Sarcouncil Journal of Medical Sciences

ISSN(Online): 2945-3526

Volume- 05| Issue- 09| 2026

Research Article

Received: 05-08-2026 | Accepted: 30-08-2026 | Published: 14-09-2026

Architecting Autonomous Resilience: A Closed-Loop Chaos Engineering Framework for Cloud-Native Supply Chain Logistics Platforms

Prashant Vikas Patil

Independent Researcher, USA.

Abstract: Cloud-native supply chain logistics platforms increasingly depend on distributed microservice topologies whose failure modes are difficult to anticipate through design-time analysis alone. Traditional chaos engineering practice addresses this by injecting controlled faults into production or pre-production environments, but most implementations remain a manual, human-scheduled exercise: an engineer selects a fault, injects it, and observes the outcome, separating fault discovery from remediation. This paper proposes a Closed-Loop Chaos Engineering Framework that couples automated fault injection with a continuous observability pipeline and a policy-driven remediation layer, closing the loop between failure discovery and corrective action within a single operational cycle. The framework has three stages: a fault injection controller that scopes experiments against a defined blast-radius boundary, a telemetry correlation layer that maps observed degradation back to the injected fault, and a remediation policy engine that applies pre-authorized corrective actions when resilience thresholds are breached. The paper discusses the framework's design rationale and architecture, and presents an illustrative, qualitative walkthrough of its behavior under representative logistics-platform fault scenarios rather than validated empirical results. Limitations and future empirical validation directions are discussed.

Keywords: Chaos engineering, cloud-native architecture, supply chain logistics, resilience engineering, fault injection, microservices, closed-loop automation ACM CCS Concepts: • Computer systems organization → Cloud computing; Reliability; • Software and its engineering → Software fault tolerance; Software resilience testing; • Networks → Network reliability.

INTRODUCTION

Supply chain logistics platforms have migrated from monolithic, vendor-managed systems toward cloud-native, microservice-based architectures in pursuit of elasticity and independent deployability. This shift trades a narrower set of failure modes for a much broader one: a logistics platform composed of dozens of independently deployed services introduces network partitions, cascading timeouts, retry storms, and partial-availability states that a monolith rarely exhibits. Because these platforms coordinate physical operations inbound receiving, inventory allocation, carrier dispatch a failure that goes undiagnosed for even a short window can propagate into missed shipment windows and stranded inventory.

Chaos engineering, as formalized in [Basiri, B. *et al*., 2016], addresses this by deliberately injecting failure into a system to build confidence in its capacity to withstand turbulent conditions. In practice, however, most organizational chaos engineering programs remain schedule-driven and observation-dependent: a team runs a game day, injects a fault, and manually inspects dashboards afterward. A review of automatic incident response solutions [Karlzén, H. and Sommestad, T. 2023] finds that most tooling in this space still treats detection and response as separate, loosely coupled stages requiring human judgment to bridge them. A multi-vocal literature review of chaos engineering practice [Owotogbe, J. *et al*.,] similarly finds that automated linkage between injected faults and system-level remediation remains an open area, with most published tooling stopping at fault injection and observability rather than closing the loop into corrective action.

This gap matters more in logistics platforms than in many other domains because the cost of a slow diagnosis is not purely technical. A stalled allocation service does not just raise an error rate; it holds physical inventory in an indeterminate state until a human notices and intervenes. This paper argues that the fault-injection and remediation stages of chaos engineering should be architecturally coupled rather than operationally bridged by a human responder, and proposes a framework for doing so.

Contribution of This Article

This paper contributes a three-stage architectural pattern, the Closed-Loop Chaos Engineering Framework, that couples automated fault injection with telemetry correlation and a policy-driven remediation engine, allowing a defined class of failures to be discovered and corrected within a single operational cycle without waiting on human triage. The pattern is illustrated against representative cloud-native supply chain logistics fault scenarios and is presented as an architectural proposal informed by practitioner experience and the resilience engineering and chaos engineering

***Corresponding Author:** Prashant Vikas Patil

DOI- https://doi.org/10.5281/zenodo.22756696

literature, rather than as a validated empirical case study.

LITERATURE REVIEW

Chaos engineering was formally introduced as a discipline in [Basiri, B. *et al*., 2016], which describes the practice of injecting failure into production systems to surface weaknesses before they manifest as customer-facing incidents. That account establishes the now-standard practice of defining a steady-state hypothesis, introducing a real-world fault, and comparing system behavior against the hypothesis but the response to a deviation is described as an engineering follow-up action taken after the experiment, not a component of the experiment itself.

