UDC 004.056.5:004.75 DOI: https://doi.org/10.64076/iedc251023.04

Improving cloud security on AWS: zero trust for containerized services and lightweight edge anomaly detection

Vitalii Molnar
Lviv Polytechnic National University, Lviv
https://orcid.org/0009-0001-3183-0117

Abstract. *This paper presents a reproducible security blueprint for containerized applications on Amazon Web Services. The approach combines Zero Trust enforcement in an Elastic Kubernetes Service service mesh, using mutual Transport Layer Security and least-privilege, route-scoped policies, with lightweight edge anomaly detection on Amazon CloudFront and the Application Load Balancer, plus event-driven remediation. Measurements indicate a smaller attack surface and minimal overhead.*
Keywords: *cloud security, zero trust, service mesh, anomaly detection.*

As cloud workloads scale and evolve rapidly, perimeter-centric security proves insufficient against modern threats and frequent misconfigurations in public clouds [1]. This work proposes a practical, reproducible approach for AWS-hosted, containerized applications that combines a Zero Trust architecture within EKS and lightweight anomaly detection at the edge (CloudFront/ALB) with automated, event-driven remediation [2]. The goal is to minimize implicit trust between microservices and to detect traffic deviations early without heavy operational overhead, with decisions grounded in measurable security outcomes. Inside clusters, lateral movement is encouraged by over-permissive connectivity and coarse-grained authorization, while at the boundary volatile traffic regimes–from benign flash-crowds to short, intense bursts–can resemble denial-of-service behaviors and defeat static rule sets [1]. A robust solution must be measurable, CI/CD-friendly, and resilient under fluctuating loads while preserving service-level objectives for latency and throughput [2].

To ensure reproducibility, the traffic corpus is partitioned into disjoint time windows that reflect real operating regimes rather than random shuffles. Background minutes capture routine usage across diurnal patterns; controlled bursts provide short, high-pressure intervals without saturating the edge; and benign surges approximate flash-crowd dynamics that are known to trigger false alarms in naive detectors. Labels are derived from scenario schedules and cross-checked against rate and error-ratio thresholds. Features are kept deliberately compact and interpretable–minute-level request rate, unique-client cardinality, response-code proportions, and dispersion across source networks and user agents–so inference remains cheap and diagnostic explanations stay straightforward.

The architecture has four layers. At the edge, Amazon CloudFront fronts the application with AWS WAF attached; access logs are persisted to S3 for aggregation and analysis [3]. The load-balancing tier uses Application Load Balancer with access

logging to S3 [4]. The compute layer is Amazon EKS, where a service mesh enforces mutual TLS by default, assigns strong service identities to pods, and applies least-privilege, route-scoped policies that confine communication to explicitly defined paths [3], [5]. The observability and response layer consolidates telemetry (CloudWatch and CloudTrail), derives compact features, performs inference using simple models, and triggers automated mitigation when anomalies are confirmed [6].

Anomaly detection operates on one-minute aggregates that capture edge-traffic characteristics: request rate, count of unique clients, proportions of 4xx/5xx responses, and dispersion across IPs and user agents. Using transparent models such as logistic regression and decision trees keeps inference predictable and debuggable; critically, detection thresholds are fixed after validation on representative slices that cover steady background traffic, adversarial bursts, and benign surges [7]. This stabilization avoids constant retuning during spiky periods and reduces false alarms during clean intervals.

The experiment follows a fixed-threshold protocol: detector thresholds are frozen on a validation slice that spans the three regimes, and then evaluated on hold-out minutes without retuning. Two ablations clarify what drives reliability. First, the benign-surge slice is removed from the training/validation mix; the expected effect is a rise in false alarms during clean peaks. Second, dispersion indicators are removed from the feature set; this typically slows the onset of alerts on mixed traffic and increases time-to-detect. Together, these ablations justify both the feature choices and the decision to stabilize thresholds instead of auto-tuning them under load.

