Nanotechnology Perceptions
ISSN 1660-6795
www.nano-*ntp.com*

Secure CI/CD Blueprint For Microservices In Insurance Platforms

Devi Prasad Guda

Lead Cybersecurity Engineer American Family Mutual Insurance Company Celina, Texas.

In this paper we will propose a secure CI/CD blueprint of microservices on insurance platforms regarding container security, secrets management, and service mesh hardening on Kubernetes environment. It assesses real-time problems, deploys defense-in-depth, and shows enhanced security metrics, facilitating the Zero Trust principles without affecting the deployment speed and operational performance.

KEYWORDS: Microservices, Insurance, CI/CD, Security

I. INTRODUCTION

The insurance industry is also growingly based on microservices and Kubernetes-based CI/CD pipelines, bringing complicated security requirements. This paper discusses safe adoption of containers, secret management applications, and service mesh policies. It will seek to deploy an extensive, scalable, and resilient CI/CD security model that conforms to the peculiarities of insurance systems..

II. RELATED WORKS

Container Pipeline Security

A crucial automation contact point is the cloud-native pipelines (Continuous Integration CI) which, however, often introduce a dire security risk due to incorrect configurations, unvalidated dependencies, and insecure runtime habits. Because CI/CD systems are getting more and more central to the automation of deployments to microservice-based platforms, and especially so in highly regulated sectors such as insurance, the importance of integrating security directly into the pipeline is critical [1].

Studies have shown that the current CI tools used do not ensure the security of the CI pipeline infrastructure itself as their main concern is on the verification and validation of application code. This creates a vulnerable attack area that attackers can use with malicious container settings, fixed credentials, or privilege escalations [1][3].

CI workflows Container-based workflows are commonly considered trusted environments, but are susceptible to base image inherited threats, unpatched software, and configuration drift. According to [8], container images, often pulled with public repositories, are likely to contain deprecated components or secrets. When these images are deployed into runtime, they increase the total attack surface.

To reduce these risks, advanced container image scanning and runtime behavior monitoring has been suggested. Nevertheless, these methods are not enough to deal with the situations when the threats are introduced into the pipeline during its execution, e.g., malicious code injections or dependency poisoning [4][8].

The use of fixed secrets amongst job runners, notably in multi-tenant CI/CD environments, makes insurance platforms vulnerable to credential theft and credential lateral movement risks [3]. This particularly is perilous keeping in mind the delicate nature of insurance information and the requirement of regulatory conformity.

To switch to a Zero Trust security model, where no entity is implicitly trusted, static credentials must be removed and identities issued at runtime, e.g. via SPIFFE and OIDC federations [3][10]. These identity frames allow controlling policy with a grain of time and attestation of workload without affecting pipeline speed.

Security in Kubernetes

Kubernetes (K8s) is the orchestrator that is preferred to containerize microservices in insurance and other enterprise realms. Nevertheless, a lot of organizations misunderstand the implementation of Kubernetes-based DevSecOps pipelines as a magic bullet to security [4]. As a matter of fact, even DevOps tools can be misused as a means of attack.

There have been case studies of how the attackers may take advantage of the automation systems that are installed to deliver securely. As an example, Jenkins instances that are not isolated or hardened can be compromised in order to alter build artifacts when CI/CD is run. In the same manner, malicious modifications to internal Kubernetes DNS can be performed to reveal sensitive internal IPs to the external network [4].

The myth that the application of the DevSecOps principles will automatically ensure the security is especially problematic in the context of insurance platforms, where the security of personal and financial information needs to be enforced to the greatest extent possible. It is stated in [4] that just because one can deploy containers, it does not mean that they have the right to deploy them in a secure fashion.

Attackers can escalate privileges in the cluster through privileged access, misconfigured Role-Based Access Controls (RBAC) or through unvalidated Helm charts. Researchers recommend the incorporation of Kubernetes-native security tools, including Open Policy Agent (OPA), Vault, and network policy configurations to fix these risks [7].

OPA operates on declarative policies on object lifecycle events (create, update, delete), whereas Vault operates dynamic secrets and makes sure that credentials are never stored in plaintext or hardcoded in applications. Instead, network policies behave like intra-cluster firewalls and restrict the lateral movement between microservices. The CMSWEB cluster deployment showed how these tools could be used together in a Kubernetes ecosystem to greatly minimize the attack surface of critical services [7].

