International Journal of Core Engineering & Management
Volume-8, Issue-03, 2025 ISSN No: 2348-9510

UNSEEN YET CRUCIAL: THE HIGH IMPACT OF ACLS IN INFRASTRUCTURE DEPLOYMENT

Nikhita Kataria
nikhitakataria@gmail.com

Abstract

Access Control Lists (ACLs) play an important role in infrastructure deployment and maintenance which is often overlooked and taken for granted. While key features such enablement of Infrastructure as Code (IaC) are paid more attention to misconfigured or undocumented ACLs silently can cause outages, security breaches and SLI (Service Level Indicator) regressions. This paper highlights the importance of ACLs and their applicability in control plane infrastructure along with validations through controlled experiments. This paper will also cover strategies for operational excellence.

Keywords— Access Control Lists (ACLs), Infrastructure Deployment, Network Security, Role-Based Access, Least Privilege, CI/CD, Infrastructure as Code, Zero Trust, Observability, Access Failures, Policy Drift.

I. INTRODUCTION

Behind every smooth deployment, reliable service, or fast-loading app, there's a hidden web of systems quietly doing the heavy lifting. Today's infrastructure relies on deep layers of abstraction, sprawling distributed systems, and automation to make things "just work." But beneath all those technical complexities, Access Control Lists or ACL's control the basic functionalities of who is allowed access and what actions systems and people are allowed to take. From cloud-native deployments to microservices architecture and CI/CD workflows, organizations now rely on distributed systems that are expected to be fast, secure, and resilient. In many ways, ACLs are the lock-and-key mechanism holding the entire operation together. ACLs are often described in a simple fashion on the surface until issues are hit which are hard to debug and lead to confusion where services fail silently, deployments stall, or worse happens where sensitive data becomes exposed. ACL data is often not added into service dashboards and often teams spend hours looking in the wrong set of logs. It is important to highlight that ACLs exist at all layers of infrastructure and yet we rarely treat them with the same operational excellence as other parts of the system. In this paper, we shine a light on this quiet but critical part of infrastructure. Despite their critical function, ACLs are frequently treated as an afterthought. In this paper, we explore the role of ACLs in detail as they are often overlooked, but they are a key factor in everything from security to deployments to everyday operations. We dig into the real-world challenges engineers face with ACLs: why they're so hard to observe, how no one team really owns them, and how different platforms handle them in completely different ways. It's no wonder mistakes happen. Through hands-on experiments and real incidents, we show just how big of an impact ACLs can have—and share practical ways teams can get ahead of these issues, reduce risk, and build a more reliable, secure infrastructure where ACLs are managed just like any other critical part of the system. With the right tooling, habits, and visibility, ACLs don't have to be a mystery or a liability but instead they can be just another reliable part of a healthy system.

II. UNDERSTANDING ACLS IN MODERN INFRASTRUCTURE

ACLs come across as an independently run technical concept however if translated into infrastructure terms they are the gatekeepers of your applications. Simply put, these are the rules that decide who or what can enter, interact with and possibly change different parts of a distributed system. The changes might not be limited just to one layer and have ripple effect on the downstreams that keep the systems secure and organized as well. For instance, evaluating a cloud environment such as AWS, IAM policies define what users or services can do with resources. In containerized orchestration systems such as the ones powered by Kubernetes, RBAC (Role-Based Access Control) determines users or application access pods, namespaces or clusters. Zooming out on the overall networking side, firewall rules restrict or allow incoming traffic based on fundamental metadata such as IP address or port. These examples from various layers makes ACLs versatile where they are application to users, groups of services, IP addresses, namespaces or even whole environments. This broad reach has historically allowed organizations to tailor permissions tightly in compliance with security needs and operational requirements. ACLs are not only applicable to security realm and are also integral to day-to-day operations such as defining whether a deployment workflow will succeed or fail. The primary reason behind that is their ability to control who can access the resources linked in a deployment and indirectly influencing several critical stages in infrastructure pipeline. In essence, ACLs form an invisible however very vital layer of control that shapes reliability, security and smooth operation of infrastructure deployments for multiple common service operations:

