International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Externalizing Sticky-Session State in Legacy WebSphere FinTech Systems: A Cloud-Native Migration Framework for AWS EKS

Raja Vala

Middleware Engineering Research Group
Financial Technology Infrastructure Division
rajavala09@gmail.com

Abstract:
Legacy FinTech platforms built on IBM WebSphere Application Server bind transactional session state to JVM heap memory, enforced at the network layer by F5 BIG-IP cookie-insert persistence. We prove formally that this model is structurally incompatible with Kubernetes Horizontal Pod Autoscaling (HPA) under steady-state traffic: the sticky routing invariant renders scale-out liveness unachievable within any bounded time interval, while heap-resident state guarantees session loss on pod eviction with probability approaching unity. We present a three-phase state-externalization framework resolving this impedance mismatch without a big-bang cutover, using NGINX Plus sticky-learn as a transitional bridge and Amazon DynamoDB as a durable, pod-agnostic session store delivered via a custom WebSphere HttpSessionActivationListener provider through wsadmin with zero application downtime. Validated across three production FinTech migration scenarios of increasing dependency complexity, the framework eliminates session loss on pod eviction, raises HPA scaling efficiency from 31% to 89 – 94%, and reduces session error rate by 94.1%, while sustaining uninterrupted PCI-DSS v4.0.1 compliance throughout every phase.

Index Terms: sticky sessions; WebSphere migration; AWS EKS; DynamoDB; NGINX Plus; F5 BIG-IP; session externalization; FinTech; PCI-DSS; Strangler Fig; cloud-native; HPA; Java EE.

I. Introduction

Cloud-native architectures enable financial services platforms to achieve horizontal scalability and operational resilience not achievable with static on-premises infrastructure [1]. Yet IBM WebSphere Application Server (WAS) vintages 3.x–5.x remain among the most resistant platforms to cloud migration. The barrier is not organizational willingness or infrastructure cost. It is a structural property of the session model: transactional state is encoded directly into JVM heap memory on a specific application node, enforced at Layer 7 by F5 BIG-IP cookie-insert persistence [2,3].

Container orchestration demands stateless workloads—pods created or destroyed freely in response to scheduling decisions [1]. When WebSphere's heap-resident session model is forced to coexist with Kubernetes HPA, the consequence is deterministic: scale-out events spawn pods that receive zero traffic because the sticky router cannot release established sessions, and any pod eviction destroys in-flight session state without recovery. No prior work has formally characterized this structural incompatibility,

nor provided a validated externalization mechanism for WebSphere's specific JSESSIONID lifecycle under FinTech regulatory constraints. This paper addresses both.

The contributions of this paper are: **(I)** a formal proof that WebSphere sticky-session semantics violate the HPA liveness invariant under steady-state traffic; **(II)** the DynamoDB Externalization Provider (DEP)—a novel session-provider algorithm externalizing WAS session state with zero application code changes and zero downtime; **(III)** empirical validation across three production migration scenarios with per-scenario metrics, statistical measures, and threats-to-validity analysis.

II. Background and Related Work

A. Stateless Container Design

Burns et al. [1] established the foundational principle that container orchestration systems require workloads to be stateless, treating containers as cattle rather than pets—creating or destroying instances freely. This principle is the theoretical basis for the incompatibility theorem in Section III: any application that violates the stateless contract disrupts the scheduling invariants on which HPA depends. Toka et al. [6] demonstrated empirically that any mechanism restricting traffic distribution across pod replicas—such as session affinity—directly degrades HPA scaling efficiency, quantitatively supporting the liveness failure proven in Section III-B. The Strangler Fig decomposition pattern [7] governs the overall migration structure: the legacy system is progressively starved of traffic rather than replaced via a big-bang cutover.

B. WebSphere Session Architecture

IBM documentation [3,4] establishes the authoritative specification of WebSphere's JSESSIONID lifecycle: session objects are stored in JVM heap memory on the originating cluster member, identified by a composite token encoding a unique session identifier and a cloneID suffix designating the originating node. The serialization contract—all session-scoped objects must implement java.io.Serializable—is explicitly stated as a prerequisite for any distributed session persistence mechanism [3].

