International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Zero-Trust API Platforms: Secure API Gateway Architectures

Sudarshan T N

Abstract: Application Programming Interfaces have become the fundamental connective tissue of modern digital ecosystems, enabling seamless data exchange across organizational boundaries, cloud platforms, and device types. As organizations in highly regulated sectors increasingly rely on API-driven architectures, the attack surface associated with these interfaces has expanded dramatically, rendering traditional perimeter-based security models inadequate against sophisticated threats, including supply chain compromises, insider attacks, and lateral movement by advanced persistent threat actors. This paper presents a comprehensive examination of how Zero-Trust security principles can be systematically applied to modern API platforms. The Zero-Trust paradigm, rooted in the axiom of never trust, always verify, demands that every API request be authenticated, authorized, and validated regardless of its origin. This paper explores the key pillars of a Zero-Trust API architecture: robust authentication and authorization models including OAuth 2.0, OpenID Connect, mutual TLS, and SPIFFE-based workload identity; deep integration with enterprise identity providers; fine-grained policy enforcement leveraging the Kubernetes Gateway API and policy engines such as Open Policy Agent; and the critical role of service meshes in securing east-west API traffic within microservices architectures. Additionally, this paper offers a layered reference blueprint for organizations seeking to implement Zero-Trust API architecture from edge security through to observability and compliance.

Keywords: *Zero-Trust, API Security, API Gateway, OAuth 2.0, OpenID Connect, Mutual TLS, SPIFFE, Open Policy Agent, Service Mesh, Kubernetes Gateway API, Identity Provider, Policy Enforcement*

Introduction

1.1 The Expanding API Threat Landscape

The proliferation of APIs across industries has fundamentally altered the threat landscape facing modern organizations. The average enterprise now manages thousands of APIs spanning internal microservices, partner integrations, public developer portals, and IoT device communication channels [1]. Each API endpoint represents a potential entry point for malicious actors, and the consequences of a breach can be catastrophic in regulated sectors where stringent data protection requirements and severe non-compliance penalties apply. OWASP's API Security Top 10, first published in 2019 and updated in 2023, catalogs the most critical API vulnerability categories, including broken object-level authorization, authentication weaknesses, and excessive data exposure, collectively representing the primary attack vectors responsible for the majority of API-related security incidents [2].Traditional perimeter-based security architectures, which grant implicit trust to all traffic originating from within the corporate network boundary, have proven fundamentally inadequate for securing modern API ecosystems. The dissolution of the network perimeter through cloud adoption, remote work, and API-driven partner integrations means that the concept of a trusted internal network no longer reflects the actual security topology of enterprise environments. Advanced persistent threat actors routinely achieve lateral movement from an initial point of compromise to sensitive API endpoints by exploiting the implicit trust granted to internal traffic, while supply chain attacks demonstrate that even trusted software components can become vectors for unauthorized access [3].

This paper proposes a Zero-Trust API platform architecture that addresses these challenges by eliminating implicit trust from all API communication paths and replacing it with continuous verification of identity, authorization, and context for every API request. The paper is organized as follows: Section 2 defines the Zero-Trust principles applicable to API platforms. Section 3 examines authentication and authorization models. Section 4 describes workload identity with SPIFFE/SPIRE. Section 5 covers policy enforcement architecture. Section 6 examines service mesh integration. Section 7 presents the Kubernetes Gateway API integration. Section 8 addresses observability and threat detection, and Section 9 concludes.

Independent Researcher, USA

1.2 Zero-Trust Principles for API Security

The Zero-Trust security model, originally articulated by Forrester Research and subsequently formalized in NIST Special Publication 800-207, is built on three foundational tenets: verify explicitly, meaning every access request must be fully authenticated and authorized using all available data points; use least privilege access, meaning access rights are minimized to the specific resources and operations required for each request; and assume breach, meaning security controls are designed to limit blast radius and contain damage on the assumption that the network has already been compromised [4]. Applied to API platforms, these tenets translate into concrete architectural requirements: every API call must carry a valid, verifiable identity credential; authorization decisions must be made at the individual request level rather than at the network perimeter; and API gateway infrastructure must be designed to detect and contain anomalous access patterns that may indicate an active breach.

Figure 1: Zero-Trust API Platform Layered Architecture.

