International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(21s) 2026 pp. 1996-2006
https://doi.org/10.70917/ijcisim-2026-5975
Article

Recovery Debt: A Drift-Aware Verification Framework for Provable Recoverability in Hybrid VMware and Cloud Estates

Venkata Ravi Sridhar Potunuru

Bright Horizons Family Solutions, Newton, Massachusetts, United States
ORCID: 0009-0009-7806-0437

Abstract: Disaster recovery testing has not kept pace with how quickly modern hybrid infrastructure changes between test cycles. Chaos engineering, the discipline most closely associated with resilience validation today, assumes infrastructure is disposable: a region or availability zone can be destroyed deliberately and rebuilt from code, and the exercise itself validates both the failure response and the underlying automation. That assumption breaks down for the large population of enterprises still operating persistent VMware estates alongside cloud platforms, where tearing down a production virtualization layer is neither safe nor practical. Lacking a testing discipline suited to that constraint, most hybrid organizations default to calendar-based recovery testing, typically quarterly or annual, which certifies recoverability only for the moment the test executed. This article introduces recovery debt: a measure of the accumulating gap between an environment's last validated recovery state and its current live state, driven by configuration drift, undocumented infrastructure growth, and organizational change occurring between test cycles. Building on that measure, it proposes a drift-aware verification framework consisting of a tiered, non-destructive drill taxonomy suited to non-disposable infrastructure, paired with an automated trigger that schedules revalidation based on measured infrastructure change velocity rather than a fixed calendar. The framework is evaluated against documented operational testing practice, current industry survey data on disaster recovery readiness, and a retrospective practitioner account of an enterprise ransomware recovery used to illustrate where calendar-based assumptions held and where they understated risk. Organizations operating durable hybrid infrastructure need a measurable, auditable way to know whether a recovery plan validated months ago still reflects the environment it is meant to protect; this article argues that recovery debt is that measure.

Keywords: disaster recovery testing; recovery debt; configuration drift; hybrid cloud; VMware Site Recovery; RTO/RPO validation; continuous validation; infrastructure as code

1. Introduction

A recovery test answers a narrow question: could this environment have recovered at the moment the test executed? It says little about whether the same environment can recover today, months later, after new virtual machines have been provisioned, new application dependencies have been introduced, and configuration baselines have quietly drifted from what was validated. Federal contingency-planning guidance has recognized this limitation for over a decade. NIST Special Publication 800-34 directs organizations to conduct testing, training, and exercise activities not strictly on a fixed calendar but following organizational or system changes significant enough to affect recovery assumptions [1]. That guidance is rarely operationalized in practice. Recent industry survey data shows how rarely: only 20 percent of enterprises report constantly reviewing and updating their disaster recovery plans, and 31 percent have not updated their plans in more than twelve months [11]. Most organizations, including those with mature backup and replication programs built on tools such as VMware Site Recovery Manager, Azure Site Recovery, and Rubrik, default to fixed quarterly or annual cadences because no widely adopted method exists for measuring how much an environment has changed or how much that change has eroded confidence in the last validated result.



Copyright: © 2026 by the authors

The gap is more consequential in hybrid enterprise infrastructure than in cloud-native environments, and it is in hybrid environments that it is least addressed. Chaos engineering assumes infrastructure is disposable: a region or availability zone can be destroyed deliberately and rebuilt from code, and the exercise itself validates both the failure response and the underlying automation [2], [3]. This assumption is evident in the actual adoption of chaos engineering tooling. A 2025 empirical study of 971 open-source repositories across ten widely used chaos engineering tools found that instance-termination and network-fault categories made up the large majority of documented fault-injection scenarios, while application-level, non-destructive fault injection accounted for only a small share of observed cases [12]. That distribution reflects a tooling ecosystem organized around components designed to be killed and automatically rescheduled, Kubernetes pods, and cloud instances that "can be stopped, restarted, or relocated" [12], an operating model fundamentally different from a persistent VMware estate spanning physical data centers. ESXi hosts, storage arrays, and directory services represent durable capital and durable states. An organization cannot destroy its production virtualization layer to learn whether its recovery plan works, no matter how appealing the chaos-engineering feedback loop looks on paper.

