure [17]. This has driven multiple research studies on formal verification approaches for cloud computing, attempting to replace static inspection with mathematically grounded assurance [18]. This push is also motivated by real-world needs, as incident analyses repeatedly show that attackers exploit configuration and policy weaknesses to move from an initial foothold to high-impact access [19]. Within cloud IAM specifically, several works propose analysis techniques to detect problematic policies [20]. Other work uses SMT-based reasoning to provide stronger assurance about the semantics of access policies [21]. While these methods provide strong insights into policy semantics and misconfiguration patterns, they are largely scoped to a specific infrastructure model (notably AWS) and do not directly address the broader spectrum of authorisation mechanisms commonly used in Kubernetes environments.

A parallel research area proposes solutions for pre-deployment assessment: rather than discovering weaknesses after deployment, it aims to predict them from configuration artifacts and environment models. Semantic reasoning has been used for pre-deployment security assessment of cloud services, capturing security-relevant relationships and checking whether service compositions violate intended constraints [22]. In the same spirit, reachability analysis approaches model connectivity and check whether unintended paths exist in network configuration files [23]. In this view, many of the analysis techniques later adapted to cloud and Kubernetes settings originate in earlier work on anomaly analysis and verification for firewall and access-control policies [24]. Early contributions classified and analysed conflicts in distributed firewall policies [25], and later work provided methods to detect and resolve them [26].

More recent approaches have pushed this direction towards automated refinement and verification of access-control policies [27], and towards structuring (or 'atomising') policies to enable more systematic anomaly identification and resolution [28]. Related work has also explored automated reconfiguration of firewall rule bases to restore intended security properties after changes [29].

A complementary, increasingly practical direction is tooling that helps operators understand and stress-test the security implications of configurations. For Kubernetes deployments, graph-based generation and attack reconstruction have been proposed to assess the security consequences of deployment choices [30]. More broadly, stress-test methodologies combine permissions, capabilities, and known vulnerabilities into knowledge-graph-like representations to support 'what-if' analyses and to recommend safer configurations [31]. These approaches are less about proving a single property and more about exploring the space of possible consequences, resulting in useful information when operators need decision support rather than a binary 'safe/unsafe' verdict.

Finally, a notable body of work focuses on automation that turns high-level objectives into enforceable policy, and on checking whether different objectives can coexist. In traditional networks, the literature has extensively investigated this problem, leveraging policy and traffic flow models [32, 33] or security configuration abstractions [34] to improve tasks such as security reconfiguration transient optimisation [35]. Instead, in cloud computing settings, where resources may be shared across administrative domains, work on intent-driven network isolation explicitly models cases with simultaneous objectives and introduces mechanisms to verify compatibility, harmonise intents, and translate them into Kubernetes network policies [36, 37].

Recent practitioner and grey literature has started to address the role of more expressive authorisation systems in Kubernetes, although mostly from an implementation-oriented rather than comparative academic perspective. On the OPA side, ecosystem material discusses Kubernetes integration primarily through admission control and policy enforcement workflows, especially via Gatekeeper and related deployment patterns for policy evaluation in cloud-native environments [38, 39]. On the relationship-based side, practitioner sources present SpiceDB as a Zanzibar-inspired authorisation system that can be deployed natively on Kubernetes and extended towards Kubernetes-specific access mediation, for example through the SpiceDB Kubernetes API proxy [40–42].

In addition, CNCF grey literature has begun to document SpiceDB adoption in Kubernetes settings, including operational guidance for Amazon EKS deployments [43]. These materials are valuable because they reflect emerging operational knowledge around externalised authorisation in Kubernetes, but they typically emphasise architecture, deployment, and product-specific guidance, rather than a structured cross-mechanism assessment.

Taken together, the literature provides strong building blocks but it also exhibits a recurring limitation that motivates this paper's positioning. Many works are deliberately vertical. They go deep into a single layer (e.g. IAM, RBAC/admission, Network Policy, Helm, or reachability) to achieve precise results within that technology stack. This leaves open an engineering question central to Kubernetes platform design, addressed by this paper: how to compare different authorisation mechanisms as alternatives that must be operated, audited, and evolved in realistic multi-tenant cluster workflows.

4. Methodology

Our goal is to compare Kubernetes authorisation mechanisms in the way they are encountered in real platform work: policies must be authored, deployed, reviewed, audited, and evolved while the cluster keeps changing. For this reason, the study does not rely on a purely feature-based checklist and does not treat the mechanisms as isolated theoretical models. Instead, we use a structured methodology that combines three elements: (i) a conceptual inspection of each mechanism and its integration point in the Kubernetes request flow, (ii) a scenario-based evaluation that translates typical multi-tenant needs into concrete allow/deny requirements, and (iii) a synthesis step that compares outcomes using a shared evaluation matrix.

4.1. Conceptual inspection

The conceptual inspection establishes a baseline for understanding what each mechanism is designed to control and how it fits into Kubernetes access control. For each mechanism, we examine (a) the policy unit (e.g. roles and bindings, attribute rules, external decision logic, or relationship schemas), (b) the decision semantics (including how allow and deny outcomes are derived), and (c) the operational workflow for policy distribution and updates. This step is crucial because two mechanisms can appear to solve the same problem while imposing very different operational constraints. For example, a mechanism may be expressive in principle

but hard to evolve safely, or simple to operate but limited when policies must incorporate context.