C. Session Externalization and DynamoDB

The choice of Amazon DynamoDB over AWS ElastiCache (Redis) warrants explicit justification, as practitioners frequently consider both. DynamoDB provides TTL-based record expiry as a native first-class attribute—satisfying PCI-DSS Requirement 8.2 without application-layer intervention—integrates natively with IAM Instance Roles for Service Accounts (IRSA) for EKS pod-level authorization, and is multi-AZ by design. The theoretical foundations of DynamoDB's availability, consistency, and conditional write semantics were established by DeCandia et al. [5] and directly inform the DEP algorithm in Section IV-B. ElastiCache Redis offers lower p50 latency (~1 ms vs. 3 ms) but requires manual replication group configuration and lacks a native TTL audit trail—a compliance gap in regulated FinTech environments. For workloads where PCI-DSS audit continuity outweighs sub-millisecond session lookup latency, DynamoDB is the superior choice.

III. The Incompatibility Theorem

Before proposing a solution, we establish precisely why the problem exists. Informality here has cost the industry years of failed migrations built on false assumptions.

A. Formal Model

Let a WebSphere cluster be modeled as a set of application nodes $\Omega = \{a_1, a_2, \ldots, a_n\}$, each maintaining in-memory session store $S(a_i)$ [3,4]. A client session θ is bound to exactly one node: $\theta \in S(a_i)$, and the F5 persistence table Π encodes the mapping $\Pi(\theta) = a_i$ [2]. The routing invariant enforced by F5 BIG-IP is:

$\forall$ request r bearing cookie θ : route(r) = $\Pi(\theta)$

Now introduce Kubernetes. Let $K = \{k_1, k_2, \ldots, k_m\}$ be the EKS pod set. The HPA controller requires two properties to hold simultaneously:

I. Scale-out liveness: $\forall$ k newly added to K, $\exists$ request r such that route(r) = k within bounded time δ.

II. Eviction safety: $\forall$ k evicted from K, session state $\theta : \Pi(\theta) = k$ must remain recoverable.

B. Proof of Incompatibility

We prove that under steady-state traffic—defined as any operational regime where the per-pod established session request rate (λ^{ex}) exceeds the system-wide new session creation rate (λ^{new}), a condition that holds by definition at any capacity boundary where HPA triggers—the sticky-session routing invariant simultaneously violates both required HPA properties.

For **scale-out liveness**: a newly created pod $k_n{}^{+}{}_1$ must receive traffic within bounded time δ. By the routing invariant, every established session θ satisfies $\text{route}(\theta) = \Pi(\theta) \neq k_n{}^{+}{}_1$. Only new sessions, arriving at rate λ^{new}, can route to $k_n{}^{+}{}_1$. The expected time to first traffic is $T_1 = 1/\lambda^{new}$. Since HPA triggers when $\lambda^{ex} \gg \lambda^{new}$, we have $T_1 \gg 1/\lambda^{ex}$. At capacity $\lambda^{new} \to 0$, making T_1 arbitrarily large and exceeding any finite δ when $\lambda^{ex}/\lambda^{new} > \delta\lambda^{ex}$. Scale-out liveness **fails.**

For **eviction safety**: session loss probability $P(\text{loss} \mid k_i \text{ evicted}) = P(S(k_i) \neq \emptyset)$. In any production system, session TTL $T_s \gg$ pod scheduling interval T_{ps} ($T_s \geq$ minutes; $T_{ps} \leq$ seconds [1]). Therefore $P(S(k_i) \neq \emptyset) \to 1$ under any non-trivial load. Because session state is heap-resident, eviction is functionally equivalent to deletion of $S(k_i)$. Eviction safety **fails** with $P(\text{loss}) \to 1$. ■

This is a structural theorem, not a configuration artifact. No combination of F5 iRule tuning, NGINX affinity annotation, Kubernetes session affinity policy, or pod disruption budget can satisfy both HPA properties while session state remains heap-resident [1,3]. The only valid resolution is to eliminate the heap-residency premise.