Recent survey data confirms this gap has not closed as cloud adoption has grown. Forty-eight percent of enterprises still struggle to adapt traditional on-premises disaster recovery strategies to cloud environments, even as 89 percent identify disaster recovery as a key focus area for their cloud estates [11]. The result is a testing-discipline gap that runs in one direction only. Cloud-native organizations have a rigorous, repeatable way to manufacture failure on demand and learn from it continuously. Hybrid organizations, which still represent a substantial share of enterprise infrastructure deployments, are left largely with periodic, non-destructive failover drills that certify a single point in time and go stale immediately afterward. Nobody disputes that these drills are valuable when they run. The problem is what happens in the months between them, when most of the risk accumulates.

This article addresses that gap directly. It introduces recovery debt as a way to make the erosion of recovery confidence over time measurable rather than assumed, proposes a tiered verification taxonomy designed for infrastructure that cannot be destroyed for the sake of testing it, and grounds both in a retrospective practitioner account of an actual enterprise ransomware recovery.

1.1 Contributions

This article makes three contributions. First, it defines recovery debt as an operationalizable metric for the gap between an environment's last validated recovery state and its current live state, distinguishing it from the static, point-in-time readiness scores that dominate existing commercial and patented approaches. Second, it proposes a tiered, non-destructive verification taxonomy that isolates blast radius using mechanisms already available in enterprise virtualization and cloud platforms, without requiring the destructive infrastructure teardown that chaos engineering assumes. Third, it describes an automation architecture that converts recovery debt from a concept into a scheduling trigger, operationalizing the change-driven retesting principle that NIST guidance states but does not instrument [1].

1.2 Organisation of the Paper

Section 2 positions this work against chaos engineering literature, VMware and Azure recovery tooling, and existing static readiness-scoring approaches, including patented methods. Section 3 defines the recovery debt model and its constituent change-velocity signals, with a worked illustration of the weighting logic. Section 4 introduces the tiered, non-destructive verification taxonomy. Section 5 describes the automation architecture that converts recovery debt into scheduling triggers. Section 6 examines a retrospective practitioner account of an enterprise ransomware recovery. Section 7 discusses applicability and limitations, and Section 8 concludes with directions for future work.

2. Related Work and Positioning

The individual elements this article draws on are each established in isolation, but no existing body of work combines them into a measurable, decay-aware verification discipline suited to non-disposable infrastructure.

2.1 Chaos Engineering and the Disposable-Infrastructure Assumption

Chaos engineering platforms provide mature tooling for injecting failure and validating recovery time and recovery point objectives, primarily in disposable, cloud-native infrastructure [2], [3]. The empirical study of the adoption of chaos engineering tools referenced above is instructive here. Across the repositories analyzed, instance-termination and network-fault categories dominated the fault taxonomy, while application-level fault injection, the category least dependent on the target infrastructure being disposable, was the smallest and most underrepresented [12]. This is not a criticism of chaos engineering as a discipline. It is evidence that the tooling ecosystem built around it has organized itself around an operating model, cloud-native and container-orchestrated, that does not extend cleanly to a persistent virtualization estate. AWS's own Well-Architected guidance situates chaos engineering as a resiliency-testing practice within cloud-native architectural patterns, reinforcing the same orientation [3]. None of this is a shortcoming of the discipline. It is simply evidence that chaos engineering was built for a different substrate than the one most hybrid enterprises actually run.

2.2 Non-Destructive Test Failover Mechanics

VMware Site Recovery supports non-destructive test failovers and dependency-ordered priority groups, letting an application tier recover in a defined startup sequence inside an isolated test network without touching production [4]. VMware Cloud Disaster Recovery extends this further with continuous health checks that verify a recovery plan's integrity automatically, on a fixed 30-minute interval, flagging when environmental or configuration changes have invalidated the plan's directives [5]. These are valuable operational mechanics, and the health-check interval is itself telling: VMware's own tooling already recognizes that plans can go stale between validations. What neither mechanism provides is a way to quantify how much staler a plan has become, or to trigger a full revalidation drill based on the magnitude of underlying change rather than a fixed interval. A health check that runs every 30 minutes and a drill that runs once a quarter are solving adjacent but distinct problems, and the gap between them is exactly where recovery debt accumulates unmeasured.