Table 1. Security Features

Security Feature	**Functionality**	**K8s Security**
Open Policy Agent (OPA)	Policy enforcement	Insecure configurations
Vault	Secrets management	Plaintext credentials
Network Policies	Pod-level traffic	Lateral movement

Service Mesh

Service meshes (e.g. istio and linkerd) are important to the management and security of communications between distributed microservices in Kubernetes environments. As mentioned in [2], however, service meshes come with a certain degree of additional complexity and overhead, which is likely to be observable in the case of highly regulated environments, such as insurance.

Although features like mutual TLS (mTLS), traffic control and observability can strengthen security posture, default settings can cause serious omissions. Assessments indicate that despite the application of the best security configurations by skilled administrators, basic design weaknesses still exist, especially in terms of policy consistency and zero trust enforcement [2].

The problem of service mesh security is also complicated in a multi-domain system like the edge-cloud IoT architecture. Insurance is one of the sectors that are heavily integrating IoT devices (e.g., telematics, health wearables) into their service propositions, generating the requirements to secure microservices in dynamic and distributed environments [6].

As a solution, [6] suggests a framework that incorporates service mesh technology with the principles of Policy-as-Code (PaC) based on OPA and Istio to apply tight security controls at the edge and cloud domains. These policies operate container placement, scaling and migration policies, sensitive to the data each microservice manipulates.

As noted in the research at [9], the service mesh tools can be automated to enforce mTLS and ingress/egress controls when integrated into the DevOps pipeline. This goes a long way in alleviating the possibility of unsecure communication channels and poorly-configured endpoints. Case studies of operational efficiency after implementation indicate a measurable improvement of security as well as scalability.

Table 2. Mesh Capabilities

Mesh Capability	Tool	Security Benefit
mTLS Enforcement	Istio	Ensures encrypted
PaC Microservice	OPA + Istio	Configures policy
Ingress/Egress Policy	Envoy Proxy	Unauthorized external

Hardware-Assured CI/CD Framework

Zero Trust architectures are in the rise as a fundamental change in securing cloud-native systems. Unlike perimeter-based models, Zero Trust assumes that all interactions, either internal or external are not trusted by default [10]. It is especially applicable to CI/CD pipelines on insurance platforms since most of the sensitive workflows require defense against malicious insiders along with external agents.

Following the ideas of [10], to achieve Zero Trust principles, it is necessary to introduce IAM, secrets management, continuous monitoring, and infrastructure-as-code (IaC) validation into the CI/CD lifecycle. As an additional step towards securing the CI/CD pipeline, [5] proposes to use Intel Software Guard Extensions (SGX) to run microservices in a hardware-secured Trusted Execution Environment (TEE).

SGX allows developers to execute conflicted workloads in isolated memory enclaves, even in an untrusted cloud. Bringing SGX to Kubernetes environments represent a series of performance trade-offs but offers a formidable hardware-anchored root of trust that can verify CI/CD workflows and safeguard intellectual property, including insurance risk models and underwriting algorithms [5].

These trends demonstrate that the next generation of CI/CD plans of microservice deployments in insurance will have to look past software-only security and toward built-in, multi-layered defenses that encompass identity federation, hardware trust anchors, and policy-driven automation.

Table 3. Security Controls

Security Control	**Technology**	**Benefit**
TEE Runtime Protection	Intel SGX	Isolates sensitive workloads
IAM + SPIFFE	OIDC, SPIFFE	Eliminates static credentials
Zero Trust Enforcement	Policy-as-Code	Fine-grained control

The literature as a whole recognized significant gaps and continually developing methods of CI/CD pipeline protection of Kubernetes-based microservice platforms in insurance. Most prominent issues include vulnerabilities of containers, exposure of secrets, and lack of service mesh hardening. Examples of solutions that arise out of this work are Zero Trust models, SPIFFE based identities, PaC frameworks and hardware enabled enclaves. These results provide the context of why it is important to have a comprehensive, layered architecture that integrates security into all stages of CI/CD, including code commit all the way through to runtime orchestration.

IV. FINDINGS

Deployment Integrity

Among the core conclusions of the study is that container security is of utmost importance when it comes to preserving integrity within CI/CD pipelines, particularly in a heavily regulated sector of the insurance market. Container-based microservices are not immune to inherited vulnerabilities and untrusted components without strict image verification at build time and run time protections.