Fig. 1. Zero-Trust API platform layered architecture showing the Kubernetes Gateway API north-south enforcement plane, OPA policy engine, SPIFFE/SPIRE workload identity, service mesh east-west mutual TLS enforcement, and centralized OpenTelemetry observability, all operating within a continuous verification boundary.

2. Authentication and Authorization Architecture

2.1 OAuth 2.0 and OpenID Connect

OAuth 2.0 and OpenID Connect provide the foundational authentication and authorization protocols for securing external-facing APIs in modern platforms. OAuth 2.0 defines a delegation framework that enables clients to obtain limited access to protected resources on behalf of a resource owner without exposing the owner's credentials to the client. The protocol has evolved from its original design for consumer web authorization into a comprehensive framework for API security through extensions, including Proof Key for Code Exchange, Token Binding, and the Resource Indicators specification that constrains token scope to specific API resources [5]. OpenID Connect extends OAuth 2.0 with an identity layer that provides standardized claims about the authenticated user or client, enabling APIs to make authorization decisions based on verified identity attributes without requiring direct integration with the identity provider for each request.

The choice of OAuth 2.0 grant type is critical for Zero-Trust API security. The authorization code flow with PKCE is recommended for user-facing API access, where a human user is involved, to protect against authorization code interception attacks. The client credentials flow is appropriate for machine-to-machine API communication where no user is present, with strong client authentication enforced through either client secret with high entropy or client certificate-based authentication. The device authorization flow supports IoT and constrained client scenarios. In all cases, the Zero-Trust API platform requires short-lived access tokens with audiences scoped to specific API resources, combined with token introspection or signed JWT validation at the API gateway to verify token validity before forwarding requests to backend services [6].

Token validation in the Zero-Trust API platform must be performed at the gateway layer for all

incoming requests, with cryptographic verification of token signatures using the identity provider's public key infrastructure. The gateway maintains a cached copy of the identity provider's JSON Web Key Set and periodically refreshes it to pick up key rotation events. Token validation checks include signature verification, expiration validation, audience verification (confirming the token is scoped to the target API), scope verification (confirming the token authorizes the requested operation), and claim validation for any additional claims required by the API's authorization policies. Invalid tokens are rejected at the gateway before the request reaches the backend service, preventing unauthorized access even in the case of application-layer authorization vulnerabilities [7].

Grant Type	Applicable Scenario	Zero-Trust Enforcement Requirement
Authorization code with PKCE	User-facing API access where a human user initiates the request from a browser or native client [5]	Short-lived access tokens scoped to specific API resources; token introspection or signed JWT validation at the gateway before forwarding to the backend
Client credentials	Machine-to-machine API communication where no user is present, such as microservice-to-microservice calls, bypassing the service mesh [6]	Strong client authentication via high-entropy client secret or client certificate; audience and scope binding to the target API resource
Device authorization	IoT and constrained client scenarios where the device cannot initiate a browser redirect [5]	Token lifetime minimized; device identity bound to SPIFFE workload attestation, where the platform supports it; rate limiting enforced at the gateway per device identity

Table 1: OAuth 2.0 Grant Type Selection for Zero-Trust API Scenarios.

2.2 Mutual TLS for Service-to-Service Authentication

Mutual TLS provides the authentication mechanism for service-to-service API communication in the microservices architecture, enabling both the client and the server to verify each other's cryptographic identities before establishing a connection. Unlike one-way TLS, where only the server presents a certificate, mutual TLS requires the client to present a certificate signed by a trusted certificate authority, providing strong proof of the client's identity that cannot be forged without access to the corresponding private key [8]. In a microservices platform, the service mesh infrastructure automates the issuance, rotation, and validation of mutual TLS certificates, eliminating the operational complexity that has historically made mutual TLS difficult to implement at scale.

The certificate lifecycle in a Zero-Trust API platform is managed by the service mesh control plane, which issues short-lived SPIFFE-based certificates to each workload through the Envoy proxy sidecar. Short certificate lifetimes, typically 24 hours or less, limit the window during which a compromised certificate can be used for unauthorized access and eliminate the need for certificate revocation mechanisms that introduce latency and availability risks. The certificate renewal process is automated and transparent to application code, with the service-mesh sidecar handling renewal before certificate expiration, without requiring an application restart or intervention from the development team [9].