For the relationship-based paradigm, we selected SpiceDB as a representative Zanzibar-inspired open-source system. The rationale for this choice was methodological rather than normative. Since the goal of the paper is a scenario-driven comparison across authorisation paradigms, rather than an exhaustive survey of all ReBAC implementations, we needed one concrete platform that was mature enough for practical experimentation, explicit enough in its authorisation model to support analytical discussion, and sufficiently documented to ensure transparent implementation. SpiceDB met these requirements by providing a schema-based and relationship-centric authorisation model closely aligned with the Zanzibar approach, as well as a deployable integration path for Kubernetes-oriented environments. Its adoption therefore enabled us to operationalise the ReBAC paradigm in a way that was both technically meaningful and comparable with the other mechanisms considered in the study. We do not claim that SpiceDB is inherently superior to alternatives such as OpenFGA or Ory Keto. Rather, we use it as a well-defined and practically viable representative of the broader Zanzibar-inspired family.

4.2. Experimental setup

The scenario-based evaluation was conducted on self-managed Kubernetes test environments prepared to exercise both native and externalised authorisation mechanisms under controlled conditions. The experiments were carried out on Ubuntu 24.04 LTS systems using Kubernetes v1.32 and Kubeadm v1.32 as the cluster bootstrap tool. Native RBAC scenarios were validated on standard kubeadm-based clusters, while mechanisms requiring additional API server configuration or external authorisation logic were deployed on dedicated test environments configured accordingly. This choice was important because some of the mechanisms considered in the study, especially ABAC and Authorization Webhooks, require explicit API server configuration and are therefore best examined in self-managed settings rather than in managed Kubernetes services.

The experimental configuration was designed to support reproducible policy validation rather than throughput benchmarking. In line with the scenario-driven methodology, each mechanism was implemented so as to satisfy the intended allow/deny requirements with minimal sufficient privilege. Verification combined command-based

inspection with representative request execution. In particular, we used kubect1 auth can-i to perform positive and negative authorisation checks, and we complemented these checks with direct API interactions or resource operations when needed to confirm that the observed behaviour matched the intended policy outcome.

Authentication and subject representation were configured consistently with the mechanism under study. Human users and Service Accounts were used according to the scenario requirements, while certificate-based or token-based authentication was adopted where appropriate to reflect the constraints of the selected authorisation mode. This was particularly relevant for mechanisms such as ABAC, where policy matching and subject identification depend more directly on external configuration artifacts. Overall, the purpose of the setup was to provide a controlled and comparable environment for assessing expressiveness, implementation effort, and operational behaviour across mechanisms, rather than to derive benchmark-grade latency or scalability measurements.

4.3. Scenario-based evaluation

To keep the comparison grounded, we evaluate mechanisms through a set of operational scenarios that reflect recurring requirements in shared Kubernetes environments. Scenarios are written as requirements first, without assuming a particular policy language. Each scenario specifies: the subjects involved (human users and service accounts), the target resources and scope (namespace-scoped or cluster-scoped), the actions that must be allowed and those that must be denied, and any constraints that make the scenario security-relevant (e.g. protection of sensitive objects, delegation boundaries, or least-privilege automation).

The scenario set is intentionally heterogeneous. It includes cases that stress common delegation patterns (such as granting a team autonomy within its namespace), cases that stress protection of sensitive objects (such as controlling reads of Secrets), and cases that stress automation safety (ensuring that service accounts can reconcile workloads without becoming over-privileged). These scenarios are representative of the kinds of authorisation challenges that appear, as Kubernetes platforms grow from single-team clusters into shared internal platforms or multi-tenant infrastructures.

Each scenario is implemented for each mechanism under study. Implementations aim for minimal sufficient privilege: the policy grants exactly what is required to satisfy the allowed actions

and blocks what is required to satisfy the denied actions. Where a mechanism cannot represent a requirement directly, we implement the closest approximation and explicitly record the gap. This 'best-effort equivalence' reflects a common operational reality: teams often approximate intended authorisation boundaries when a mechanism cannot express them naturally.

Scenario verification consists of both positive and negative checks. For each subject, we execute representative API requests corresponding to allowed actions and verify that they succeed, then attempt representative forbidden actions and verify that they are denied. Beyond correctness, we also capture operational observations during implementation, including how easy it is to author and review policies, how transparent decisions are when troubleshooting, and how policy changes can be rolled out and validated.

Table 1 provides a compact overview of the scenario set and the main authorisation aspects each scenario stresses. The scenario set is designed to be representative rather than exhaustive. In particular, the coverage of the advanced open-source mechanisms is thinner than that of the native Kubernetes mechanisms. For OPA and SpiceDB, we selected scenarios designed to expose their characteristic policy capabilities, but not to span their full design spaces. Accordingly, conclusions regarding these mechanisms should be interpreted as comparative indications of their operational trade-offs and expressive potential, rather than as exhaustive evaluations of all possible deployment and policy patterns.

4.4. Evaluation matrix

To synthesise results across mechanisms, we use a four-dimensional evaluation matrix: complexity, granularity, scalability, and performance. The matrix is designed to combine security expressiveness with operational viability.

Complexity captures the total burden of adopting and sustaining a mechanism. It includes conceptual load (the number of ideas an operator must understand), configuration volume, integration effort, and day-two operations, such as monitoring and incident response. Complexity also includes the policy lifecycle: whether policies are easy to review, whether changes can be staged and rolled out safely, and whether updates require disruptive control-plane operations.

Granularity captures how precisely a mechanism can represent intended authorisation boundaries without broad approximations.

Table 1. Scenario set used in the scenario-based evaluation.

