Innovative Journal of Computing, Security and Digital Technologies
Volume 1, Issue 1, Jan – Apr 2026
Page no : 10-14

A Zero-Trust Security Framework for Protecting Cloud-Native Applications

Dr. R. Karthik

Department of Computer Science and Engineering, Anna University, Chennai, Tamil Nadu, India

Abstract

Cloud-native applications use distributed services, containers, APIs, and dynamically changing workloads. These characteristics make perimeter-based security insufficient because network location alone cannot establish trust. This paper proposes a conceptual zero-trust security framework combining identity and workload verification, contextual policy evaluation, least-privilege authorization, service-level enforcement, continuous telemetry, threat detection, and controlled response. The framework also considers secure software delivery, secrets management, data protection, and policy governance. As a design-oriented paper, it reports no experimental results; instead, it defines evaluation criteria for future validation in production-like environments.

Keywords: zero trust; cloud-native security; microservices; least privilege; identity; continuous monitoring

Publication Date : 16.01.2026

1. INTRODUCTION

Cloud-native applications are commonly composed of independently deployable services running in containers and managed by orchestration platforms. Their flexibility enables rapid delivery and scaling, but expands the number of identities, interfaces, secrets, and runtime relationships that require protection. Static network boundaries cannot reliably represent trust when services move across clusters and cloud environments.

Zero trust evaluates access requests using identity, resource, context, and explicit policy rather than assuming trust from network location. This paper proposes a practical framework for cloud-native applications and outlines its workflow, implementation controls, risks, and evaluation criteria.

2. BACKGROUND AND RELATED CONCEPTS

2.1 Zero-trust principles

Core principles include explicit verification, least-privilege access, and the assumption that compromise may occur. These principles require reliable identities, clear resource boundaries, policy enforcement, and useful telemetry.

How to Cite

A Zero-Trust Security Framework for Protecting Cloud-Native Applications. (2026). Innovative Journal of Computing, Security and Digital Technologies, 1(1), 10-14.
https://ijcdst.in/index.php/about/article/view/5

2.2 Cloud-native characteristics

Microservices communicate through APIs and service-to-service calls; containers are short-lived, workloads scale dynamically, and deployment pipelines frequently introduce new versions. Controls should therefore attach to identities and workloads rather than rely only on static IP addresses.

2.3 Complementary controls

Zero trust complements secure development, vulnerability management, segmentation, runtime protection, encryption, and incident response. Shared identity and policy information helps these controls operate consistently.

3. PROPOSED FRAMEWORK

The framework follows a continuous decision-and-feedback cycle. Each request is associated with a verified human or workload identity, target resource, relevant posture signals, and context. A policy engine evaluates these attributes and returns a narrowly scoped decision. Enforcement occurs close to the application or service, while telemetry supports monitoring and response.

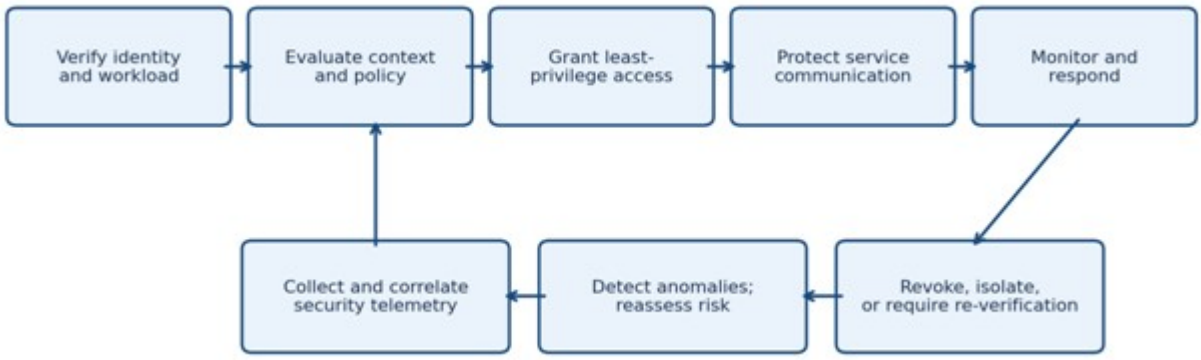


Figure 1. Proposed Zero-Trust Workflow for Cloud-Native Applications.

3.1 Identity and workload verification

Human users should authenticate through a centralized identity provider using strong authentication. Workloads should use attributable, short-lived identities. Where reliable, device and workload posture—such as patch state, image provenance, and runtime configuration—can inform decisions.

3.2 Contextual policy evaluation

Policies should consider requester, action, target resource, environment, risk signals, and data sensitivity. They should be explicit, version-controlled, tested, and periodically reviewed to prevent broad or conflicting permissions.