3. Workload Identity with SPIFFE and SPIRE

The Secure Production Identity Framework for Everyone (SPIFFE) provides a standardized specification for workload identity in cloud-native environments, addressing the fundamental challenge of establishing cryptographic identity for software workloads in dynamic, ephemeral infrastructure [9]. SPIFFE defines a URI-based identity format, the SPIFFE Verifiable Identity Document (SVID), and a workload API for distributing identity credentials to workloads without requiring them to manage certificate operations directly. SPIRE, the reference implementation of the SPIFFE specification, provides the runtime infrastructure for issuing and rotating SPIFFE-compliant identity documents across multi-cloud and hybrid environments.

Workload identity through SPIFFE provides several advantages over alternative approaches to service-to-service authentication. Unlike API keys or shared secrets, which require secure distribution and storage, SPIFFE identities are cryptographically bound to the workload's runtime characteristics and cannot be trivially exfiltrated or reused in a different environment. The identity attestation process verifies that the requesting workload matches its declared identity using platform-specific attestors that validate Kubernetes pod annotations, AWS instance metadata, or other runtime properties before issuing identity credentials [8]. This attestation model implements the Zero-Trust principle of verifying explicitly at the workload level, ensuring that identity credentials are issued only to workloads that can prove their identity through cryptographic attestation rather than merely knowing a shared secret.

The integration of SPIFFE workload identity with the service mesh and API gateway infrastructure creates a unified identity plane that spans both north-south traffic from external clients through the API gateway and east-west traffic between internal microservices. External client identities, established via OAuth 2.0 tokens at the gateway layer, are propagated to backend services via JWT forwarding or token exchange, enabling them to make fine-grained authorization decisions based on the original caller's identity. Internal service identities, established through SPIFFE certificates managed by SPIRE, authenticate all service-to-service calls within the platform. This unified identity model enables end-to-end traceability for every API call, supporting both security incident investigation and regulatory audit requirements [10].

4. Policy Enforcement Architecture

4.1 Open Policy Agent for Authorization

Open Policy Agent (OPA) provides the policy engine that implements fine-grained authorization decisions in the Zero-Trust API platform. OPA evaluates authorization queries using the Rego declarative policy language, enabling policies to incorporate complex logic, including claims-based conditions, time-of-day restrictions, geographic location constraints, and data classification labels that cannot be expressed through simple role-based access control [11]. The OPA policy engine is deployed as a sidecar or external service consulted by the API gateway for every authorization decision, with policies cached locally for low-latency evaluation. Policy updates are distributed through the OPA bundle server, which provides versioned policy packages that enable atomic policy deployments and rollbacks through the same GitOps pipeline used for other platform configurations.

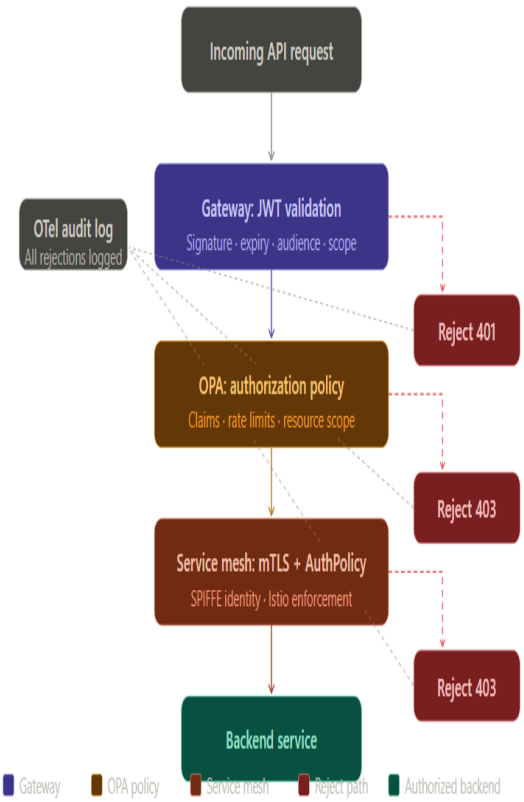

Figure 2: Zero-Trust API Per-Request Authentication and Authorization.