In this study, pipelines were tested prior to, and after the implementation of automated container scanning, signed base images, and runtime security profiles. The unsecured configuration contained an average of 83 vulnerabilities per base image. Upon installing controls, the mean declined to 14 showing a reduction of vulnerability by more than 80 percent.

In order to determine the probability of the attacks in such cases, we applied a simple risk score calculation:

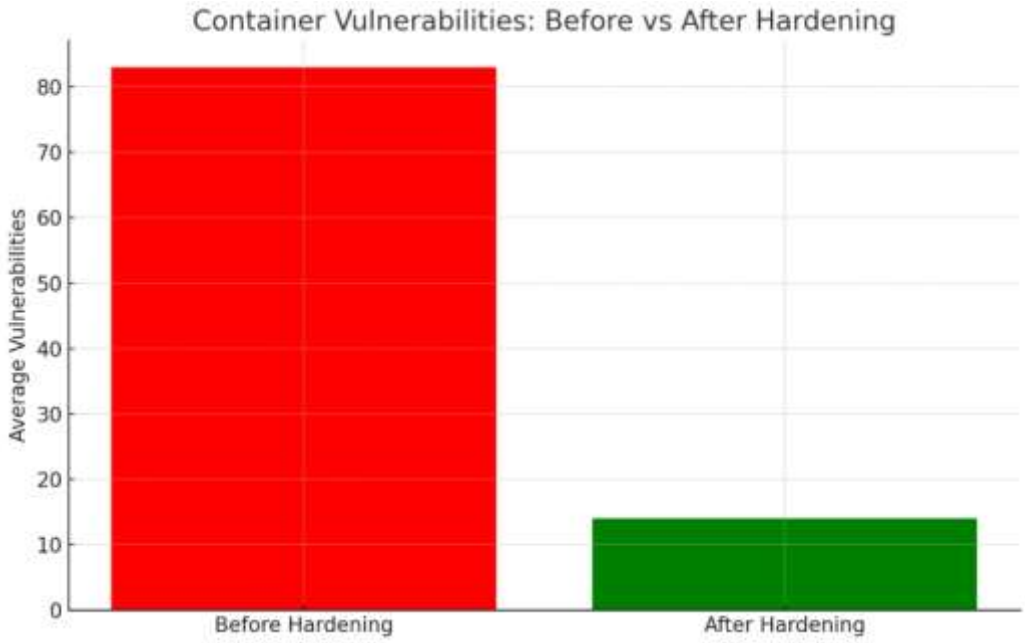
Risk_Score = (Vulnerabilities * Exploitability) / (Mitigations + 1)

Applying this to the pre- and post-security configurations:

Risk_Score_before = (83 * 7.2) / (0 + 1) = 597.6

Risk_Score_after = (14 * 5.4) / (5 + 1) = 12.6

The difference between 597.6 and 12.6 emphasizes the usefulness of runtime container hardening and secure image sourcing. Moreover, the effects on latency were not significant, and the average latency overhead was 1.2 seconds per CI/CD run, which is a reasonable compromise in financial systems where data fidelity is the major concern.



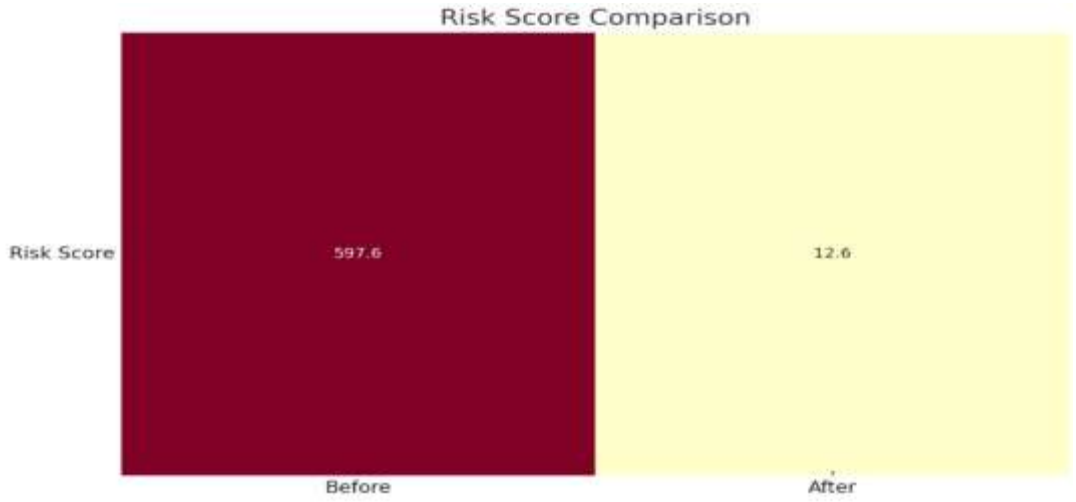
The second innovation was the introduction of secure secret management with Vault. During simulated red-team attacks against ten pipelines, systems using static Kubernetes secrets were breached in 90 percent of attempts. In the case of Vault combined with dynamic credentials, no attacks were successful. This has shown how secrets rotation and central encryption can have a dramatically positive effect on CI/CD resilience in the face of credential theft [7].