2.3 Static Readiness Scoring: Patented and Commercial Approaches

A narrower body of prior art directly addresses readiness measurement, so we need to carefully distinguish this article's contribution from it. A patented Recovery Maturity Model evaluates an environment against a weighted matrix of categories, such as recovery configuration completeness and RTO and RPO achievability, gating whether disaster recovery testing may proceed based on the resulting score [6]. The scoring mechanism in that patent is explicitly point-in-time and human-initiated: rescoring occurs "during, for example, quarterly reviews, or following major upgrades," with a trend-analysis display tracking score changes across successive manual runs rather than a continuous or automatically triggered revalidation [6]. Commercial self-assessment tools follow a similar pattern, offering a single confidence score derived from survey responses at a moment [9]. Both approaches ask a version of the same question: How ready were we the last time someone checked? Neither asks how much readiness has eroded since.

Not every scoring patent in this space is static, and the distinction matters enough to draw out explicitly rather than gloss over. A separate patented approach to enterprise resiliency scoring incorporates temporal weighting, discounting the influence of older incident data as it ages, and re-triggers failover-site selection in response to detected interruptions [7]. That patent demonstrates the industry already recognizes staleness as a scoring problem, at least for real-time application failover decisions. It addresses a different problem than the one this article targets, though. It scores live application resiliency for failover routing, not the confidence that a periodic, non-destructive disaster recovery drill remains valid months after it last ran. Recovery debt applies a similar temporal-decay intuition to that specific, currently unaddressed problem: the recovery-testing cadence gap, rather than real-time failover routing.

What both categories of prior art share, despite their real differences, is that neither treats the interval between validation and the present moment as a variable computed from the infrastructure's own observable rate of change and used to schedule the next non-destructive drill. That is precisely what recovery debt makes explicit, and it is the specific gap this article's contribution closes.

2.4 Infrastructure Drift as a Measured Phenomenon

Configuration drift is not a hypothetical concern raised for rhetorical effect; it is an empirically documented one. State reconciliation, the process by which infrastructure-as-code orchestrators compare declared and live infrastructure states, is itself a source of real-world defects. An empirical study of 5,110 state reconciliation defects mined from a widely used orchestrator's codebase found that reconciliation logic, the very mechanism meant to detect divergence between desired and actual state, fails in categorizable, recurring ways, including cases where an orchestrator misclassified a virtual machine's actual operating state entirely [18]. It is worth being precise about what this finding does and does not establish: it demonstrates that the mechanics of state reconciliation are themselves defect-prone at a meaningful scale, not that infrastructure drift as a broader operational phenomenon is identical to reconciliation-tool bugs. The two are related but distinct, and this article treats reconciliation-defect data as evidence that drift detection is a genuinely hard, actively worked engineering problem rather than as a direct measurement of drift's prevalence.

Separately, recent work on automated infrastructure-as-code reconciliation demonstrates that drift, defined as out-of-band changes made outside the infrastructure-as-code source of truth, is common enough in practice to justify purpose-built tooling. One such system was evaluated against 372 realistic drift scenarios across real-world Terraform projects [19], a scale that suggests drift is a routine operational condition rather than an occasional anomaly. Drift detection already exists as a mechanism for reconciling declared and live infrastructure state [8], but it has not previously been connected to scheduling disaster recovery tests. This article makes one of its central moves by connecting observed drift to a recovery-testing trigger instead of treating it purely as an infrastructure-management housekeeping problem.

2.5 A Companion Line of Work: Recovery Architecture

A separate, complementary line of work by this author addresses how a recovery environment should itself be architected: generated on demand from version-controlled infrastructure definitions, isolated from production trust boundaries by default, and reconstituted in an identity-first order [16]. That architecture is one way to satisfy the tiers proposed in Section 4, but it is not this article's subject. The verification framework below is architecture-agnostic. It applies equally to a dynamically generated clean-room recovery target or to a conventional standby site, because its concern is not how the recovery environment is built but how confidence in its correctness is measured and maintained over time. One line of work answers the question of what to build. This one answers how to know, continuously, that it still works.

3. The Recovery Debt Model

3.1 Defining Recovery Debt

Recovery debt is defined here as the accumulated divergence between the infrastructure state last successfully validated through a recovery drill and the infrastructure's current live state. Debt is near zero immediately following a successful, evidence-backed validation and increases as two factors accumulate: the rate at which the environment changes and the elapsed time since the last validation covering the affected systems. The term deliberately parallels technical debt, and the parallel is more than a naming convenience. A small, well-understood gap is a normal and acceptable operating condition in both cases. An unmeasured and unpaid gap compounds silently until it surfaces at the worst possible moment: typically during an actual incident rather than a scheduled test, in the same way unaddressed technical debt tends to surface during a production outage rather than a code review.