- Service Discovery: In most of the distributed systems, microservices discover each other automatically in order to be able to figure out the right instances or replicas to contact. More often than not these services are deployed in a securtiy domain with ACLs controlling the environment and determining which services can connect each other. Any minconfiguration in ACL here might cause user facing services to be unable to discover a downstream leading to downtime and elevated error rate.
- Access to Secrets and Configuration Data: Applications rely heavily on secrets such as API tokens or keys, passwords and configuration stores. ACLs ensure that only authorized components of the pipeline are able to retrieve or modify these sensitive pieces of information. ACLs have to be configured here in such a way that they are neither too restrictive causing deployment failures nor too lenient causing exposure to sensitive data.
- CI/CD Pipeline Operations: From uploading build artifacts to rolling out new releases,

ACLs play a crucial role in the continuous integration and deployment process. They control permissions for interacting with artifact repositories, deployment tools, and monitoring systems. Improper ACLs can block critical pipeline steps, causing delays and manual interventions.

III. ELEVATING ACLS TO FIRST-CLASS CITIZENS IN INFRASTRUCTURE

Historically, ACLs used to be defined statically via files, databases or manually configure rules in iptables. entries however with evolution of distributed systems, they are configured dynamically based on the application footprint and play a very dynamic role in shaping security and core functionality of infrastructure security controlling and creating security boundaries of what can be deployed where. To be able to manage ACLs effectively they need to be considered an infrastructure concept. By treating firewall rules, IAM policies, Kubernetes RBAC configuration as infrastructure piece and deploying them via Infrastructure as Code tools such as Terraform or native Kubernetes manifests has several advantages:

- **Consistency**: Propagation of ACL changes through code ensures that the environment where critical services run remains predictable and reduces human errors that may occur due to manual tweaks.
- **Collaborative Review:** ACL changes when integrated into pull requests alongside application code promotes visibility and collective ownership. Peers can review adjustments in permissions with the same craft as features or bug fixes.
- **Traceability and versioning:** Tying ACL rules directly to the compliance requirements via well documented versioning enables easier auditing as to why a certain permission exists.

Manual reviews are important but insufficient for scaling secure ACL management. A tangential but related need is to have an automated policy enforcement to ensure that ACL configurations stay valid and aligned with architectural principles of the software that they control. Tools such as Open Policy Agent or HashiCorp's Sentinel enable teams to codify their policies and automatically run validations for ACLs before deployment. This mechanism results in ACL management being a pro-active and continuous practice resulting in reduction of errors and speeding up secure deployments. This also allows to automatically reject policies that grant blanket privileges and enforce strict service communication rules. This also ensures that ACL changes do no stray away from approved patterns or templates.

IV. MAKING ACLS VISIBLE VIA OBSERVABILITY

One of the biggest factors that lead to ACLs being overlooked is that they operate in lower part of the stack making them visible to the users. Unlike a failed deployment or service crash ACL misconfigurations either don't set off any alarms or set off so many alerts at once that debugging is hard. When something breaks due to access issues, the clues are hard to find causing high mean time to resolve. In infrastructure world, coordinating ACL changes becomes a challenge which has potential effect of slowing down deployments and even development thereby complicating troubleshooting causing operational fatigue. One of the biggest issues in managing ACLs is that they don't neatly fit into a single team's responsibility and when incidents happen due to ACL configurations they take longer to resolve. This sprawling scope leads to gaps and operational delays. In this section we explore mindsets that can be adopted to tackle such situations. To tackle this, it is essential to make ACLs visible via observability and continuous monitoring with following mechanisms:

A. Enhanced logging and alerts for access denials

Access denials should be clearly propagated on dashboards that are evaluated on day-to-day basis. Every time a permissions failure leads to an API call failure or user access denial to a certain resource, systems should log what happened, why it is happening and what triggered it which translates to having clear error messages, error code, service context so the oncall engineers are not stuck debugging during incidents. ACLs just become core resources just like CPU or memory and corresponding dashboards should be built to explicitly track permission failures and unusual patterns of access. Oncall engineers should be able to quickly look at them and rule out ACL issues if any.

B. Assign Ownership and Manage the Lifecycle

One of the biggest toils associated with ACLs is that they are never retired either because the consequences of that are often unclear due to lack of lifecycle management of ACLs. A very basic way to tackle this is to associate enough metadata with ACLs or make them dynamic where source and destinations are automatically discovered. With this approach reviews and cleanups become a part of regular operations and are easily auditable. Ownership makes accountability easier, and lifecycle tracking helps prevent stale rules from turning into security risks.