ID	Scenario (short description)	ID	Scenario (short description)
RBAC			
R1	Two namespaces; each user can read Secrets only in their own namespace (baseline isolation).	R2	Single user may read only one specific Secret in a namespace (resourceName scoping).
R3	ClusterRole bound via RoleBinding (scoped) vs ClusterRoleBinding (clusterwide) to highlight over-privilege risk.	R4	ServiceAccount can read a ConfigMap; an admin user can modify ConfigMaps (human vs workload identities).
R5	ServiceAccount permissions remain namespace-scoped; access does not automatically extend across namespaces.	R6	Cross-namespace read: a workload in one namespace needs to read ConfigMaps in another namespace.
R7	Debugging least privilege using subresources (e.g., pods/log allowed while pods/exec denied).	R8	"Stolen credentials" variant: compare blast radius between a human user and a narrowly scoped ServiceAccount.
R9	Verb restriction: allow create/update/patch/read, deny delete (protect availability in production-like namespace).	R10	Role aggregation using aggregationRule to compose complex roles from smaller ones.
R11	Privilege escalation prevention when managing RBAC objects (constraints on creating/updating roles/bindings).	R12	Non-resource URL rules (e.g., /metrics) with differentiated access.
ABAC			
A1	Single namespace: one user with full permissions, one with read-only permissions.	A2	Environment separation (dev/test/p rod namespaces) with differentiated rights.
A3	Attribute-driven conditions reusing the environment framework (policies depend on environment attributes).	A4	ServiceAccount access to non-resource endpoints (e.g., /metrics).
A5	Group-based rules with multiple resource types; observe rule interactions and manageability.		
Authorization Webhook			
W1	External decision logic granting specific access to a ServiceAccount in the default namespace.	W2	Label-aware decision logic (e.g., allow listing only Pods with env=test, exclude env=prod).
OPA (Policy-based control)			
O1	Context-aware policy with explicit deny semantics (e.g., timedependent constraints).		
SpiceDB (Relationship-based control)			
S1	Relationship-derived permissions via an authorization service/proxy (membership/ ownership/delegation with inheritance).		

We interpret granularity in practical terms: fine-grained control is not only about theoretical expressiveness but about whether a policy can be written in a way that remains understandable and auditable. This includes the ability to scope permissions tightly, encode

contextual constraints when required, and avoid permission leakage when protecting sensitive resources.

Scalability captures how the mechanism behaves as the system grows in users, services, and tenants. This has two facets. Administrative scalability concerns whether policy management remains tractable when the number of roles, rules, or relationships increases. Decision scalability concerns whether authorisation evaluation remains robust under higher request volumes and larger policy datasets. Because multi-tenant platforms frequently repeat patterns across namespaces, scalability also includes support for policy reuse, composition, and avoiding uncontrolled policy sprawl. In addition to governance challenges, rule proliferation can introduce measurable operational overhead. Prior work shows that as rule sets grow in size and redundancy, enforcement components must evaluate increasingly complex decision structures [44].

Performance captures runtime costs of authorisation, primarily decision latency on API requests. Authorisation is on the critical path of Kubernetes API operations, so even small delays can affect both human workflows and automated controllers. For mechanisms that rely on external services, performance assessment also considers dependency risk, such as the effect of network latency, timeouts, and availability on overall control-plane behaviour. Finally, we consider safe failure semantics, because the way authorisation behaves under partial failure has direct security and availability implications.

5. Comparative Analysis

This section presents the comparative analysis of the considered authorisation mechanisms using the shared evaluation matrix introduced in Section 4. The aim is to make trade-offs explicit in terms that matter in real Kubernetes-based service systems: how difficult a mechanism is to adopt and operate, how precisely it can express required constraints, how well it scales with organisational and platform growth, and what performance and reliability implications it introduces.

5.1. Complexity

Role-based access control typically presents the lowest entry barrier because it is fully integrated with Kubernetes objects. Roles and bindings are represented as resources, can be updated

dynamically, and fit naturally into GitOps-style workflows. In practice, RBAC also benefits from broad familiarity: most platform engineers have encounter edit, and basic misconfigurations are relatively easy to diagnose by inspecting roles, bindings, and API audit logs. However, RBAC complexity can increase gradually as organisations grow, primarily due to role proliferation and the need to encode subtle constraints through conventions, rather than through policy semantics.

Attribute-based access control tends to introduce higher operational complexity because policy artifacts are commonly managed outside the normal Kubernetes resource lifecycle. When policy changes require control-plane configuration updates, the authorisation system becomes less agile: changes are harder to stage, riskier to roll out, and less aligned with rapid iteration. Even when the individual rules are readable, the surrounding workflow increases the cost of policy evolution, which is a major factor in shared service environments where authorisation requirements change frequently.

Webhook authorisation shifts complexity away from Kubernetes policy objects and towards the design and operation of an external authorisation service. This can be an advantage if an organisation already operates mature control services, but it introduces an additional component that must be secured, monitored, scaled, and made resilient. The Webhook also becomes part of the critical request path. As a consequence, complexity is driven not only by policy design, but also by distributed systems concerns: latency control, timeout handling, and safe behaviour under partial failure.

Among open-source extensions, policy engines typically add conceptual and operational load because they introduce a dedicated policy language and a deployment mechanism for distributing policy and data. The benefit is that policy becomes a more disciplined artifact: it can be reviewed, tested, and rolled out with stronger engineering practices than *ad hoc* YAML fragments. Relationship-based systems add a different kind of complexity: rather than writing many per-resource permissions, operators define a schema that captures the authorisation domain and then maintain relationships as data. This initial modelling step can be demanding, but it can also reduce long-term complexity when access is naturally derived from organisational structure, because it avoids repeated policy duplication across resources and tenants.