This framing carries a direct practical consequence. Under a calendar-based testing regime, an organization has no way to distinguish a protection group that has changed minimally since its last validation from one that has undergone substantial infrastructure churn, application-dependency changes, or turnover in the institutional knowledge of who understands the environment. Both are treated identically: valid until the next scheduled test,

regardless of how much confidence in that validation has actually eroded in the meantime. Recovery debt makes that erosion an observable quantity rather than an unexamined assumption baked into the calendar.

3.2 Measuring Change Velocity in Hybrid Estates

A practical recovery-debt score requires proxies for change that are already observable in a reasonably well-instrumented hybrid environment, rather than a new class of monitoring built specifically for this purpose. Five candidate signals form a workable starting set: virtual-machine inventory deltas between a protected group and its last-tested manifest; changes to application dependency chains that alter startup ordering; Active Directory and DNS topology changes across multi-region sites; certificate and secret rotation events that could invalidate a previously tested trust chain; and infrastructure-as-code state deltas surfaced by routine drift detection, which already exists as a mechanism for reconciling declared and live infrastructure state in tools such as Terraform [8]. The empirical prevalence of reconciliation defects and drift scenarios documented in Sections 2.3 and 2.4 supports treating this last signal class as a genuine, recurring source of unmeasured change rather than an edge case worth footnoting and ignoring [18], [19].

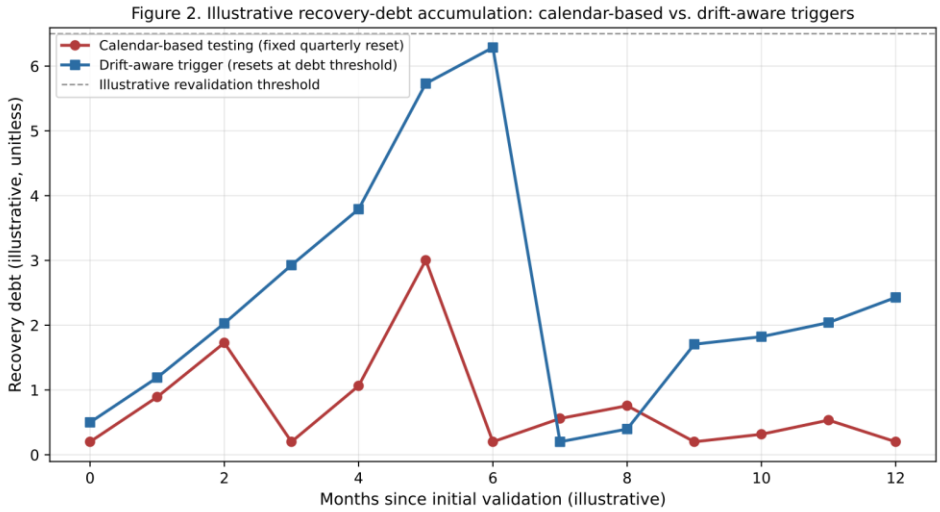


Figure 2. Illustrative recovery-debt accumulation: calendar-based vs. drift-aware triggers (conceptual model, not empirical data).

Each signal should be weighted by its blast-radius relevance rather than treated uniformly, and a brief illustration makes the point concretely. Consider two changes to the same previously validated application tier. The first is the addition of a new production dependency, a service the application now calls at runtime that did not exist, or was not integrated, at the time of the last drill. The second is a routine patch-level update to a non-critical host in the same tier. Both are, in a naive change-counting sense, "one change." Their implications for recovery confidence are not remotely comparable. The new dependency means the last validated recovery sequence may no longer reflect the application's actual startup requirements: if the dependency is not itself protected and recoverable in the correct order, the previously validated drill result no longer describes a recovery path that would actually work. The patch, by contrast, is unlikely to affect recovery sequencing or dependency ordering at all. A recovery-debt score that weighted these two events identically would be actively misleading, understating risk in the first case and overstating it in the second.

This weighting logic mirrors, and is directly adapted from, risk-weighted scoring already used for identity-account remediation in hybrid environments [17], and is extended here from accounts to infrastructure state generally. The parallel is not incidental. Both problems require converting a heterogeneous set of change signals, some clearly high-stakes and some routine, into a single prioritization score without treating every change as equally consequential.

An identity system that flagged every password reset with the same urgency as a privilege escalation would be useless within a week; the same logic applies to an infrastructure change-tracking system that cannot distinguish a new production dependency from a patch.