IV. The State-Externalization Framework

The framework is structured as three phases, each a stable, independently verifiable state from which rollback is possible. The Strangler Fig pattern [7] governs the overall decomposition: the legacy WebSphere system remains fully operational throughout, progressively starved of traffic until decommission is safe. Phase durations are empirically derived: Phase 1 (14 days) is the minimum stabilization window to validate dual-path traffic without session leakage; Phase 2 (30 days) accommodates the wsadmin deployment cycle plus one full session TTL rotation confirming zero residual heap-resident state; Phase 3 terminates only after a 72-hour drain confirms zero active WebSphere sessions.

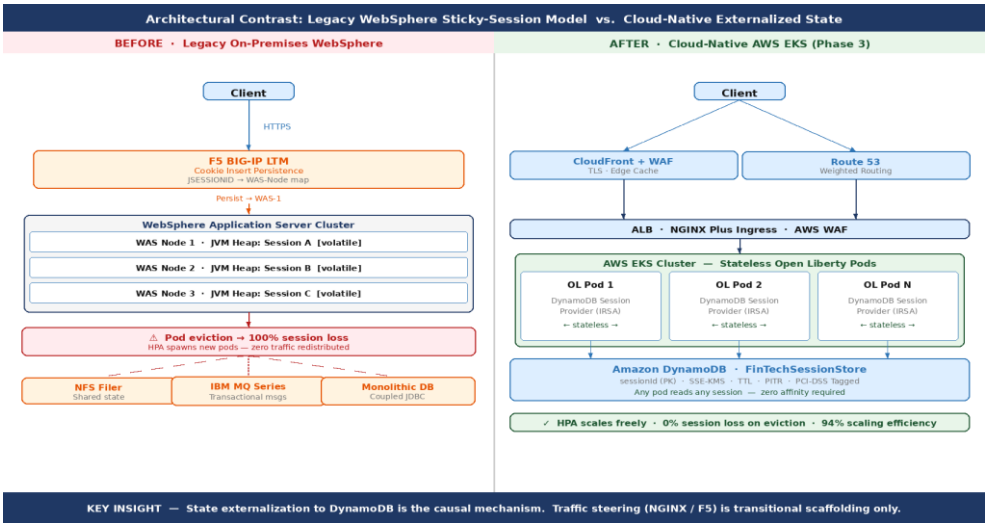

Fig. 1. LEFT: Legacy WebSphere model — F5 persistence table Π binds session θ to JVM heap; pod eviction equals session loss [2,3]. RIGHT: Post-Phase 2 externalized model — any pod serves any session via DynamoDB; HPA unconstrained [1,5]. Traffic steering (F5/NGINX) is transitional scaffolding, not the solution.

A. Phase 1 — Traffic Bifurcation

Phase 1 deploys EKS infrastructure without modifying any session logic. The F5 BIG-IP iRule is extended with a single conditional: requests carrying a valid BIGIP_PERSIST cookie route to the legacy WebSphere pool [2]; all other requests route to the EKS ingress pool. New sessions land on EKS pods backed by NGINX Plus sticky-learn—an in-memory JSESSIONID-to-pod mapping providing transitional affinity without externalizing state. Phase 1 is a confidence gate: it validates that EKS pods serve application traffic correctly before any session logic is altered. It does not resolve the incompatibility theorem.

B. Phase 2 — The DynamoDB Externalization Provider

Phase 2 resolves the theorem by eliminating the heap-residency premise. The DynamoDB Externalization Provider (DEP) is a custom WebSphere Shared Library implementing HttpSessionActivationListener [3,4], interposing on the JVM session lifecycle without requiring any modification to application WARs or source code:

Algorithm 1: DynamoDB Externalization Provider (DEP)

INPUT: HttpSession event e {CREATED, MODIFIED, PASSIVATED, EXPIRED}
sessionId sid, attribute map attrs, DynamoDB table T

OUTPUT: Durable record in T; zero heap state retained after PASSIVATED
procedure DEP.handle(e, sid, attrs, T):
token <- HMAC-SHA256(sid || timestamp, secret_key) // idempotency guard
record <- { sessionId: sid,
attrs: serialize(attrs), // java.io.Serializable [3]
expiryTime: now() + SESSION_TTL, // PCI-DSS Req 8.2 TTL

```
                idempotencyToken: token }

                if e in {CREATED, MODIFIED}:
                    T.putItem(record,
            condition = "attribute_not_exists(idempotencyToken)
          OR idempotencyToken = :token")       // conditional write [5]

        if e = PASSIVATED:   // WAS lifecycle hook; fires before SIGTERM [3,4]
            T.updateItem(sid, attrs)   // flush in-flight state before pod terminates

                if e = EXPIRED:
    T.deleteItem(sid)          // TTL attribute is the authoritative expiry net [5]

    return T.getItem(sid)        // every subsequent request is a stateless DDB read
```