Fig. 2. Zero-Trust API per-request authorization decision flow illustrating the three sequential

enforcement gates — JWT token validation at the Kubernetes Gateway API, Rego-based OPA policy evaluation, and service mesh mutual TLS identity verification — with reject paths at each gate feeding the centralized OpenTelemetry audit log.

The Zero-Trust API authorization policy model combines coarse-grained policies enforced at the API gateway with fine-grained policies enforced at the service level. Gateway-level policies implement the first line of defense against unauthorized access, validating token claims, enforcing rate limits, and rejecting requests that violate organizational security policies before they reach the backend service. Service-level OPA sidecars implement resource-level authorization that verifies the requester's permission to access specific data objects or perform specific operations, thereby implementing the object-level authorization controls, which are the most critical and most frequently violated category in the OWASP API Security Top 10 [2]. This defense-in-depth approach ensures that authorization failures at either layer prevent unauthorized data access, eliminating the single-point-of-failure risk of relying on either gateway or service-level authorization alone [12].

Enforcement Layer	Authorization Scope	Tooling and Mechanism
API gateway layer	First-line coarse-grained enforcement: token claims validation, rate limiting, rejection of requests violating organizational security policies before reaching backend services [11]	Kubernetes Gateway API with OPA policy attachment; JWT validation against identity provider JWKS; HTTPRoute-scoped policy filters
Service mesh layer (east-west)	Cryptographic identity verification for all service-to-service calls; mutual TLS enforcement; Istio Authorization Policy by SPIFFE principal, namespace, HTTP method, and path [13]	Envoy proxy sidecar applies policy transparently to application code; SPIRE-issued SVID certificates are rotated automatically
Service-level OPA sidecar	Fine-grained object-level authorization verifying requestor permission to access specific data objects or perform specific operations, addressing the top OWASP API Security category [2]	OPA Rego policies evaluated per request; policy bundles distributed via GitOps pipeline; defense-in-depth eliminates single-point authorization failure [12]

Table 2: Defense-in-Depth Authorization Control Layers.

4.2 Policy as Code and GitOps Integration

Zero-Trust API policies are managed as code through the same GitOps pipeline used for infrastructure and application configuration, ensuring that policy changes are subject to the same review, testing, and deployment controls as other platform changes. OPA policy bundles are stored in Git repositories, with automated testing pipelines that validate policy correctness using OPA's built-in testing framework before changes are promoted to production. Policy change proposals are subject to peer review by security engineers, and changes to high-sensitivity policies may require approval from a security review board, depending on the organization's risk management framework. The complete history of policy changes is preserved in Git, providing the audit trail required by regulatory frameworks that mandate change management for security controls [11].

5. Service Mesh Integration for East-West Security

Service mesh infrastructure secures east-west API traffic between microservices by enforcing mutual TLS authentication, authorization policies, and traffic management for all service-to-service communication. In the Zero-Trust API platform, the service mesh extends the trust boundary established

at the north-south API gateway to the interior of the microservices architecture, ensuring that the Zero-Trust principle of never trust, always verify applies to internal service calls as well as external API requests [13]. Without service mesh enforcement, a compromised internal service could make arbitrary calls to any other service in the cluster, leveraging the implicit trust that flat internal networks grant to all traffic. The service mesh eliminates this implicit trust by requiring cryptographic identity for every service call and enforcing authorization policies that specify exactly which services are permitted to communicate with which endpoints.

Istio's Authorization Policy resource provides the primary mechanism for enforcing east-west API authorization in the service mesh. Authorization policies specify allowed sources (by SPIFFE identity, namespace, or principal), allowed operations (by HTTP method, path, and headers), and allowed conditions (by JWT claims, source IP, or request metadata). The policy enforcement is performed by the Envoy proxy sidecar in each pod, which applies the authorization policy before forwarding the request to the application container. This enforcement model is transparent to application code, enabling Zero-Trust authorization without modifying application logic [13]. The service mesh also provides mutual TLS authentication as a default for all service-to-service communication, with the control plane automatically provisioning and rotating SPIFFE-based certificates for each workload.