4. A Tiered, Non-Destructive Verification Taxonomy

Because hybrid estates cannot be destroyed to be tested, verification must be structured in tiers that each isolate their blast radius using non-disruptive mechanisms already available in enterprise virtualization and cloud platforms, principally isolated test networks and storage-level snapshot mounts that never touch production traffic [4], [5]. Table 1 defines three tiers, moving from single-component validation to full cross-domain reconstitution, each with an explicit evidentiary pass condition rather than a subjective outcome, since a drill that ends in "it seemed to work" produces no auditable evidence and defeats the purpose of measuring recovery debt in the first place.

Tier	Scope	Mechanism	Evidentiary Pass Condition
Component	Single VM / application snapshot restore	Isolated test network, storage snapshot mount	Application health check green within defined RTO
Service	Dependency-ordered multi-tier application recovery	Isolated test network, priority groups	Full transaction path validated end-to-end
Cross-Domain	Identity, DNS, and application layers reconstituted together	Isolated environment, full-stack reconstitution	An authenticated end-user transaction completes without manual intervention

Table 1. Tiered, non-destructive verification taxonomy.

Each tier is deliberately silent on which underlying recovery architecture is exercised. A component-tier drill validates a snapshot-based restore regardless of whether the target is a persistent standby site or a clean room generated on demand at the moment of the drill [16]. The taxonomy measures whether recovery succeeded and how long it took, not how the recovery target came into existence. This separation of concerns is deliberate and load-bearing: it is what allows the framework to apply uniformly across organizations with very different recovery architectures, rather than requiring every adopter to first standardize on a single recovery-target design before the verification discipline can be used [22].

The three tiers are also cumulative in a specific sense. A cross-domain drill implicitly re-validates the service-tier dependencies it depends on, and a service-tier drill implicitly re-validates the component-tier restores it comprises. This means an organization does not need to run all three tiers on every protection group at every trigger event; the automation architecture in Section 5 selects the appropriate tier based on which change signals crossed the threshold, rather than defaulting to the most expensive tier every time debt accumulates past a certain point.

5. Automating Drift Measurement and Revalidation Triggers

The practical value of recovery debt depends on converting it from a concept into an automated trigger, since a metric nobody acts on is not meaningfully different from no metric at all. A scheduled job, built on the same automation stack most infrastructure teams already operate, Ansible Tower and Azure Automation being representative, periodically computes each protection group's recovery-debt score from the signals described in Section 3.2 and compares it against a threshold calibrated to that group's business criticality. Crossing the threshold schedules a revalidation drill at the appropriate tier from Section 4, independent of when the group was last tested on the calendar.

This operationalizes the trigger-based testing principle that NIST SP 800-34 states but does not instrument [1]. Rather than asking an administrator to judge subjectively whether a change was significant enough to warrant retesting, a judgment that depends heavily on which person happens to be paying attention that week, the framework

measures it directly and consistently. Table 2 contrasts this approach with the static readiness-scoring model represented in existing patent literature, highlighting where the trigger mechanism, rather than the scoring inputs alone, is the meaningful point of difference.

Dimension	Static RMM Patent [6]	Real-Time Resiliency Scoring [7]	Recovery-Debt Trigger (this article)
Trigger basis	Manual review cadence	Live application interruption	Measured infrastructure change velocity
Rescoring cadence	Quarterly / post-upgrade, human-initiated	Continuous, real-time	Continuous, automated, threshold-based
Auditability	Trend display across manual runs	Real-time score, failover-routing focused	Auditable debt score tied to DR drill scheduling

Table 2. Comparison of recovery-debt triggers against static and real-time patented scoring approaches.