5.2. Granularity

Role-based access control provides clear control over resource types, verbs, and scope, but it remains limited when policies depend on context beyond these attributes. In many clusters, this leads to patterns where RBAC defines a baseline and other mechanisms are added to handle constraints that RBAC cannot express cleanly. For example, protecting sensitive objects such as Secrets often requires careful role design and separation of service accounts, but RBAC alone cannot directly express constraints that depend on object content or on contextual conditions beyond namespace and resource identity.

Attribute-based access control improves granularity in the sense that decisions can incorporate request attributes more flexibly. This can support policies that are difficult to represent in RBAC without role explosion. However, ABAC does not inherently address relationship-driven needs such as delegated administration chains or inherited permissions; it can express them only indirectly through attributes or external data management. Webhook authorisation can be highly granular because decision logic is unconstrained: it can incorporate arbitrary context and external information, and it can implement explicit deny semantics. The trade-off is that this granularity is achieved through custom logic, which must be engineered and maintained and can become hard to audit if policy is embedded in application code, rather than in declarative artifacts.

Policy engines can provide high granularity because policies can encode complex constraints declaratively and can incorporate context in a controlled way. Relationship-based approaches tend to excel when permissions mirror organisational structure, because they represent delegation and inheritance directly rather than forcing them into static roles. In such cases, fine-grained access is achieved by composing relationships rather than by enumerating all permissions. This is particularly relevant in multi-tenant environments where access may depend on team membership, ownership, and resource hierarchy.

5.3. Scalability

Role-based access control can scale well in small-to-medium environments, but it is susceptible to growth in the number of roles and bindings as exceptions accumulate. This ‘policy sprawl’ can make audits and reviews difficult and can increase the risk of accidental over-privilege. ABAC faces a different scaling barrier: as the number of rules grows, policy files become harder to

reason about and changes become increasingly disruptive if they require control-plane configuration updates.

Webhook scalability is dominated by the architecture of an external service. A well-designed authorisation service can scale horizontally, but it also introduces a potential bottleneck and a single point of failure if it is not engineered with high availability. For policy engines, scalability often depends on how policies and data are distributed. Local evaluation patterns can avoid network bottlenecks, but complex policies and large data sets can introduce memory and CPU overhead. Relationship-based systems are designed to scale in domains where permissions are naturally expressed as relationships; their scalability advantage emerges most clearly when organisations need both fast checks and efficient queries over accessible resources, which are important for large service ecosystems and for governance reporting.

5.4. Performance and reliability

Native mechanisms typically have the lowest latency because evaluation occurs within the API server without additional network hops. RBAC, therefore, performs well on high-throughput clusters where authorisation is part of frequent automation loops. ABAC can also have low decision latency, but its operational workflow can introduce indirect availability costs if policy updates require disruptive operations.

Webhook authorisation introduces network-induced latency and depends on timeout and retry behaviour. Under load or network partitions, Webhook behaviour can significantly influence the perceived responsiveness of the Kubernetes API. For externally integrated mechanisms more broadly, the critical question becomes safe failure semantics: whether the system denies requests when the external component is unavailable, or allows them to preserve availability. Fail-closed behaviour preserves security but can cause control-plane disruption; fail-open preserves availability but can undermine policy enforcement. The chosen behaviour must match the threat model and operational posture of the organisation.

Policy engines can be deployed in ways that minimise latency, but their performance depends not only on policy complexity and data size, but also on deployment architecture and evaluation strategy (e.g. co-location and caching). Relationship-based systems typically optimise for fast permission evaluation through caching and efficient storage, but they still introduce an external dependency if decisions are queried over the network. In these cases,

Table 2. Qualitative cross-comparison using the four-dimensional evaluation matrix.

Mechanism	Complexity	Granularity	Scalability	Performance
RBAC	Low	Low–Medium	Medium	High
ABAC	High	Medium	Low	Medium
Webhook	High	High	Medium–High[a]	Low–Medium[b]
OPA	Medium–High	High	High	Medium[c]
SpiceDB	High	Very High	Very High	High[d]

[a]Depends on architecture and operations of the external authorization service.
[b]Sensitive to network RTT and external processing; proximity/caching become critical.
[c]Depends on deployment mode and policy/data size; complex rules and large inputs may increase latency.
[d]External calls add overhead, but graph traversal and caching support fast checks at scale.

engineering the integration to keep authorisation responsive and resilient becomes as important as the policy model itself. Table 2 summarises the comparative assessment across four dimensions.

Prior work broadly supports the qualitative ordering presented in Table 2. Native mechanisms such as RBAC typically exhibit low-decision overhead, as authorisation is handled directly within the Kubernetes control plane rather than delegated to external services [3]. In contrast, studies have shown that ABAC generally performs worse than RBAC due to increased policy complexity and the need to evaluate subject attributes [45]. Webhook- and OPA-based approaches, on the other hand, tend to introduce greater latency sensitivity, since decision evaluation may involve external processing, network communication, and more expressive policy logic [4, 46–48]. However, this overhead is highly dependent on architectural factors such as co-location, caching strategies, and policy structure, meaning that results across different studies are not directly comparable [47, 48]. Similarly, Zanzibar-inspired relationship-based systems aim to maintain responsive authorisation at scale through caching and efficient relationship evaluation, indicating strong performance potential for systems like SpiceDB [10]. Overall, Table 2 should be interpreted as a qualitative synthesis, rather than a strict numerical comparison across equivalent benchmark settings.