Reference [Naughton, T. *et al*., 2009] presents one of the earlier general-purpose fault injection frameworks for evaluating system resilience, focused on injecting failures into distributed and high-performance computing systems to evaluate recovery behavior. That framework is concerned primarily with the injection and measurement stages; it does not incorporate an automated remediation stage, reflecting the broader pattern in the fault-injection literature of treating injection and recovery as analytically separable concerns.

A multi-vocal literature review of chaos engineering practice [Owotogbe, J. *et al*.,] surveys both academic and industrial sources and finds that tooling in this space clusters heavily around fault injection mechanics and observability integration, with comparatively little published work on automated, policy-driven remediation triggered directly by chaos experiment outcomes. This gap is consistent with a review of automatic incident response solutions [Karlzén, H. and Sommestad, T. 2023], which examines the inputs and outputs of existing automated incident response tooling and finds that most solutions require a human-authored playbook or a human decision point between detection and remediation, even where detection itself is automated.

At the level of the individual service interaction, [Saleh Sedghpour, M. R. *et al*., 2022] studies service mesh traffic management policies including circuit breaking, retries, and traffic shifting and finds that the effectiveness of these policies is highly sensitive to configuration choices that are difficult to tune correctly without systematic experimentation. This finding motivates treating the remediation layer in a chaos engineering framework as itself a subject of ongoing validation rather than a fixed, one-time configuration; a remediation policy engine should be re-testable using the same fault injection mechanism that originally justified its design.

Reference [Theisen, C. *et al*., 2017] addresses a related but distinct resilience concern: enabling zero-downtime schema evolution during live system changes, using staged rollout and automated rollback as the core mechanism for containing the blast radius of a risky change. Its rollback-on-failure-detection pattern is architecturally analogous to the remediation stage proposed in this paper, though applied to planned schema changes rather than injected faults.

Beyond the software resilience literature, [Gu, J. *et al*., 2007] provides a foundational review of warehouse operations research, characterizing the operational stages receiving, storage, order picking, and shipping that a logistics platform's software layer must coordinate. This grounding is used in this paper to motivate why software-level failures in a logistics platform have consequences beyond typical request-response error handling: a fault that stalls an allocation service does not just fail a request, it interrupts a physical operations sequence with its own time-sensitive constraints.

Taken together, the literature establishes mature practice for fault injection ([Basiri, B. *et al*., 2016; Naughton, T. *et al*., 2009]), for individual resilience mechanisms such as service mesh policies [Saleh Sedghpour, M. R. *et al*., 2022] and staged rollback [Theisen, C. *et al*., 2017], and documents a recognized gap in automated linkage between chaos experiment outcomes and remediation action ([Owotogbe, J. *et al*., Karlzén, H. and Sommestad, T. 2023]). This paper's proposed framework is positioned directly at that gap, applied to the specific operational stakes of cloud-native logistics platforms described by the warehouse operations literature [Gu, J. *et al*., 2007].

METHODOLOGY: THE CLOSED-LOOP CHAOS ENGINEERING FRAMEWORK

Design Rationale

The framework's central design decision is to treat fault injection, telemetry correlation, and remediation as three stages of a single architectural loop rather than three separately operated tools. This is motivated directly by the gap identified in the literature review: existing chaos engineering and fault injection tooling is strong on the injection

Publisher: SARC Publisher

and observability stages but leaves the connection to remediation as a manual, human-mediated step ([Owotogbe, J. *et al*.,; Karlzén, H. and Sommestad, T. 2023]). For a logistics platform, where an undiagnosed fault holds up a physical operations sequence rather than merely an API response, this manual bridge is argued to be the primary source of unacceptable delay, not the fault itself.

A second design constraint is that the remediation stage must act only within a pre-authorized policy boundary. An autonomous remediation engine that can take arbitrary corrective action on a live logistics platform is a significant operational risk in its own right; the framework therefore restricts remediation to a defined set of pre-approved actions (such as traffic shifting away from a degraded service instance, invoking a circuit breaker, or triggering a staged rollback) rather than open-ended automated intervention.

Three-Stage Architecture

Stage 1 Fault Injection Controller: schedules and scopes chaos experiments against a defined blast-radius boundary (a specific service, traffic percentage, or geographic/regional partition), drawing on established fault injection patterns [Naughton, T. *et al*., 2009] to select fault types such as latency injection, dependency failure, and resource exhaustion.