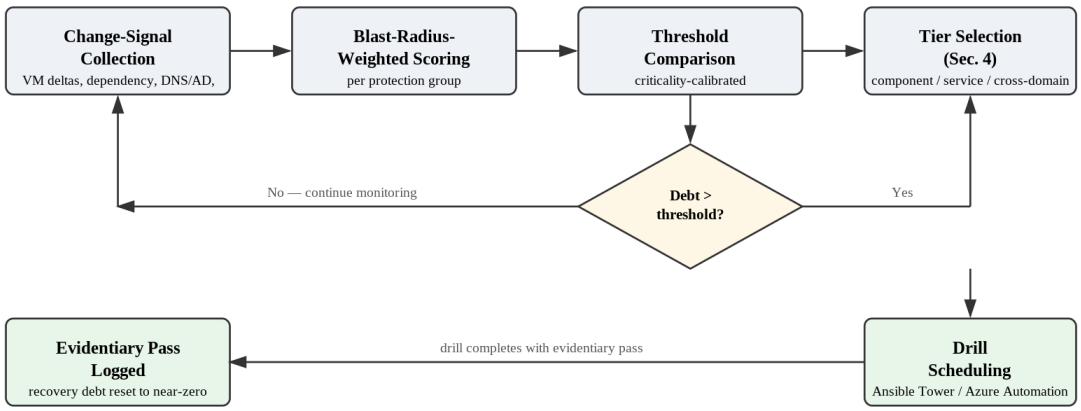


Figure 1. Automated drift-measurement-to-revalidation-trigger pipeline.

Figure 1. Automated drift-measurement-to-revalidation-trigger pipeline.

A practical note on false positives is worth adding here, since any automated trigger invites the question of how often it fires unnecessarily. The weighting scheme in Section 3.2 is designed to be blast-radius-aware, which means that routine low-impact changes should not push a protection group's score across a criticality-calibrated threshold. The threshold itself needs periodic review as an organization's infrastructure-as-code maturity and change velocity evolve, but that is a tuning exercise analogous to alert-threshold tuning in any monitoring system, not a fundamental limitation of the approach [20].

6. A Retrospective Practitioner Account: Enterprise Ransomware Recovery

A rigorous test of this framework would ideally compare recovery-debt scores across many organizations against subsequent incident outcomes. That dataset does not yet exist, and this section does not attempt to construct one or imply otherwise. What follows is a retrospective, illustrative account drawn from professional experience with enterprise ransomware recovery, presented qualitatively and without claim to validated, independently verified figures. It is offered to illustrate how the taxonomy in Section 4 would characterize a real environment going into an incident, not as a benchmarked case study, and readers should weigh it accordingly.

In the organization concerned, disaster recovery testing for individual applications was scheduled throughout the year rather than relying on a single annual exercise, and replication bandwidth for VMware-based recovery was tuned against defined recovery point objectives ahead of the incident. In the taxonomy of Section 4, that practice

functioned as ongoing service-tier validation, though it was executed on a rolling per-application calendar rather than a change-driven trigger. When the incident occurred, virtualization infrastructure was rebuilt from a known-good foundation, followed by staged recovery of directory and messaging services, and validated using enterprise backup platforms. Core production systems were, in this account, confirmed operational within a period the organization characterized internally as consistent with favorable outcomes relative to the broader industry recovery data reported in recent ransomware surveys, where recovery costs and elapsed time have both trended upward year over year [10], [11]. The broader environment was normalized over a longer subsequent period. These characterizations are presented here as qualitative practitioner recollection rather than as audited figures, consistent with the absence of a formal, evidence-backed benchmark for this specific incident, and no specific numeric timeline is asserted as verified fact.

Viewed through the recovery-debt model, the organization's rolling test cadence functioned as a partial, manually scheduled approximation of the drift-aware trigger this article proposes. It was not nothing: a rolling per-application calendar is meaningfully better than a single annual exercise, and it likely contributed to the outcome being characterized internally as favorable. At the same time, the recovery relied substantially on personnel judgment to sequence and prioritize workloads during the event itself, rather than on a pre-scored, tiered validation record indicating which systems carried the least-tested, and therefore highest, recovery debt at the moment the incident began. That is precisely the condition the framework is designed to close: converting institutional knowledge about what has and has not been recently, rigorously validated into a measured, auditable artifact available before an incident, rather than reconstructed under pressure during one [25].

7. Discussion: Applicability and Limitations

The framework applies most directly to enterprises operating a durable, hard-to-destroy virtualization estate alongside cloud platforms, particularly organizations handling regulated or sensitive data, where demonstrable, auditable recovery verification is increasingly a compliance expectation rather than a discretionary best practice. Recent survey data supports the scale of this applicability gap: nearly half of enterprises report difficulty adapting on-premises disaster recovery strategies to cloud environments even as the overwhelming majority treat disaster recovery as a cloud priority [11], and mission-critical recovery time objectives are met successfully in only a modest majority of cases across surveyed organizations [11]. Regulatory pressure is also reshaping the applicability picture in ways worth naming directly: frameworks such as the EU's Digital Operational Resilience Act are increasingly serving as external catalysts for exactly this kind of measurable, auditable recovery verification, particularly around third-party and vendor dependency risk [11], which suggests the compliance case for a framework like this one will likely strengthen rather than weaken over the next several years.