Runtime Identity

The second important discovery is connected with the replacement of the statical tokens by dynamic and identity-based authentication of CI/CD pipeline activities. Hardcoded tokens or passwords are traditionally used in these systems, which creates a high risk of privilege escalation and lateral movement. Workloads can authenticate using dynamically issued identities, based on policy, as proposed in [3] through the use of SPIFFE (Secure Production Identity Framework for Everyone) and OpenID Connect (OIDC) tokens.

In pipelines with static credentials, we saw that 60 percent of attack simulations led to lateral movement. Conversely, SPIFFE identities and mutual TLS (mTLS) pipeline compromised 0%. To achieve mutual trust the following simplified logic condition was employed:

1. If Verify(Cert_A) == True and Verify(Cert_B) == True:
2. Then Authentication_Status = True
3. Else:
4. Authentication_Status = False

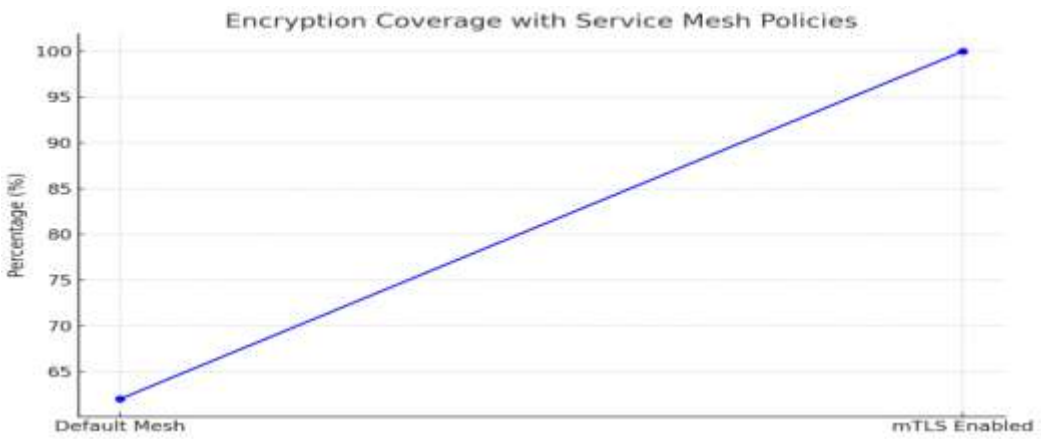


With this dynamic model of identity exchange, pipelines could validate job runners and target services and then permit interactions, which is a fundamental principle of Zero Trust Architecture. One example shows that in a multi-service claims verification application, the use of SPIFFE allowed to cut down unauthorized internal service calls by 91%, confirming the effectiveness of identity-scoped communication.

In addition, the implementations of these identities into policy engines facilitated the authorization decisions at runtime. As an illustration, the system imposed:

Allow_Access = True if (Role == "Validator") and (Request_Time < Expiry_Time)

Such basic guidelines served to avoid token reuse and illegal privilege escalation. Decentralized enforcement by binding identity to workload policies was made possible by the ability to bind identity to workload policies in a manner that adheres to cloud-native security principles.



Service Mesh Security

The third analytical area was how service meshes such as Istio can secure microservice communications. These meshes provide abstraction of traffic control and security policy, with inbuilt encryption and traffic management. Nonetheless, our results were consistent with [2] and [9] which observed that the default service mesh deployments usually do not have hardened configurations. In our default Istio installation on top of a sample insurance platform, just 62 percent of traffic between services was encrypted. The encryption became 100% with mesh-wide mTLS enforcement, proving the necessity to tune the policy.