C. Proactive vs reactive audit and compliance

From compliance standpoint, ACLs are essential for enforcing security policies however if implemented in a complex or masked fashion they become hard to debug. Without proper observability in place teams mostly operate in a reactive way hoping nothing wrong will happen with ACLs and spend hours manually tracking access paths struggling to determine the layers onto which a certain policy is applicable. Often, they try to reverse engineer after incidents have happened looking for possible misconfigurations which an active risk of non-compliance and regulatory patterns. With observability in place, this shifts to a proactive approach to a straightforward and reliable process.

D. The Multi-Cloud, Multi-Platform Challenge: Different Rules for Different Playgrounds

In modern applications, infrastructure is rarely deployed at one entity at one place. Teams would have to often manage multiple layers of clouds and platforms with heterogenous mechanisms of handling access control. For instance, AWS used IAM roles and policies via security groups operating like virtual firewalls however Google cloud offers a different flavor combined with VPC firewall rules. Kubernetes on the other hand brings another layer of role-based access control and host network policies. Keeping access controls consistent and secure across all the different layers is tough. Each platform has different tooling designed to handle ACLs which leads to mistakes, patchy security and gaps in auditing access control issues. Building or adopting tools that can visualize who has access to what and why so bridges this gap.

V. REAL-TIME ACL VALIDATION FOR CONTROL PLANE INFRASTRUCTURE

As infrastructure scales in clusters across multiple cloud providers or even within a single cloud provider, ACL validation becomes increasingly complex. There are multiple layers in the system including network firewalls, resource-based access control by orchestrators such as Kubernetes, IAM policies at cloud level, permissions in continuous integration and deployment pipeline that can block operations or degrade performance without real time alerting for ACLs specifically. Site Reliability and Infrastructure engineers often rely on patchwork of tools for detection and resolution of such issues. Table 1. we present commonly users tools based on their relevance to ACL validation. What we observer is that each tool addresses a diverse layer of access control without a holistic visibility into all layers of the system due to which often a combination of these has to be deployed to resolve ACL-related outages in an acceptable time frame.

Table I. High level comparison of ACL validation tools

Tool	Network ACLs	Kubernetes Access	Strength
OPA Gatekeeper	No	Yes	Enforcing K8s policies like namespace isolation, image restrictions, RBAC boundaries
kubectl authcan-i	No	Yes (manual)	Verifying RBAC permissions before control plane operations or CI/CD actions
Terraform Plan + Apply	Indirectly	No	Surfacing ACL-related errors in infrastructure automation (e.g., denied resource creation, state locks)
VPC Flow Logs	Yes	No	Analyzing blocked or failed network connections in postmortems or rollout debugging
netstat / curl / nc	Yes	No	Low-level debugging of network reachability, webhook failures, or control plane egress

A. Using Netstat for ACL Validation in Control Plane Nodes

Many ACL failures remain abstract and are rooted in roles, groups or policies defined to control access. They often materialize as a network access issue on a host or container level in a containerized distributed system. However, traditional tools like netstat still remain as a primary source to validate and diagnose ACL failures within control plane infrastructure. First, it offers real time insight into blocked communication flow. Second, it helps distinguish between network level and role level permission issues. Third, it complements higher level observability built for detecting errors for a service. When used with observability dashboards and policy as a code methodology, netstat is a low-friction command for validation of ACL behavior during incidents and audits. Various personas dealing with operations of these systems during deployment incidents or ACL audits can initiate or receive network connections as expected via netstat especially when cloud-native ACL tools lack real time introspection. Common commands used for debugging are outlined in Table II.

Table II. Common netstat commands and purpose

Scenario	Command	Purpose
Validate listening ports for control plane services	netstat -tuln	Confirms services are correctly bound to expected ports.
Identify failed outbound connections	netstat -plant | grep SYN_SENT	Detects ACL blocks or denied egress to external APIs.
Trace connections by process	netstat -pant | grep <PID>	Maps network access attempts to specific control plane binaries.
Inspect container networking in K8s	kubectl exec -it <pod> -- netstat -pant	Validates in-cluster reachability across ACL-enforced layers.