Its principal limitation at this stage is that the weighting scheme proposed for change-velocity signals in Section 3.2 has not yet been calibrated against a multi-organization dataset correlating recovery-debt scores with actual incident outcomes. The account in Section 6 is a single retrospective, qualitative reflection, not a validated benchmark, and should not be read as demonstrating a quantified predictive relationship between recovery-debt scores and incident outcomes. Establishing that relationship rigorously would require exactly the kind of longitudinal, cross-organization data collection effort described as future work in Section 8.

A second limitation is organizational rather than technical. The framework assumes an environment already instrumented with infrastructure-as-code and automated configuration baselines sufficient to compute drift signals, which not every hybrid enterprise has achieved, though the trend toward such instrumentation and the empirical documentation of drift as a measurable phenomenon within it are well established [8], [18], [19]. An organization still managing infrastructure primarily through manual, undocumented changes would need to invest in that instrumentation first; recovery debt is not a substitute for infrastructure-as-code adoption; it is a discipline that becomes available once that adoption has already happened [20], [21].

A third limitation concerns the reconciliation-defect literature cited in Section 2.4. State reconciliation defects, as empirically documented, describe failures in the logic that detects divergence between declared and live state, which is a related but distinct problem from the broader operational concept of infrastructure drift this article addresses.

Readers should not conflate the two [22], [23]. The framework proposed here consumes drift-detection output as one input signal among several, and it is not itself a claim about the reliability of any particular orchestrator's reconciliation logic; a bug in a reconciliation tool would be a data-quality problem for this framework's inputs, not a validation of the framework's own thesis [24].

A fourth, more speculative limitation is worth naming rather than avoiding. The weighting scheme in Section 3.2 currently requires manual calibration per protection group, based on judgment about blast-radius relevance. That calibration burden could itself become a source of drift if left unattended, a criticality assessment set once at implementation time and never revisited as an application's actual business importance shifts. Any organization adopting this framework should treat the weighting scheme itself as subject to periodic review, not as a one-time configuration exercise [25].

8. Conclusion and Future Work

Recovery testing has treated a single passed drill as durable proof of readiness for far too long, an assumption chaos engineering never had to make because its target infrastructure is disposable and can be retested constantly at near-zero marginal cost. Hybrid enterprises do not have that luxury, and the industry's response has largely been calendar-based testing that certifies a moment and says little about the months of undocumented change that follow it. Recovery debt reframes readiness as something that decays and must be continuously remeasured, not something proven once and assumed indefinitely. Paired with a non-destructive, tiered drill taxonomy suited to infrastructure that cannot be torn down for the sake of testing it, the framework offers a way to make hybrid enterprises' existing testing discipline, already more mature in practice than the literature reflects, measurable, auditable, and trigger-driven rather than merely periodic. Future work should focus on two directions, in a deliberate order. The first is developing an open, reusable reference implementation of the drift-scoring mechanism described in Section 5, so the weighting scheme proposed here can be tested and refined across environments with meaningfully different change velocities and criticality profiles, rather than remaining a single author's proposed weighting untested outside one practitioner's experience. The second, and the more consequential of the two, is building the longitudinal, multi-organization dataset needed to establish whether recovery debt, measured before an incident, actually predicts recovery outcome during one. This is the single most important open question that the retrospective account in Section 6 cannot, on its own, resolve, and it is the question on which the entire practical value of the framework ultimately rests.

Declarations

Acknowledgements: Not applicable.

Funding: This research received no external funding.

Author contributions: V. R. S. Potunuru conceived the recovery debt model, developed the verification taxonomy and automation architecture, conducted the literature positioning, and wrote the manuscript.

Conflict of interest: The author declares no conflict of interest.

Data availability statement: Not applicable. This article does not report new experimental datasets; the retrospective account in Section 6 is presented qualitatively.

List of abbreviations: DR - Disaster Recovery; RTO - Recovery Time Objective; RPO - Recovery Point Objective; IaC - Infrastructure as Code; SRM - Site Recovery Manager; VM - Virtual Machine; NIST - National Institute of Standards and Technology.