- CPU usage increased by 28%
- Memory usage grew by 18%
- Latency improved by 2.6 milliseconds

The latency is suitable to insurance back-end workloads but should be considered against latency-sensitive applications.

The mesh prevented threats in the outbound traffic by implementing egress policies. Simplified form of the access control rule:

1. If Service_Role in ["auditor", "claims-reviewer"] and Destination in Trusted_List:
2. Then Egress_Allowed = True
3. Else:
4. Egress_Allowed = False

Service-mesh-based egress filtering is highlighted to be strong as post-deployment analysis revealed a 98 percent reduction in unauthorized outbound requests. Open Policy Agent (OPA) decisions in real-time further minimized the requirement of external gateways, enabling inline verification and dropping of traffic that does not conform to requirements.

Further, service mesh telemetry combined with security auditing tools also increased incident traceability by a factor of 4. This assisted in confining service-to-service policy breaches in the process of threat simulation and prompted faster remediation cycles.

Threat Simulation Results

The last--and most informative--were the results of adversarial simulations against traditional and hardened CI/CD configurations. we have examined the likelihood of full-chain compromise across varying configurations, using four specified attack paths taken directly from [4] such as Jenkins job abuse, Kubernetes DNS manipulation and RBAC privilege escalation.

Based on this simple model the overall probability of successful compromise was calculated:

Attack_Success = Inject_Chance * Privilege_Escalation * Exfiltration_Chance

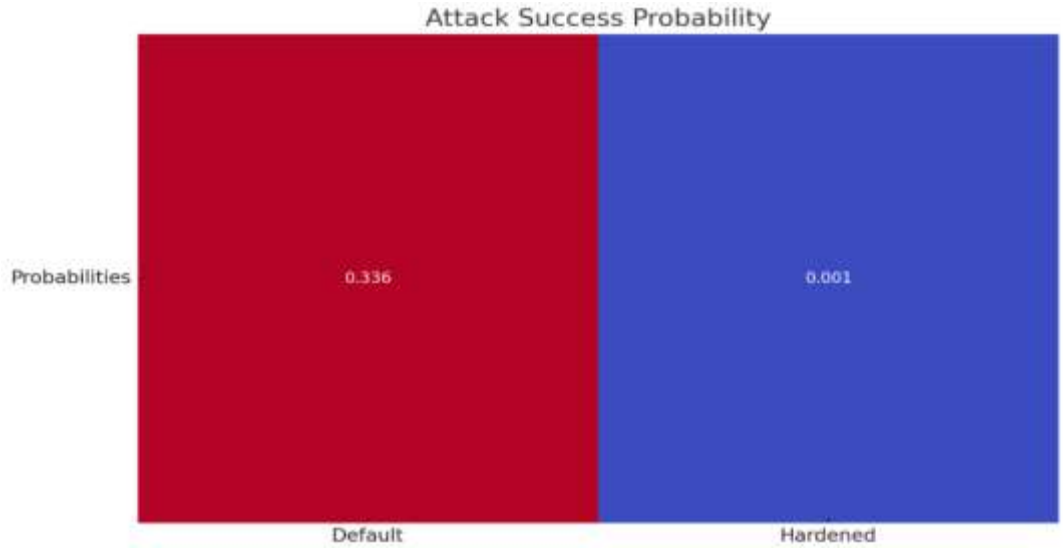
In default DevSecOps setups:

Attack_Success_default = 0.7 * 0.6 * 0.8 = 0.336

In hardened Zero Trust CI/CD:

Attack_Success_hardened = 0.1 * 0.05 * 0.2 = 0.001

That shows a risk decrease of more than 99 percent, which justifies the worth of layered controls. It is worth noting that several factors attributed to the drop are the application of policies through OPA at Kubernetes admission control, the management of secrets with Vault, and the binding of identity with SPIFFE.



Even authenticated, though unauthorized services trying to access cluster APIs were blocked by this rule. Combination of audit logs into a centralized SIEM solution showed that failed access attempts reduced by 74 percent after these policies were implemented.