Service mesh observability capabilities are essential for Zero-Trust API security operations. The Envoy proxy sidecar generates detailed access logs and distributed traces for every service call, including the source identity, destination service, HTTP method, path, response code, and latency. These signals feed the security observability pipeline, enabling detection of anomalous access patterns such as unusual call volumes, unexpected service communication paths, and authentication failures that may indicate an active breach or reconnaissance activity. The correlation of service mesh access logs with application-layer event logs through OpenTelemetry provides the end-to-end visibility required for rapid incident response and forensic investigation [14].

6. Kubernetes Gateway API Integration

The Kubernetes Gateway API provides the north-south traffic management layer for the Zero-Trust API platform, implementing the external-facing API gateway that authenticates and authorizes incoming requests before routing them to backend services. The Gateway API's extensibility model, implemented through policy attachment and filter extensions, enables Zero-Trust security capabilities, including JWT authentication, OAuth 2.0 token validation, rate limiting, and request transformation to be expressed as declarative Kubernetes resources managed through the GitOps pipeline [15]. This declarative approach to API gateway configuration provides the consistency, auditability, and change management capabilities required for Zero-Trust API security governance.

HTTPRoute resources in the Kubernetes Gateway API define routing rules that map incoming API requests to backend services based on request attributes, such as path, method, headers, and query parameters. In the Zero-Trust API platform, each HTTPRoute is associated with authentication and authorization policies that must be satisfied before the gateway routes the request to the backend. The policy attachment mechanism enables security policies to be applied at different levels of granularity, from global policies that apply to all routes on a gateway to path-specific policies that apply only to particular API endpoints. This granular policy model enables differentiated security requirements for APIs with varying sensitivity levels within the same gateway infrastructure [15].

API rate limiting and quota enforcement are critical Zero-Trust controls that prevent abuse of API endpoints by compromised clients or applications. The Gateway API integration with a rate-limiting service implements per-client, per-endpoint, and per-tenant rate limits based on the authenticated identity of the requesting client, enabling fine-grained control over API consumption patterns. Rate limit violations generate security events that are forwarded to the observability pipeline for anomaly detection analysis, enabling the security team to distinguish between benign traffic spikes and potential credential stuffing or API abuse attacks. Dynamic rate limit adjustment based on AI-driven threat detection system anomaly scores enables the platform to automatically respond to detected attacks by throttling suspicious clients without requiring manual intervention [16].

7. Observability and Threat Detection

Zero-Trust API security requires a comprehensive observability architecture that provides real-time visibility into authentication events, authorization decisions, API usage patterns, and anomalous behaviors across the entire API platform. The observability stack integrates data from the API gateway access logs, service mesh telemetry, OPA policy evaluation logs, and application-level security events into a unified security event pipeline that enables correlation analysis and anomaly detection [14]. OpenTelemetry provides the standardized instrumentation framework for collecting and forwarding observability data from all platform components, with a centralized collector infrastructure that normalizes event formats and enriches events with contextual metadata before forwarding to the security analytics platform.

Threat detection in the Zero-Trust API platform combines rule-based detection for known attack patterns and machine learning-based anomaly detection for novel or zero-day attacks. Rule-based detection identifies known attack signatures such as SQL injection attempts in API parameters, credential stuffing patterns characterized by high-volume authentication failures from multiple source addresses, and API enumeration behavior characterized by systematic scanning of API paths. Machine learning anomaly detection models are trained on historical API access patterns and learn the normal behavioral profile of each client identity, enabling detection of subtle behavioral anomalies that do not match known attack signatures but deviate significantly from established normal behavior [17].

Security incident response in the Zero-Trust API platform is supported by automated response capabilities that can take immediate protective action when a threat is detected, without waiting for human analyst review. Automated responses include temporarily suspending a compromised client credential when credential abuse is detected, dynamically reducing rate limits for clients exhibiting anomalous request patterns, and automatically escalating affected API resources to enhanced monitoring mode that captures additional forensic detail. All automated response actions are logged with the triggering detection event and the model or rule that generated the alert, providing the evidence chain required for post-incident review and the documentation required by regulated industries for security incident reporting [18].