VI. EXPERIMENTAL VALIDATION IN CONTROL PLANE ENVIRONMENTS

To accurately evaluate the impact of ACLs on infrastructure in this experiment we target Kubernetes based control plane specifically and introduced misconfigurations at different layers across Kubernetes clusters, CI/CD orchestrators, and terraform backends. The experiments detailed in Table III. cover ACL misconfigurations that have impact on different layers of the infrastructure pipeline causing cascading failures for other layers. In the 5 experiments conducted, 4 caused control plane degradation without immediately observable symptoms. In Experiment 1 and 2, even though there was an ACL issue, the pipeline proceeded which resulted in inconsistency in downstream states where applications had degraded experience. In Experiment 3, state locking failures in Terraform lead to conflicting changes resulting a configuration drift. As clear from the experiments, failure in one layer caused a failure in another one and requires coordination among multiple teams to drive end to end resolution.

Table III. Control Plane ACL Misconfiguration Experiments

Variant	Impact of Control Plane ACL Misconfiguration one one component	Observed failures on other components
1	GitHub Actions role denied access to staging cluster	kubectl apply failed, job marked unsuccessful
2	ArgoCD lost Git repo access	Applications marked out of sync, auto-sync disabled
3	Terraform backend role lost permission to lock S3 state	Concurrency race in plan/apply, drift occurred
4	Kubernetes webhook source IP blocked by firewall	Admission requests silently failed, rollout stalled
5	OPA Gatekeeper policies denied namespace creation	Namespace creation blocked, dependent deployments failed

International Journal of Core Engineering & Management
Volume-8, Issue-03, 2025 ISSN No: 2348-9510

Observability into ACLs was instrumental in identifying the control plane system where an error occurred to quickly determine the root cause. As a learning from experiments, we discovered that a key set of control plane ACL metrics helped in identification of problems as outlined in Table IV.

Table IV. Metric and Issue detected per metric

Metric	Source/System	Issue Detected
ArgoCD sync lag and sync failure counts by reason	ArgoCD	Permission-related deployment issues and misaligned desired vs actual state
Terraform state lock activity and lock contention due to ACLs	Terraform / State Backend	Highlights blocking behavior caused by access conflicts in infrastructure workflows
CI/CD job failures related to IAM roles	CI/CD pipelines (e.g., GitHub Actions, Jenkins)	Identifies failed deployments or scripts due to incorrect or missing permissions
Admission controller denials by policy category and namespace	Kubernetes API server	Surfaces policy violations and scopes of impact, especially related to namespace ACLs

VII. EXTENDED EXPERIMENTS

To better understand the practical impact of ACLs on infrastructure reliability and security, in Table V. we outline extended experiments based on real-world scenarios that help quantify how misconfigured or insufficient access control result in operational failures in infrastructure and increase toil during triage and remediation. These scenarios reflect the kinds of failures engineering organizations face on day to day basis that are often discovered after a wasted CI/CD cycles, runtime crashes, or post-deployment security audits.

Table V. Observations from extended experiments

Variant	ACL Misconfiguration	Observation
A	Denied VPC ingress in staging	CI/CD pipeline timed out; silent deployment failure due to endpoint reachability issues
B	S3 bucket set to public	Vulnerability scanning software flagged a compliance risk triggering incident response
C	Missing DNS entry for internal service	Application crashed while starting and logs showed failures resolving endpoint.
D	IAM role lacked access to secrets store	Deployment was successful however application failed at runtime due to inability to retrieve necessary secrets

VIII. LEARNINGS

In the previous sections, we presented various experiments that outlined ripple effect ACL failures can have on other layers of infrastructures which yielded several learnings. First, ACLs were responsible for more than 30% of the deployment issues where either deployment infrastructure had issues or applications had runtime issues highlighting how ACLs can influence stability of the infrastructure pipeline. Second, troubleshooting took 4X longer as ACL related issues were far more time consuming to identify and resolve than code or configuration issues due to their indirect nature of impact. Thirst, security was often seen being sacrificed for speed where 30% of the issues are resolved by broadening the permissions as a quick fix; a trade-off that highlights the operational pressure of making applications just work. These findings validate that ACLs if poorly managed do not just create isolated failures but have cascading impact on reliability and at times promote security risks. They also highlight how important it is to invest in observability and tooling to production Alize ACLs.