5.5. Synthesis

The comparative analysis highlights that no single mechanism dominates across all dimensions. Native RBAC

remains a strong baseline where operational simplicity and low-latency decision-making are primary requirements and the authorisation model is relatively stable. ABAC can express certain conditional requirements, but its operational workflow can make it a poor fit for environments that demand rapid and frequent policy evolution. Webhook authorisation provides maximal flexibility when bespoke logic is needed, but it shifts the burden towards operating an additional security-critical service and managing the reliability and latency consequences of externalised decisions. Policy engines and relationship-based systems become attractive when fine grained constraints, governance discipline, and organisationally derived permissions are central, provided that the added architectural and operational cost is justified by the security and compliance benefits. The next section expands this synthesis into design guidelines oriented towards real deployment decisions.

6. Discussion, Implications, and Design Guidelines

The comparative analysis shows that Kubernetes authorisation is less a matter of selecting a single 'best' mechanism and more a matter of aligning an authorisation strategy with the reality of the platform. Kubernetes-based service systems are dynamic: teams and services evolve, automation is continuous, and security requirements are rarely static. In this environment, authorisation must balance two forces that often pull in opposite directions. On one side, there is the need for strict least privilege, especially in multi-tenant settings where a mistake can lead to cross-tenant impact. On the other side, there is the need for operational usability, because policies that are difficult to author and evolve tend to be bypassed in practice.

A first implication concerns the role of *native RBAC* as a baseline. RBAC remains a strong default for many deployments because it is integrated, fast, and naturally managed through Kubernetes resources. It aligns well with GitOps workflows, supports transparent inspection, and keeps authorisation decisions in the API server without introducing extra dependencies. However, RBAC frequently becomes a ceiling when requirements move beyond basic isolation and delegated administration. In particular, policies that depend on contextual constraints, explicit denial semantics, or rich organisational structure tend to force RBAC into indirect solutions: role proliferation, naming conventions, and external guardrails. These patterns can work, but they increase the risk that permissions drift

overtime, and they make audits harder because intent is distributed across many artifacts.

A second implication concerns *policy evolution*. Authorisation requirements change not only because new services appear, but also because organisations reorganise, compliance requirements shift, and incidents trigger temporary access patterns. Mechanisms that make policy change costly create a structural incentive to grant broad permissions to avoid repeated disruption. This is one of the main practical drawbacks of ABAC in Kubernetes when it is managed as a static configuration. Even when ABAC can encode a desired rule, the operational cost of updating and validating rules can push teams towards fewer, broader policies. In multi-tenant service systems, this dynamic is particularly risky because it widens the gap between intended least privilege and implemented least privilege. When policy refinement is expensive or tightly coupled to control-plane operations, organisations tend to favour over approximation and long-lived privileges in order to preserve operational continuity. Overtime, this leads to privilege accumulation, semantic drift between intended and enforced policy, and increasing difficulty in auditing effective access boundaries.

A third implication is that *externalised authorisation* changes the reliability story of the control plane. Webhook-based authorisation and externally queried policy systems can offer strong expressiveness, but they introduce a dependency that sits directly on the API request path. This dependency must therefore be treated as security-critical infrastructure: it requires hardening, high availability, capacity planning, and careful timeout and retry behaviour. Most importantly, the deployment must make an explicit choice about failure semantics. A fail-closed configuration preserves policy enforcement but can degrade availability of the Kubernetes API when the dependency is unavailable. A fail-open configuration preserves availability but can silently relax security boundaries at the worst possible time. For service systems where authorisation is part of compliance or tenant isolation guarantees, fail-closed behaviour is often necessary, which in turn requires stronger operational maturity for the external component.

These implications lead to practical design guidelines. The first guideline is to treat authorisation as a layered control rather than a single mechanism. In many clusters, RBAC provides a stable base for coarse permissions, while additional mechanisms handle constraints that are difficult to encode in RBAC. When a policy requirement is primarily about *who* can operate on *which* resource types

within clear scopes, RBAC is often sufficient and should be preferred for its simplicity and performance. When a requirement depends on richer conditions, it is typically better to introduce a mechanism designed for such conditions than to approximate through RBAC role sprawl. Approximations can be acceptable for small systems, but as the number of tenants and exceptions grows, approximations become a maintenance liability.

The second guideline is to choose advanced mechanisms based on the shape of the access model. If constraints are primarily *policy-driven* (e.g. rules that depend on request context or organisational governance requirements), policy engines can be a good fit because they allow policies to be written, reviewed, tested, and versioned as a code. This can reduce drift and improve auditability, provided that the organisation invests in the policy lifecycle: code review practices, automated tests, and staged rollouts. If access is primarily *relationship-driven* (e.g. access derived from team membership, ownership hierarchies, and delegated administration chains), relationship-based authorisation becomes attractive because it models that structure directly. In such cases, the effort spent designing a schema is often repaid by better reuse and clearer reasoning as the system scales.

The third guideline is to align authorisation with identity and tenancy design. Many authorisation failures are not caused by the expressiveness of the authoriser but by weak identity practices: shared user credentials, long-lived service account tokens, unclear group ownership, or lack of separation between human and automation identities. Authorisation mechanisms work best when identities reflect real responsibility boundaries and when service accounts are scoped tightly to the controllers or workloads that require them. In multi-tenant clusters, it is particularly important to reduce the blast radius of automation identities and to ensure that tenant-level access cannot be extended indirectly through cluster-scoped permissions.

The fourth guideline is to plan for auditing and explainability as part of authorisation design. When a security event occurs, teams need to answer not only whether an action was permitted, but why it was permitted. Mechanisms that support clear reasoning about decisions reduce the time to investigate incidents and reduce the chance that teams overcorrect by granting broad emergency access. In practical terms, this means designing policies that map to organisational intent, using consistent naming and documentation for roles and rules, and ensuring that audit logs can be correlated

reliably to identities. Advanced mechanisms should be adopted with equal attention to observability: decision logs, metrics, and failure alerts become part of the security posture.