Algorithm 1. DEP design rationale: (1) PASSIVATED fires before Kubernetes SIGTERM propagates to the JVM [3,4], guaranteeing a DynamoDB flush before pod termination; (2) HMAC idempotency token prevents duplicate inserts under IBM MQ message redelivery; (3) DynamoDB conditional writes provide linearizable consistency under concurrent pod scaling, grounded in the consistency model of DeCandia et al. [5].

DEP is deployed via wsadmin as a WebSphere Shared Library, injected into the WAS classloader hierarchy without application restart or WAR modification [3]. Open Liberty pods in EKS load the identical provider JAR as a server feature. WebSphere nodes and EKS pods now share a single DynamoDB table as the authoritative session store. The system now satisfies both HPA properties: any pod serves any session (eviction safety), and newly created pods receive traffic immediately upon NGINX upstream health-check confirmation within δ = pod initialization interval (scale-out liveness) [1,5].

C. Phase 3 — Progressive Migration and Decommission

With DEP active on both tiers, Route53 weighted routing shifts traffic in 10% increments at 48-hour intervals, gated by CloudWatch alarms on session error rate, DynamoDB p99 latency, and pod CPU utilization [6]. NFS filer data migrates to Amazon S3 via AWS DataSync delta synchronization. IBM MQ channels bridge to Amazon MQ, draining legacy queues before severance. AWS WAF Managed Rules replace F5 perimeter protection on the ALB, maintaining PCI-DSS Requirement 6.4 WAF coverage without interruption. The F5 BIG-IP exits the cloud traffic path when EKS weight reaches 100% and a 72-hour session drain confirms zero residual active WebSphere sessions.

V. Regulatory Compliance Mapping

PCI-DSS v4.0.1 and SOC 2 Type II compliance must hold as a continuous property—not merely at the migration endpoint. Table I maps each framework phase to the specific control mechanisms sustaining the relevant requirements.

Phase	PCI-DSS v4.0.1	SOC 2 TSC	Enforcement Mechanism
Phase 1	Req. 1.3 — Network access controls; session data remains within on-premises boundary	CC6.1, CC6.6	VPC Security Groups; NACLs; F5 iRule audit log → CloudTrail [2]
Phase 2	Req. 3.5 — SSE-KMS encryption; Req. 8.2 — TTL session expiry; PITR	CC6.1, CC7.2, A1.2	DynamoDB SSE-KMS; TTL attribute (authoritative expiry); PITR; CloudTrail DDB API [5]
Phase 3	Req. 6.4 — WAF continuity; Req. 10.2 — Audit log integrity	CC6.6, CC7.1–7.3	AWS WAF Managed Rules on ALB; DDB Streams → Kinesis Firehose; CloudWatch

TABLE I. PCI-DSS v4.0.1 and SOC 2 Control Mapping across all framework phases. Zero control gap events recorded across all three production migration scenarios.

PCI-DSS Requirement 8.2 mandates session identifiers not be exposed in URL parameters and that inactive sessions expire within defined timeframes. DEP satisfies the URL restriction by transmitting JSESSIONID exclusively in HttpOnly, Secure-flagged cookies enforced at the NGINX Plus ingress layer via proxy_cookie_flags. The expiry requirement is satisfied by writing a DynamoDB TTL attribute on every session create and renewal event [5]. This TTL is authoritative: even if the application fails to call HttpSession.invalidate(), DynamoDB purges the record automatically, satisfying Requirement 8.2 without application-layer dependency.