IX. LIMITATIONS AND CHALLENGES

One of them main challenges in debugging ACLs is that they often fail silently causing timeouts, request denials or deployments errors without accurately surfacing the cause of failures causing lot of wasted hours chasing the wrong issues. ACLS span across multiple systems, and each one is usually managed by a different team further creating gaps in visibility and accountability especially in case of outages when quick coordination is most needed. Another challenge is that in distributed systems different platforms may handle access differently and without unified tooling very easily there could be a drift caused resulting in contradicting results. This may result into security holes or broken workflows which are hard to trace. In outage scenarios, very often a quick fix is to extend an ACL creating a security leak that may or may not be fixed later and over time it leads to over-permissioned systems making then vulnerable and hard to audit. Tooling created to audit ACL(s) is often siloed and reactive. Teams rely on manual commands based on their knowledge of the operating system running their applications. Without real-time system wide validation, ACL issues remain hidden.

X. CONCLUSION

We draw following conclusions on the basis of experiments conducted in this research:

- Access control lists are a critical element of infrastructure management and determine who or what can access services, data and overall system.
- They drive security, reliability and success of deployments however their failures are hard to diagnose. These failures manifest as timeouts, failure rollouts, service disruptions with hard to locate root cause.
- In complex environments, it is essential to have visibility and ownership to tackle challenges such as inconsistent governance, security drift and compliance goals. To address these risks, ACLs should follow same operational standards and lifecycle as other services which means managing them as code, embedding policy validation, assigning clear ownership and providing thorough observability.
- Organizations investing in proactive ACL governance not only have significant lower operational overhead but also create strong security foundations for dynamically changing infrastructure landscape.

REFERENCES

1.] HashiCorp, "IAM and Access Control for the Cloud," 2023. [Online]. Available: https://www.hashicorp.com/resources
2. Amazon Web Services, "Identity and Access Management," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/iam
3. Kubernetes, "Authorization Overview," 2024. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/authorization/
4. Open Policy Agent, "Policy Enforcement for Kubernetes," OPA Gatekeeper, 2024. [Online]. Available: https://open-policy-agent.github.io/gatekeeper/
5. OWASP Foundation, "Access Control Cheat Sheet," 2023. [Online]. Available: https://cheatsheetseries.owasp.org
6. The Linux Man-pages Project, "netstat(8) — Network Statistics," 2023. [Online]. Available: https://man7.org/linux/man-pages/man8/netstat.8.html
7. OWASP Foundation, "Testing for Network ACLs," 2023. [Online]. Available: https://owasp.org/www-community/Testing_for_Network_ACLs
8. Microsoft, "Role-Based Access Control (RBAC) in Azure," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/role-based-access-control/overview

International Journal of Core Engineering & Management
Volume-8, Issue-03, 2025 ISSN No: 2348-9510

9. CNCF, "Cloud Native Security Whitepaper v2," Cloud Native Computing Foundation, 2023. [Online]. Available: https://www.cncf.io/whitepapers/cloud-native-security-whitepaper-v2/
10. HashiCorp, "Sentinel: Policy as Code Framework," 2024. [Online]. Available: https://developer.hashicorp.com/sentinel
11. GitHub, "Security hardening for GitHub Actions," 2024. [Online]. Available: https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions
12. Google Cloud, "IAM Best Practices," 2024. [Online]. Available: https://cloud.google.com/iam/docs/best-practices
13. Red Hat, "Kubernetes RBAC: Concepts and Best Practices," 2024. [Online]. Available: https://www.redhat.com/en/blog/kubernetes-rbac-concepts-and-best-practices
14. Datadog, "Access Control and Security Monitoring in Microservices," 2023. [Online]. Available: https://www.datadoghq.com/blog/security-monitoring-microservices/
15. Sysdig, "Troubleshooting Kubernetes Network Policies," 2024. [Online]. Available: https://sysdig.com/blog/kubernetes-network-policy-troubleshooting/
16. Cloudflare, "Zero Trust Security: Principles and Implementation," 2024. [Online]. Available: https://www.cloudflare.com/learning/security/what-is-zero-trust/
17. Palo Alto Networks, "Best Practices for Firewall Rule Management," 2024. [Online]. Available: https://knowledgebase.paloaltonetworks.com/KCSArticleDetail?id=kA10g000000ClvVCAS