3.3 Least-privilege enforcement

Grant only the actions and resources needed for a defined task. Enforcement may occur at an API gateway, service mesh, application middleware, or cloud control plane. Authentication should not automatically authorize every later service call.

3.4 Continuous monitoring and response

Telemetry should record identity, policy outcome, resource, time, and relevant request metadata while minimizing sensitive data. Correlated signals can reveal unusual access or unexpected service communication. Response may revoke credentials, block requests, isolate workloads, or require re-verification; automated playbooks need testing and rollback procedures.

4. IMPLEMENTATION ACROSS THE LIFECYCLE

4.1 Build and supply-chain controls

Scan dependencies and images, protect build credentials, verify artifact provenance, and prevent unapproved artifacts from reaching production. Treat infrastructure and policies as code, with review and automated tests.

4.2 Deployment and orchestration

Admission controls can reject workloads that violate baseline requirements. Namespaces and network policies reduce unnecessary communication. Deliver secrets through managed mechanisms, rotate them, and keep them out of source code and images.

4.3 Runtime and service communication

Workload identities and mutual authentication can protect service calls. Service-mesh controls should be application-aware; overly broad allow rules undermine zero trust. Runtime monitoring should detect deviations without overwhelming operators.

4.4 Data protection

Apply access rules according to data sensitivity and purpose. Encryption, controlled key access, audit trails, and data minimization reduce the impact of unauthorized access. Logs and traces should avoid exposing credentials or personal data.

5. THREAT MODEL AND RISK REDUCTION

The framework targets stolen credentials, compromised containers, lateral movement, excessive permissions, exposed secrets, and vulnerable software components. It cannot eliminate all risk: compromised identities may still perform permitted actions, policy errors may grant excessive access, and monitoring may miss subtle attacks. Teams should document assets, trust boundaries, likely adversaries, entry points, and the consequences of compromise.

6. EVALUATION CRITERIA

The framework is proposed, not experimentally validated. A future pilot should compare it with a documented baseline using realistic attack scenarios. Recommended measures include:

- Coverage of service paths protected by authenticated, policy-enforced access
- Number and scope of standing permissions and unused privileges
- Detection precision, recall, false-positive rate, and time to detect
- Time to contain incidents and frequency of response rollback
- Request latency, deployment time, policy-change failures, and availability

- Policy review completion and identity-lifecycle compliance

Reports should specify the test environment, workload, measurement method, and limitations. This paper makes no claims about measured performance.

7. CHALLENGES AND LIMITATIONS

7.1 Policy complexity

Large environments can accumulate overlapping policies and exceptions. Ownership, testing, and periodic review are needed to prevent inconsistent enforcement.

7.2 Legacy integration

Some applications cannot enforce fine-grained authorization. Transitional controls may be needed, but exceptions should have owners, compensating safeguards, and expiration dates.

7.3 Telemetry and privacy

Monitoring depends on accurate timestamps, consistent identifiers, and useful event fields. Excessive collection creates cost and privacy risk; insufficient context limits investigations. Define purpose, retention, access, and redaction rules.

7.4 Operational adoption

Zero trust requires coordination across application, platform, security, and operations teams. Training, clear ownership, staged rollout, and usable exception processes help prevent controls from being bypassed.

8. FUTURE DIRECTIONS

Future work should validate the framework in representative environments and report measurable outcomes. Research directions include risk-adaptive authorization, policy verification, cross-cloud identity federation, privacy-preserving analytics, explainable detection, and maintaining access decisions during control-plane outages.

9. CONCLUSION

This paper proposed a conceptual zero-trust framework based on verified identities, contextual policy decisions, least-privilege enforcement, continuous monitoring, and controlled response. Its effectiveness depends on consistent identity management, governed policies, secure software delivery, and operational feedback. The evaluation criteria provide a basis for empirical validation; production claims should be supported by transparent measurements.

REFERENCES

1. 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
2. Souppaya, M., Morello, J., & Scarfone, K. (2017). Application Container Security Guide (NIST SP 800-190). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-190

3. National Institute of Standards and Technology. (2022). Secure Software Development Framework (SSDF) Version 1.1 (NIST SP 800-218). https://doi.org/10.6028/NIST.SP.800-218

4. Cybersecurity and Infrastructure Security Agency. (2023). Zero Trust Maturity Model, Version 2.0. https://www.cisa.gov/zero-trust-maturity-model

5. Cloud Security Alliance. (2017). Security Guidance for Critical Areas of Focus in Cloud Computing v4.0. https://cloudsecurityalliance.org

6. National Security Agency. (2021). Embracing a Zero Trust Security Model. https://www.nsa.gov

7. Cloud Native Computing Foundation. (n.d.). Cloud Native Security Whitepaper. https://www.cncf.io

8. European Union Agency for Cybersecurity. (2023). Cloud security. https://www.enisa.europa.eu