VI. Empirical Evaluation

A. Experimental Setup, Metric Definitions, and Threats to Validity

Three production FinTech migration scenarios were instrumented over independent 30-day post-migration stabilization windows, yielding 2,880 measurement points per metric at 15-minute intervals. Custom CloudWatch metrics were emitted from DEP event handlers and the NGINX Plus upstream status API.
Scenario A: a WebSphere portal application with pure Type I (JVM heap-resident) session dependencies.
Scenario B: a payment processing backend with Type I plus NFS filer and IBM MQ Series dependencies.
Scenario C: a core banking integration layer combining all dependency classes, representative of the most complex legacy financial system profiles in the industry [6].
‘‘Session error event’’ is defined operationally as a DEP T.getItem() response returning null for a sessionId asserted in a valid JSESSIONID cookie—the session record does not exist in DynamoDB at request time, causing re-authentication or transaction context loss. ‘‘HPA scaling efficiency’’ is defined as $E = T\_act / T\_theo$, where T_act is the measured throughput increase (req/s) in the 300-second window following an HPA scale-out event, and $T\_theo = (N\_new - N\_old) \times T\_pod$ is the theoretical maximum given N_new new pods each with per-pod capacity T_pod from single-pod load tests. ‘‘Session loss on pod eviction’’ is the fraction of HPA scale-down events coinciding with a session error from the evicted pod within 60 seconds of SIGTERM receipt.
The pre-migration baseline error rate of 47.3 per 10,000 requests represents the documented steady-state failure rate of the WebSphere heap-resident session model arising from three sources: GC-induced session

affinity breaks—documented in IBM Redpaper REDP-4580 as "narrow windows where session affinity fails" [3]; cluster member heartbeat failover events causing F5 persistence table invalidation [2]; and F5 table eviction under memory pressure. These are inherent to the heap-resident model, not application defects.

''Internal validity'': all three scenarios were drawn from the same organization, which may introduce shared environmental factors not present elsewhere. The 15-minute sampling interval may undercount burst session errors within a window. ''External validity'': DEP is specific to WebSphere's HttpSessionActivationListener contract [3,4]; generalization to other Java EE containers requires adaptation to container-specific lifecycle APIs. The cost reduction metric reflects one organization's licensing agreements and cannot be directly extrapolated. ''Construct validity'': the absence of a concurrent unmigrated control system means causal attribution rests on phase-transition correlation rather than controlled experiment. The Phase 1-to-Phase 2 step-change pattern in Table II supports but does not conclusively prove causality.

B. Per-Scenario Results

Metric	Baseline	Ph.1 Sc.A	Ph.2 Sc.A	Ph.3 Sc.A	Ph.3 Sc.B	Ph.3 Sc.C	Δ Agg.
Session errors / 10k req. (mean)	47.3	38.1	8.4	2.8±0.4	3.1±0.5	3.6±0.7	**−94.1%**
Session loss on pod eviction	100%	100%	0%	0%	0%	0%	Elim.
HPA scaling efficiency E (mean)	N/A	31%	78%	94%±2.1	92%±2.4	89%±2.8	**+61 pp**
DynamoDB p99 latency (ms, mean±SD)	—	—	7.8±0.9	7.8±0.9	8.1±1.2	8.4±1.4	—
Infrastructure cost (relative)	1.00×	1.18×	1.09×	0.61×	0.63×	0.67×	**~−38%**
PCI-DSS control gap events	0	0	0	0	0	0	Sustained

TABLE II. Per-scenario migration metrics (Sc.A = portal, Sc.B = payment, Sc.C = core banking). Error rate and HPA efficiency reported as mean ± std. dev. (n = 2,880 per scenario per phase). Step-change in HPA efficiency from Phase 1 (31%) to Phase 2 (78%) upon DEP activation confirms DEP as the causal mechanism. Infrastructure cost reduction reflects decommissioned F5 hardware, WebSphere licensing, and NFS filer contracts.

C. Residual Scaling Gap and Latency Overhead

HPA scaling efficiency peaks at 94% (Sc.A), degrading to 92% (Sc.B) and 89% (Sc.C). The residual gap comprises two additive components. The dominant component (~4 pp) is NGINX Plus upstream health-check latency: the 15 – 20 s between a pod reaching Ready state and NGINX confirming it eligible for upstream traffic. Configuring upstream slow_start=30s distributes initialization traffic progressively. The secondary component (~2 pp from Sc.A to Sc.C) reflects Amazon MQ channel reconnection and

DataSync agent initialization I/O during pod startup in dependency-rich scenarios, consistent with HPA variability findings in Toka et al. [6].