Finally, the analysis suggests that many organisations benefit from an incremental adoption path. A common trajectory starts with RBAC as a baseline, then introduces more expressive controls only when the system reaches a scale or governance maturity that justifies the cost. The key is to avoid adopting advanced mechanisms merely because they are powerful; they are most valuable when they replace brittle approximations and enable policies that would otherwise be impractical to maintain. Conversely, delaying adoption too long can lead to a fragile RBAC configuration that becomes difficult to re-factor, because the platform and organisational structure have already grown around informal exceptions.

7. Conclusions and Future Work

This paper examined fine-grained authorisation in Kubernetes-based digital service systems by comparing native mechanisms (RBAC, ABAC and Authorization Webhooks) with representative open-source approaches that embody more expressive paradigms (policy- and relationship-based authorisation). The comparison was grounded in operational scenarios and synthesised through an evaluation matrix spanning complexity, granularity, scalability, and performance.

The results indicate that no single mechanism dominates across all dimensions. Native RBAC remains a strong and efficient baseline when permissions can be expressed primarily in terms of resource types, verbs, and scope, and when operational simplicity and low-latency decisions are key requirements. ABAC can encode conditional constraints but often suffers from an operational model that makes policy evolution costly in rapidly changing environments. Webhook authorisation offers maximum flexibility, but it shifts the burden towards operating an additional security-critical service and managing the latency and failure semantics introduced by externalised decisions. More expressive approaches become compelling when fine-grained constraints and governance discipline are central requirements. Policy-based systems enable policies to be managed with software engineering practices, while relationship-based systems provide a natural model for permissions that derive from organisational structure and delegation.

While the study provides a structured comparative perspective on Kubernetes authorisation mechanisms, some aspects could be further strengthened in future developments. The scenario set was designed to capture recurring and practically relevant access-control needs in shared Kubernetes environments, but it does not aim to exhaust the full diversity of production authorisation requirements. Likewise, the comparative matrix should be read primarily as a qualitative synthesis, since complexity, granularity, scalability, and performance were assessed through implementation experience, policy analysis, and scenario outcomes rather than through a dedicated benchmarking campaign. In addition, the breadth of analysis is not entirely uniform across all mechanisms, with native Kubernetes solutions examined through a broader scenario set than the more advanced open-source mechanisms. Accordingly, the results are best interpreted as a scenario-driven comparative framework intended to support practitioners and guide further investigation.

Several directions for future work emerge from this study. One direction is to quantify the performance impact of externalised authorisation under realistic mixed workloads, including controller-driven traffic, and to evaluate caching strategies and timeout configurations under failure conditions. A second direction is to study combined deployments more systematically, since real clusters frequently compose mechanisms, for example RBAC for baseline permissions combined with policy engines for additional constraints. A third direction is to explore formal methods and automated tooling to reduce policy drift, including regression testing of authorisation policies against evolving scenario suites and automated detection of over-privileged bindings. Finally, future work could investigate how authorisation strategies interact with other layers of Kubernetes security, such as admission control, network policies, and secret management, with the aim of providing a more integrated blueprint for secure multi-tenant platform operation.

Funding

This project received funding from the European Union's Horizon Europe Research and Innovation Programme under grant agreement No. 101168144 (MIRANDA). The views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the granting authority can be held responsible.

References

[1] D. Bringhenti, R. Sisto, F. Valenza, "Security automation for multi-cluster orchestration in Kubernetes," in Proceedings of the *9th IEEE International Conference on Network Soft-warization, NetSoft 2023, Madrid, Spain, June 19–23, 2023*. New York, NY: IEEE, 2023, pp. 480–485, doi: 10.1109/NETSOFT57336.2023.10175419.

[2] Kubernetes Documentation. (2026). *Admission controllers*. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/. [Accessed: Feb. 14, 2026].

[3] Kubernetes Documentation. (2026). "Authorization." [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/authorization/. [Accessed: Feb. 14, 2026].

[4] Kubernetes Documentation. (2025). "Webhook mode." [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/webhook/. [Accessed: Feb. 14, 2026].

[5] Kubernetes Documentation. (2025). "Multi-tenancy." [Online]. Available: https://kubernetes.io/docs/concepts/security/multi-tenancy/. [Accessed: Feb. 14, 2026].

[6] Kubernetes Documentation. (2025). "Auditing." [Online]. Available: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/. [Accessed: Feb. 14, 2026].

[7] Kubernetes Documentation. (2026). "Using RBAC authorization." [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/rbac/. [Accessed: Feb. 14, 2026].

[8] V.C. Hu, D.R. Kuhn, D.F. Ferraiolo, J. Voas, A. Schnitzer, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations.* NIST Special Publication 800-162. Gaithersburg, MD: National Institute of Standards and Technology, 2014.

[9] Open Policy Agent Documentation. (2026). *Policy language.* [Online]. Available: https://www.openpolicyagent.org/docs/policy-language. [Accessed: Feb. 14, 2026].

[10] R. Pang, R. Caceres, M. Burrows, Z. Chen, P. Dave, et al., "Zanzibar: Google's consistent, global authorization system," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2019, D. Malkhi and D. Tsafrir (Eds.), USENIX Association, USA.