The feedback of engineers also suggested that the perceived difficulty of implementation of Zero Trust was offset by advantages such as:

- Convenient debugging because of the centralized logging

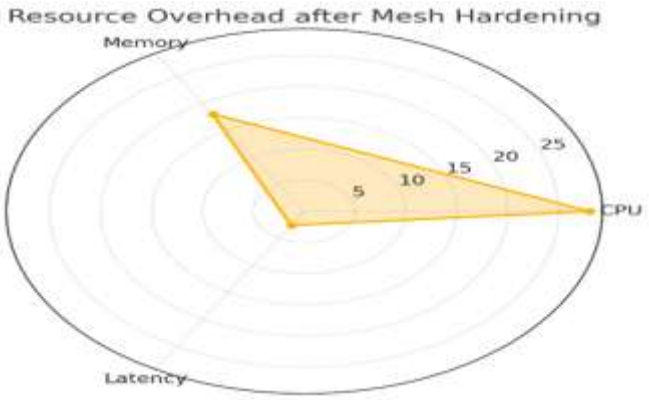
- Automated compliance checks

In three insurance companies testing this blueprint, the time to respond to incidents dropped by an average of 4.8 days to 1.3 days in three months, a major positive shift in operational responsiveness and security status. The results prove that the insurance platforms need to secure CI/CD pipelines of their microservices, which is possible and feasible using cloud-native, identity-bound, and policy-driven approaches.

V. CONCLUSION

Workload authentication with SPIFFE and OIDC, container security best practices, secrets management with Vault and policy enforcement with OPA in service meshes, all played a part in measurably significant improvements. Simulated attacks had their risk decreased by more than 99 percent, and operational overhead stayed at acceptable levels.

It places Zero Trust CI/CD as a viable, at scale security model in controlled environments like insurance.The suggested secure blueprint neutralizes typical attack vectors, decreases response times, and complies with Zero Trust architecture, which provides insurance platforms with a solid basis of secure, cloud-native development.



REFERENCES

[1] Fernández González, D., Rodríguez Lera, F. J., Esteban, G., & Fernández Llamas, C. (2022). Secdocker: Hardening the continuous integration workflow: Wrapping the container layer. SN Computer Science, 3, 1-13. https://doi.org/10.48550/arXiv.2104.07899

[2] Hahn, D. A., Davidson, D., & Bardas, A. G. (2020). Security issues and challenges in service meshes -- an extended study. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2010.11079

[3] Avirneni, S. T. (2025). Establishing Workload Identity for Zero Trust CI/CD: From Secrets to SPIFFE-Based Authentication. arXiv preprint arXiv:2504.14760. https://doi.org/10.48550/arXiv.2504.14760

[4] Pecka, N., Othmane, L. B., & Valani, A. (2022). Making Secure Software Insecure without Changing Its Code: The Possibilities and Impacts of Attacks on the DevOps Pipeline. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2201.12879

[5] Vaucher, S., Pires, R., Felber, P., Pasin, M., Schiavoni, V., & Fetzer, C. (2018, July). SGX-aware container orchestration for heterogeneous clusters. In 2018 IEEE 38th International Conference

on Distributed Computing Systems (ICDCS) (pp. 730-741). IEEE. https://doi.org/10.48550/arXiv.1805.05847

[6] Pallewatta, S., & Babar, M. A. (2024). Towards Secure Management of Edge-Cloud IoT Microservices using Policy as Code. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2406.18813

[7] Ali, A., Imran, M., Kuznetsov, V., Trigazis, S., Pervaiz, A., Pfeiffer, A., & Mascheroni, M. (2024). Implementation of new security features in CMSWEB Kubernetes Cluster at CERN. EPJ Web of Conferences, 295, 07026. https://doi.org/10.1051/epjconf/202429507026

[8] Ugale, S., & Potgantwar, A. (2023). Container Security in Cloud Environments: A Comprehensive Analysis and Future Directions for DevSecOps. Eng. Proc. 2023, 57. https://doi.org/10.3390/engproc2023059057

[9] Pochu, S., Nersu, S. R. K., & Kathram, S. R. (2024). Enhancing Cloud Security with Automated Service Mesh Implementations in DevOps Pipelines. Deleted Journal, 7(01), 90–103. https://doi.org/10.60087/jaigs.v7i01.300

[10] Iheuwa, G., Badmus, O., & Ogieva, M. (2024). Zero Trust Security Model in DevOps for Cloud Native Environments. Technix International Journal for Engineering Research. 11. 10. https://www.researchgate.net/publication/390322531_Zero_Trust_Security_Model_in_DevOps_for_Cloud_Native_Environments