8. Conclusion

The Zero-Trust API platform architecture presented in this paper provides a systematic framework for applying Zero-Trust security principles to modern API ecosystems in regulated and high-security environments. The layered architecture, integrating OAuth 2.0 and OpenID Connect authentication at the external gateway, SPIFFE-based workload identity for service-to-service authentication, Open Policy Agent for fine-grained authorization, service mesh enforcement for east-west traffic security, and Kubernetes Gateway API for declarative traffic management, addresses the full spectrum of API security requirements from edge to service layer. The declarative, code-managed approach to policy configuration ensures that security controls are auditable, testable, and subject to the same change management rigor as application code.

The elimination of implicit trust from all API communication paths fundamentally changes the platform's security posture by removing the assumption that internal network location implies authorization. Every API call, whether originating from an external client or from an internal microservice, must present a verifiable identity credential and satisfy applicable authorization policies before accessing protected resources. This consistent enforcement model limits the blast radius of credential compromise by ensuring that compromised credentials can only access the resources explicitly authorized to the compromised identity, rather than the broader access implied by network location in traditional perimeter-based architectures. The continuous authentication and authorization model also enables more granular audit trails that support regulatory reporting and incident investigation requirements.

Future research directions include applying AI-driven behavioral analysis to API security to detect sophisticated multi-stage attacks that evade rule-based detection by distributing malicious activity across extended time windows. The development of standardized API security assessment frameworks aligned with the Zero-Trust maturity model would provide organizations with a systematic approach to evaluating and improving their API security posture. As API ecosystems continue to grow in scale and complexity, the architectural patterns described in this paper will provide the foundational security

framework needed to maintain effective governance and protection of the data and services that APIs expose to an increasingly interconnected digital world.

References

[1] OWASP, "OWASP API security top 10 2023," OWASP Foundation, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0x00-header/

[2] OWASP, "OWASP API security project," OWASP Foundation, 2024. [Online]. Available: https://owasp.org/www-project-api-security/

[3] Scott Rose et al., "Zero trust architecture," NIST Special Publication 800-207, National Institute of Standards and Technology, 2020. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-207

[4] J. Kindervag, "Build security into your network's DNA: The zero trust network architecture," Forrester Research, 2010. [Online]. Available: https://www.virtualstarmedia.com/downloads/Forrester_zero_trust_DNA.pdf

[5] Internet Engineering Task Force, "The OAuth 2.0 authorization framework," RFC 6749, IETF, 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6749

[6] Internet Engineering Task Force, "OpenID connect core 1.0," OpenID Foundation, 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html

[7] O. E. Mohamed et al., "DevSecOps in practice: A systematic review of challenges, best practices and tools," in Proc. 2025 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11442153

[8] SPIFFE, " What are SPIFFE and SPIRE?," 2025. [Online]. Available: https://www.redhat.com/en/topics/security/spiffe-and-spire

[9] SPIRE, " SPIRE: SPIFFE runtime environment," CNCF, 2024. [Online]. Available: https://github.com/spiffe/spire

[10] Open Policy Agent, "OPA: The open policy agent," CNCF, 2024. [Online]. Available: https://github.com/open-policy-agent/OPA

[11] Mohamed Ahmed, "Policy as code with Open Policy Agent," Styra, Inc., 2023. [Online]. Available: https://www.cncf.io/blog/2020/08/13/introducing-policy-as-code-the-open-policy-agent-opa/

[12] A. Malhotra et al., "Evaluate canary deployment techniques using Kubernetes, Istio, and Liquibase for cloud native enterprise applications to achieve zero downtime for continuous deployments," IEEE Access, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10560002

[13] Istio, " Security," CNCF, 2024. [Online]. Available: https://istio.io/latest/docs/concepts/security/

[14] OpenTelemetry, "OpenTelemetry: Vendor-neutral observability framework," CNCF, 2024. [Online]. Available: https://opentelemetry.io/

[15] Kubernetes, "Kubernetes Gateway API," SIG Network, 2024. [Online]. Available: https://gateway-api.sigs.k8s.io/

[16] Kyverno, "Kyverno: Kubernetes native policy management," CNCF, 2024. [Online]. Available: https://github.com/kyverno/kyverno/

[17] L. Leite et al., "A survey of DevOps concepts and challenges," ACM Computing Surveys, 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3359981

[18] N. Forsgren, J. Humble, and G. Kim, "Accelerate: The science of lean software and DevOps," IT Revolution Press, 2018. [Online]. Available: https://dl.acm.org/doi/book/10.5555/3235404