[11] E. Zahoor, M. Chaudhary, S. Akhtar, O. Perrin, "A formal approach for the identification of redundant authorization policies in Kubernetes," *Computers & Security*, vol. 135, Art. no. 103473, 2023, doi: 10.1016/J.COSE.2023.103473.

[12] A. Sissodiya, E. Chiquito, U. Bodin, J. Kristiansson, "Formal verification for preventing misconfigured access policies in Kubernetes clusters," *IEEE Access*, vol. 13, pp. 141798–141813, 2025, doi: 10.1109/ACCESS.2025.3597504.

[13] Y. Li, X. Hu, C. Jia, K. Wang, J. Li, "Kano: Efficient cloud native network policy verification," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3747–3764, 2023, doi: 10.1109/TNSM.2022.3229675.

[14] H. Kang, S. Shin, "*Verikube*: Automatic and efficient verification for container network policies," *IEICE Transactions on Information and Systems*, vol. 105-D, no. 12, pp. 2131–2134, 2022, doi: 10.1587/TRANSINF.2022EDL8046.

[15] S. Dong, Y. Xie, J. Zhao, K. Qiu, "NPV: Fast network policy verification for cloud-native networking," in *44th IEEE International Conference on Distributed Computing Systems, ICDCS 2024, Jersey City, NJ, July 23–26, 2024*. New York, NY: IEEE, 2024, pp. 461–472, doi: 10.1109/ICDCS60910.2024.00050.

[16] A. Continella, M. Polino, M. Pogliani, S. Zanero, "There's a hole in that bucket!: A large-scale analysis of misconfigured S3 buckets," in *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03–07, 2018*. New York, NY: ACM, 2018, pp. 702–711, doi: 10.1145/3274694.3274736.

[17] S. Mehra, S. Sinha, V. Chaudhary, "Data breach in cloud computing due to misconfigurations," in *Integration of Cloud Computing with Emerging Technologies*. Boca Raton, FL: CRC Press, 2023, pp. 55–64.

[18] A. Souri, N.J. Navimipour, A.M. Rahmani, "Formal verification approaches and standards in the cloud computing: A comprehensive and systematic review," *Computer Standards & Interfaces*, vol. 58, pp. 1–22, 2018, doi: 10.1016/J.CSI.2017.11.007.

[19] S. Khan, I. Kabanov, Y. Hua, S.E. Madnick, "A systematic analysis of the capital one data breach: Critical lessons learned," *ACM Transactions on Privacy and Security*, vol. 26, no. 1, pp. 1–29, 2023, doi: 10.1145/3546068.

[20] T. van Ede, N. Khasuntsev, B. Steen, A. Continella, "Detecting anomalous misconfigurations in AWS identity and access management policies," in *Proceedings of the 2022 on Cloud Computing Security Workshop, CCSW2022, Los Angeles, CA, USA, 7 November 2022*. New York, NY: ACM, 2022, pp. 63–74, doi: 10.1145/3560810.3564264.

[21] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, et al., "Semantic-based automated reasoning for AWS access policies using SMT," in *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30–November 2, 2018*. 2018, pp. 1–9, doi: 10.23919/FMCAD.2018.8602994.

[22] C. Cauli, M. Li, N. Piterman, O. Tkachuk, "Pre-deployment security assessment for cloud services through semantic reasoning," in *Proceedings of the 33rd International Conference on Computer Aided Verification (CAV), Virtual Event, July 20–23, 2021*, Lecture Notes in Computer Science Series, vol. 12759. Cham Springer, 2021, pp. 767–780, doi: 10.1007/978-3-030-81685-8_36.

[23] J. Backes, S. Bayless, B. Cook, C. Dodge, A. Gacek, "Reachability analysis for AWS-based networks," in *Computer Aided Verification – Proceedings of the 31st International Conference, CAV 2019, New York City, NY, USA, July 15–18, 2019,* Lecture Notes in Computer Science Series, Part II, vol. 11562. 2019, pp. 231–241, doi: 10.1007/978-3-030-25543-5_14.

[24] F. Valenza, S. Spinoso, R. Sisto, "Formally specifying and checking policies and anomalies in service function chaining," *Journal of Network and Computer Applications*, vol. 146, 2019, Article n° 102419, pp. 1–14 doi: 10.1016/J.JNCA.2019.102419.

[25] E. Al-Shaer, H.H. Hamed, R. Boutaba, M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, 2005, doi: 10.1109/JSAC.2005.854119.

[26] H. Hu, G. Ahn, K. Kulkarni, "Detecting and resolving firewall policy anomalies," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 3, pp. 318–331, 2012, doi: 10.1109/TDSC.2012.20.

[27] M. Cheminod, L. Durante, L. Seno, F. Valenza, A. Valenzano, "A comprehensive approach to the automatic refinement and verification of access control policies," *Computers & Security*, vol. 80, pp. 186–199, 2019, doi: 10.1016/J.COSE.2018.09.013.

[28] D. Bringhenti, S. Bussa, R. Sisto, F. Valenza, "Atomizing firewall policies for anomaly analysis and resolution," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 3, pp. 2308–2325, 2025, doi: 10.1109/TDSC.2024.3495230.

[29] F. Pizzato, D. Bringhenti, R. Sisto, F. Valenza, "Automatic and optimized firewall reconfiguration," in *NOMS 2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, May 6–10, 2024.* New York, NY: IEEE, 2024, pp. 1–9, doi: 10.1109/NOMS59830.2024.10575212.

[30] A. Blaise, F. Rebecchi, "Stay at the helm: Secure Kubernetes deployments via graph generation and attack reconstruction," in *IEEE 15th International Conference on Cloud Computing, Barcelona, Spain, July 10–16, 2022*. 2022, pp. 59–69, doi: 10.1109/CLOUD55607.2022.00022.