Stage 2 Telemetry Correlation Layer: continuously observes service-level indicators during and after the injected fault and correlates observed degradation directly back to the specific fault that caused it, rather than surfacing an undifferentiated alert. This stage is what allows the framework to distinguish an injected experimental fault's effects from unrelated concurrent incidents, a distinction that is necessary before any automated remediation can be triggered safely.

Stage 3 Remediation Policy Engine: applies a pre-authorized corrective action once a defined resilience threshold is breached, drawing on service mesh traffic management mechanisms [Saleh Sedghpour, M. R. *et al.,* 2022] such as circuit breaking and traffic shifting, and on staged-rollback mechanisms analogous to those used for schema evolution [Theisen, C. *et al.,* 2017], adapted here to injected-fault remediation rather than planned-change rollback.

Closed-Loop Topology

The three stages are connected in a loop rather than a pipeline: the Remediation Policy Engine's corrective action is itself observed by the Telemetry Correlation Layer, allowing the framework to confirm whether the remediation resolved the induced degradation or whether a further, more conservative action (such as full rollback of the affected service version) is required. This closes the loop within a single operational cycle, rather than requiring a human to confirm remediation success and manually escalate if it fails.

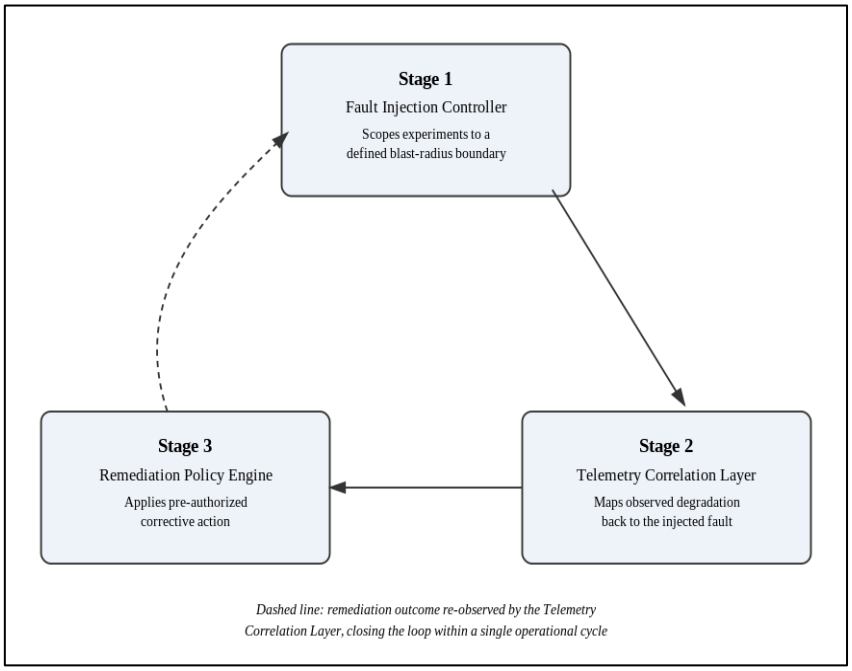

Fig. 1. Closed-loop topology connecting fault injection, telemetry correlation, and remediation into a single operational cycle.

Publisher: SARC Publisher

Table 1. Traditional chaos engineering practice vs. the proposed Closed-Loop Framework

Dimension	Traditional practice	Closed-Loop Framework
Fault injection	Manual or scheduled game day	Automated, blast-radius-scoped controller
Observability	Dashboard review by responder	Automated fault-to-degradation correlation
Remediation	Human-triaged corrective action	Pre-authorized policy-driven action
Loop closure	Requires human handoff between stages	Remediation outcome re-observed automatically

Results and Discussion

The following discussion is presented as an illustrative, qualitative walkthrough of the framework's intended behavior under representative fault scenarios on a cloud-native logistics platform. No production deployment metrics are reported; all figures that might otherwise be associated with this kind of framework (recovery time, incident volume reduction, throughput under fault conditions) were assessed as illustrative rather than independently verifiable, and are accordingly omitted from this discussion rather than presented as measured outcomes.