Zero Trust in the mesh reduces the attack surface by constraining network reachability and tying authorization to verified service identities and request context [2], [5]. Mutual TLS prevents unauthorized connections; policy guardrails ensure that only approved inter-service routes are viable. Coupling this with edge-level anomaly monitoring yields a “double safety net”: if inbound traffic drifts toward suspicious patterns, automated playbooks can escalate controls–progressively challenging clients, limiting rates, or temporarily blocking routes–without manual intervention on the critical path [3], [4]. Operationalization emphasizes policies-as-code and safe rollout. Mesh policies and edge controls are versioned and validated in CI before deployment; synthetic canaries exercise rapid rate shifts and protocol quirks; and every change is annotated in metrics and traces to attribute latency and accuracy effects unambiguously. Remediation playbooks follow a graded path–lightweight challenges and short-lived limits first; targeted blocks only if deviations persist–with automatic rollback once feature vectors re-enter the normal corridor. Cost guards are applied to logging and inference so that the blueprint remains feasible in small lab accounts while scaling predictably to larger testbeds.

To keep the evaluation focused and space-efficient, Table 1 summarizes six key KPIs used throughout the study. The metrics jointly capture overhead (latency and throughput), detection quality (precision/recall/F1 and false alarms during clean periods), and operational responsiveness (time-to-detect and the share of incidents resolved automatically). Targets are chosen to keep mesh/mTLS overhead within a strict budget while preserving timely detection during adversarial bursts and maintaining operator-free mitigation for the majority of routine incidents.

Table 1. Compact evaluation KPIs

KPI	What it captures	How it's measured	Target
p95 latency overhead	Tail impact of mesh/mTLS	Δ p95 service-to-service vs. baseline	≤ 5%
Throughput impact	Capacity under controls	Δ RPS/bitrate on control endpoints	≤ 3%
Precision / Recall / F1	Detection quality across regimes	Minute-level aggregates (CF/ALB) with labeled windows	F1 ≥ 0.95
False alarms per clean hour	Robustness in quiet periods	FP count over purely benign intervals	0–1/hr
Time-to-Detect (TTD)	Alert timeliness	Minutes from deviation to alert	≤ 1 min
Auto-mitigation share	Operator load reduction	% incidents closed automatically	≥ 80%

Using these KPIs, the evaluation reports mean and p95 latency overhead on internal calls, throughput impact, stability under load, precision/recall/F1 across three traffic regimes, the count of false alarms during clean periods, and time to detection and time to recovery for scenarios that trigger automated responses. The study also tracks the share of incidents mitigated without operator involvement and the fraction of unauthorized intra-cluster connection attempts blocked by mesh policies.

The expected outcome is a measured confirmation that representative training data matter more than model complexity for edge anomaly detection; that false-positive rates remain low during clean periods while timely detection is preserved during adversarial bursts; and that mesh-enforced mutual TLS and route segmentation introduce minimal overhead within agreed service objectives. The resulting blueprint is immediately applicable in academic labs and production-like testbeds, using managed AWS services, transparent features, and simple models to raise security posture without brittle complexity.

References

1. European Union Agency for Cybersecurity. (2024). *ENISA threat landscape 2024*. https://doi.org/10.2824/0710888.

2. Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). *Zero trust architecture (NIST SP 800-207)*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207.

3. Amazon Web Services. (n.d.). *Standard logging (access logs) – Amazon CloudFront developer guide.* https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/AccessLogs.html.

4. Amazon Web Services. (n.d.). *Access logs for your application load balancer.* https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-access-logs.html.

5. Chandramouli, R., & Rose, S. (2020). *Building secure microservices-based applications using service-mesh architecture (NIST SP 800-204A).* National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204A.

6. Amazon Web Services. (n.d.). *What is Amazon CloudWatch?* https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html.

7. Pinto, A., Herrera, L.-C., Donoso, Y., & Gutierrez, J. A. (2023). A survey on intrusion detection systems based on machine learning techniques for the protection of critical infrastructure. *Sensors*, *23*(5), 2415. https://doi.org/10.3390/s23052415.