[31] F. Minna, F. Massacci, K. Tuma, "Towards a security stress-test for cloud configurations," in *IEEE 15th International Conference on Cloud Computing, Barcelona, Spain, July 10–16, 2022.* 2022, pp. 191–196, doi: 10.1109/CLOUD55607.2022.00038.

[32] D. Bringhenti, F. Valenza, "A two-fold model for VNF embedding and time-sensitive network flow scheduling," *IEEE Access*, vol. 10, pp. 44384–44399, 2022, doi: 10.1109/ACCESS.2022.3169863.

[33] D. Bringhenti, S. Bussa, R. Sisto, F. Valenza, "A two-fold traffic flow model for network security management," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 3740–3758, 2024, doi: 10.1109/TNSM.2024.3407159.

[34] D. Bringhenti, R. Sisto, F. Valenza, "A novel abstraction for security configuration in virtual networks," *Computer Networks*, vol. 228, Art. no. 109745, 2023, doi: 10.1016/J.COMNET.2023.109745.

[35] D. Bringhenti, F. Valenza, "Optimizing distributed firewall reconfiguration transients," *Computer Networks*, vol. 215, Art. no. 109183, 2022, doi: 10.1016/J.COMNET.2022.109183.

[36] F. Pizzato, D. Bringhenti, R. Sisto, F. Valenza, "An intent-based solution for network isolation in Kubernetes," in *10th IEEE International Conference on Network Softwarization, NetSoft 2024, Saint Louis, MO, USA, June 24–28, 2024.* New York, NY: IEEE, 2024, pp. 381–386, doi: 10.1109/NETSOFT60951.2024.10588939.

[37] F. Pizzato, D. Bringhenti, R. Sisto, F. Valenza, "Intent-driven network isolation for the cloud computing continuum," *Journal of Network and Systems Management*, vol. 34, no. 1, Art. no. 6, 2026, doi: 10.1007/S10922-025-09986-1.

[38] Kubernetes Blog. (2019). *OPA gatekeeper: Policy and governance for Kubernetes.* [Online]. Available: https://kubernetes.io/blog/2019/08/06/opa-gatekeeper-policy-and-governance-for-kubernetes. [Accessed: Apr. 01, 2026].

[39] Open Policy Agent. (2026). *Deploying OPA on kubernetes.* [Online]. Available: https://www.openpolicyagent.org/docs/deploy/k8s. [Accessed: Apr. 01, 2026].

[40] Authzed. (2026). *Spicedb for open policy agent (OPA) users*. [Online]. Available: https://authzed.com/docs/spicedb/getting-started/coming-from/opa. [Accessed: Apr. 01, 2026].

[41] Authzed. (2026). *Installing spicedb on kubernetes*. [Online]. Available: https://authzed.com/docs/spicedb/getting-started/install/kubernetes. [Accessed: Apr. 01, 2026].

[42] E. Cordell. (2024). *ACL filtering kubernetes apis using spicedb*. [Online]. Available: https://authzed.com/blog/acl-filtering-kubernetes-apis-using-spicedb. [Accessed: Apr. 01, 2026].

[43] Cloud Native Computing Foundation. (Oct. 31, 2024). *CNCF on-demand webinar: Build authorization at scale with spicedb on Amazon EKS*. [Online]. Available: https://www.cncf.io/online-programs/cncf-on-demand-webinar-build-authorization-at-scale-with-spicedb-on-amazon-eks/. [Accessed: Apr. 01, 2026].

[44] D. Bringhenti, F. Valenza, "Greenshield: Optimizing firewall configuration for sustainable networks," *IEEE Transactions on Network and Service Management*, vol. 21, no. 6, pp. 6909–6923, 2024, doi: 10.1109/TNSM.2024.3452150.

[45] G. Batra, V. Atluri, J. Vaidya, S. Sural, "Enabling the deployment of ABAC policies in RBAC systems," in *Data and Applications Security and Privacy*, *XXXII – 32nd Annual IFIP WG 11.3 Conference, DBSec 2018, Bergamo, Italy, July 16–18, 2018, Proceedings*, F. Kerschbaum, S. Paraboschi, Eds., inLecture Notes in Computer Science. Cham: Springer, 2018, pp. 51–68, doi: 10.1007/978-3-319-95729-6_4.

[46] S. Berlato, R. Carbone, S. Ranise, "A methodology for the experimental performance evaluation of access control enforcement mechanisms based on business processes," *Journal of Information Security and Applications*, vol. 93, Art. no. 104158, 2025, doi: 10.1016/J.JISA.2025.104158.

[47] H. Kermabon-Bobinnec, M Gholipourchoubeh, S. Bagheri, S. Majumdar, Y. Jarraya et al., "Prospec: Proactive security policy enforcement for containers," in *CODASPY '22: Twelfth ACM Conference on Data and Application Security and Privacy, Baltimore, MD, USA, April 24–27, 2022*, A. Joshi, M. Fernández, R.M. Verma, Eds. New York, NY: ACM, 2022, pp. 155–166, doi: 10.1145/3508398.3511515.

[48] M. Rossi, M. Beretta, D. Facchinetti, S. Paraboschi, "Secure Kubernetes workload deployment with automated enforcement of cluster-defined policies," in *IEEE International Conference on Cloud Computing Technology and Science, CloudCom2025, Shenzhen, China, November 14–16, 2025*. New York, NY: IEEE, 2025, pp. 1–8, doi: 10.1109/CLOUDCOM67567.2025.11331483.