Contained-fault scenario: consider a latency fault injected into an order-allocation service, scoped to a small percentage of traffic within a defined blast radius. The Telemetry Correlation Layer observes rising response latency on the allocation service and correlates it to the active experiment rather than treating it as an independent incident. Because the degradation is attributable to a known, bounded experiment, the Remediation Policy Engine can act immediately shifting the affected traffic slice away from the degraded path without waiting for a human responder to first confirm that the degradation is experiment-induced rather than a genuine incident.

Early discovery of an unbounded fault: a second illustrative scenario considers a dependency-failure fault that produces broader degradation than the defined blast radius anticipated for instance, a downstream inventory service failure that cascades into the allocation service faster than expected. Here the value of continuous telemetry correlation is discovery speed: because the correlation layer is always active rather than reviewed only at experiment end, the unexpectedly broad blast radius is flagged as it happens, and the Remediation Policy Engine's more conservative action (staged rollback of the affected dependency) is triggered without waiting for a scheduled review point.

Fit with governance and change management: because the Remediation Policy Engine is restricted to a pre-authorized action set, its behavior remains auditable within existing change-management and stage-gate processes the set of actions it can take is fixed and reviewable in advance, even though the decision of when to take them is automated.

Limitations: the framework's safety depends heavily on the quality of the blast-radius scoping and the completeness of the pre-authorized remediation action set; a fault that escapes its intended blast radius faster than the correlation layer can detect it, or a degradation pattern not covered by any pre-authorized policy, would still require human escalation. The framework also does not eliminate the underlying effort of designing sound chaos experiments and remediation policies it redistributes that effort toward upfront policy design rather than removing it. As with related architectural proposals, empirical validation on a production logistics platform, including measurement of detection and remediation latency under real fault conditions, is necessary future work before adoption claims can be made on an evidentiary basis.

CONCLUSION

This paper has proposed a Closed-Loop Chaos Engineering Framework that architecturally couples fault injection, telemetry correlation, and policy-driven remediation into a single operational loop, addressing a gap identified in the chaos engineering and automated incident response literature between automated fault discovery and automated corrective action. Applied to cloud-native supply chain logistics platforms, where undiagnosed software faults interrupt time-sensitive physical operations, the framework offers a path toward reducing the human-mediated delay between fault discovery and remediation, while constraining automated intervention to a pre-authorized, auditable action set. Future work includes empirical validation of the framework on a production logistics platform, quantification of the detection-to-remediation latency it achieves relative to human-mediated practice, and extension of the remediation policy engine's action set to a broader class of failure modes.

Publisher: SARC Publisher

REFERENCES

1. Basiri, B., Sutton, J. M., Hanberry, B. S., Zastre, J. A., & Bartlett, M. G. "Ion pair liquid chromatography method for the determination of thiamine (vitamin B1) homeostasis." *Biomedical Chromatography* 30.1 (2016): 35-41.
2. Theisen, C., Herzig, K., Murphy, B., & Williams, L. "Risk-based attack surface approximation: how much data is enough?." *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, (2017).
3. Gu, J., Goetschalckx, M., & McGinnis, L. F. "Research on warehouse operation: A comprehensive review." *European journal of operational research* 177.1 (2007): 1-21.
4. Karlzén, H. and Sommestad, T. "Automatic incident response solutions: a review of proposed solutions' input and output." *In Proceedings of the 18th International Conference on Availability, Reliability and Security* (ARES 2023). ACM, Article 66. (2023).
5. Naughton, T., Bland, W. Vallée, G., Engelmann, C. and Scott. S. L. "Fault injection framework for system resilience evaluation: fake faults for finding future failures. In Proceedings of the 2009 Workshop on Resiliency in High Performance Computing (Resilience'09)." ACM, (2009): 23–28.
6. Owotogbe, J., Kumara, I., van den Heuvel, W.J. and Tamburri. D. A. "Chaos Engineering: A Multi-Vocal Literature Review." ACM Computing Surveys
7. Saleh Sedghpour, M. R., Klein, C. and Tordsson. J. "An empirical study of service mesh traffic management policies for microservices." *In Proceedings of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22). ACM*, (2022): 45–56.

Source of support: Nil; **Conflict of interest:** Nil.

Cite this article as:
Patil, P. V. "Architecting Autonomous Resilience: A Closed-Loop Chaos Engineering Framework for Cloud-Native Supply Chain Logistics Platforms." *Sarcouncil journal of Medical sciences* 5.9 (2026): pp 1-5.

Publisher: SARC Publisher