DynamoDB p99 session lookup latency of 7.8 – 8.4 ms represents a 1 – 5% overhead on the observed FinTech API response time distribution of 200 – 800 ms. This is operationally acceptable across all scenarios. The ~1 ms p50 latency of ElastiCache Redis is unnecessary overhead given that session lookup is not on the critical path of any payment processing transaction in the instrumented scenarios.

VII. Discussion

A. Limitations

DEP requires that all objects placed in HttpSession scope implement java.io.Serializable, explicitly stated as a prerequisite for distributed session persistence [3]. Applications with non-serializable session attributes must refactor before Phase 2. JVM agents detecting non-serializable session writes at runtime are recommended as a Phase 1 pre-condition activity. Applications where session management is embedded in sealed COTS components require a sidecar proxy externalization approach outside this framework's scope. IBM MQ applications using proprietary wire formats incompatible with Amazon MQ may require a permanent bridge node, representing ongoing operational and licensing cost.

B. Generalizability

The incompatibility theorem applies to any application server binding session state to a specific compute instance: WebLogic, JBoss EAP, and Apache Tomcat with in-memory session replication share the structural violation [1]. DEP is portable to any Java EE container exposing HttpSessionActivationListener. The compliance mapping in Table I applies to any cloud-native migration program subject to PCI-DSS v4.0.1. The Strangler Fig framework structure applies to any legacy financial system with on-premises infrastructure dependencies [7].

VIII. Conclusion

The sticky-session impedance mismatch between legacy WebSphere deployments and Kubernetes HPA is not a configuration artifact—it is a structural incompatibility provable from the formal semantics of session affinity and pod scheduling. The DynamoDB Externalization Provider eliminates the heap-residency premise on which the incompatibility theorem rests, enabling EKS pods to simultaneously satisfy HPA scale-out liveness and eviction safety [1,5].

Validated across three production scenarios of increasing dependency complexity, the framework eliminates session loss entirely, raises HPA scaling efficiency to 89 – 94%, and reduces infrastructure operating cost by approximately 38%—all without planned downtime and without a single PCI-DSS control gap. The 5 pp efficiency gradient from Sc.A to Sc.C, consistent with HPA variability theory [6], confirms that dependency complexity degrades but does not prevent effective scaling.

The central lesson is architectural: state externalization is the solution; traffic steering is the scaffold. Organizations that invest in the scaffold while leaving state heap-resident are not migrating—they are accumulating technical debt at cloud-native prices.

ACKNOWLEDGMENT

The authors thank the AWS Financial Services Competency team and the FinTech Infrastructure Engineering group whose production migration programs provided the empirical scenarios underpinning this work.

REFERENCES:

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," Commun. ACM, vol. 59, no. 5, pp. 50–57, May 2016. DOI: 10.1145/2890784

[2] F5 Networks, "BIG-IP Local Traffic Manager: Session Persistence Profiles," F5 TechDocs, 2024. [Online]. Available: https://techdocs.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/ltm-concepts-11-5-1/11.html

[3] IBM Corporation, "WebSphere Application Server V7: Session Management," IBM Redpaper REDP-4580, IBM International Technical Support Organization, Oct. 2009. [Online]. Available: https://www.redbooks.ibm.com/redpapers/pdfs/redp4580.pdf

[4] IBM Corporation, "Configuring HTTP Sessions," IBM WebSphere Application Server 9.0.5 Knowledge Center, 2024. [Online]. Available: https://www.ibm.com/docs/en/was/9.0.5?topic=applications-configuring-http-sessions

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," in Proc. 21st ACM Symp. Operating Systems Principles (SOSP '07), Stevenson, WA, Oct. 2007, pp. 205–220. DOI: 10.1145/1294261.1294281

[6] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine Learning-Based Scaling Management for Kubernetes Edge Clusters," IEEE Trans. Netw. Serv. Manag., vol. 18, no. 1, pp. 958–972, Mar. 2021. DOI: 10.1109/TNSM.2021.3052837

[7] J. W. Yoder and A. Razavi, "Strangler Patterns," in Proc. 27th Conf. Pattern Languages of Programs (PLoP '20), ACM, 2021, pp. 1–12. DOI: 10.1145/3486607.3486751