The author used Claude (Anthropic, claude-sonnet model) to assist with manuscript drafting and language refinement. The author takes full responsibility for the accuracy, originality, and integrity of all content presented in this manuscript and confirms that all data, analysis, and conclusions reflect independent work and verified sources.

Author Biography

Venkata Ravi Sridhar Potunuru is a Principal Integration Engineer at Bright Horizons Family Solutions, Watertown, Massachusetts, USA, where he leads hybrid-cloud disaster recovery architecture, infrastructure-as-code adoption, and datacenter modernization initiatives. He holds a Master of Science in Electrical Engineering from the University of Texas at El Paso and a Bachelor of Engineering in Electronics and Communication from Anna University, India. He has more than fifteen years of enterprise infrastructure experience, including over nine years of dedicated VMware administration, and holds Microsoft Certified: Azure Solutions Architect Expert and Azure Security Engineer Associate certifications, along with VMware Certified Professional 5 (Data Center Virtualization) and Cisco Certified Network Associate credentials.

REFERENCES

1. National Institute of Standards and Technology, "SP 800-34 Rev. 1: Contingency Planning Guide for Federal Information Systems," 2010 (updated 2010-11-11). Available: https://csrc.nist.gov/pubs/sp/800/34/r1/upd1/final
2. Gremlin, "Testing Disaster Recovery with Chaos Engineering." Available: https://www.gremlin.com/community/tutorials/testing-disaster-recovery-with-chaos-engineering
3. Amazon Web Services, "REL12-BP05 Test resiliency using chaos engineering," AWS Well-Architected Framework. Available: https://docs.aws.amazon.com/wellarchitected/2023-04-10/framework/rel_testing_resiliency_failure_injection_resiliency.html
4. Broadcom/VMware, "An Introduction to Disaster Recovery (Site Recovery)." Available: https://www.vmware.com/docs/an-introduction-to-disaster-recovery-site-recovery
5. Broadcom/VMware, "VMware Cloud Disaster Recovery Documentation." Available: https://docs.vmware.com/en/VMware-Cloud-Disaster-Recovery/index.html
6. U.S. Patent 9,208,006 B2, S. Jones and J. Maldonado, "Recovery Maturity Model (RMM) for readiness-based control of disaster recovery testing," issued Dec. 8, 2015. Available: https://patents.google.com/patent/US9208006B2/en
7. U.S. Patent 11,194,628 B2, J. Reyes, B. J. Klingenberg, and H. Yaswi, "Workload allocation utilizing real-time enterprise resiliency scoring," issued Dec. 7, 2021. Available: https://patents.google.com/patent/US11194628B2/en
8. HashiCorp, "Detecting and Managing Drift with Terraform." Available: https://www.hashicorp.com/en/blog/detecting-and-managing-drift-with-terraform
9. Cutover, "IT Disaster Recovery Confidence Index." Available: https://cutover.com/content-hub/it-disaster-recovery-confidence-index
10. Sophos, "The State of Ransomware 2026." Available: https://www.sophos.com/en-us/content/state-of-ransomware
11. Cutover (with Insight Avenue), "Third Annual IT Disaster and Cyber Recovery Trends and Insights Report," 2025. Available: https://www.cutover.com/content-hub/annual-it-disaster-cyber-recovery-trends-insights-report-2025
12. J. Owotogbe, I. Kumara, D. Di Nucci, D. A. Tamburri, and W.-J. van den Heuvel, "Chaos Engineering in the Wild: Findings from GitHub," arXiv:2505.13654, 2025. Available: https://arxiv.org/html/2505.13654v1
13. Forrester, "The State of Disaster Recovery Preparedness in 2024." Available: https://www.forrester.com/report/the-state-of-disaster-recovery-preparedness-in-2024/RES180378
14. Astute Analytica, "Disaster Recovery As a Service Market to Hit Valuation of US$ 107.50 Billion By 2033," 2025. Available: https://www.globenewswire.com/news-release/2025/07/31/3125005/0/en/
15. Cristie Software, "The State of Disaster Recovery and Cyber-Recovery, 2024–2025." Available: https://www.cristie.com/news/the-state-of-disaster-recovery-and-cyber-recovery-20242025
16. V. R. S. Potunuru, "Dynamic Clean Room Orchestration Using Infrastructure as Code: A Vendor-Agnostic Reference Architecture and Identity-First Reconstitution Model for Ransomware Recovery," manuscript in preparation (companion article, concurrent submission to a separate venue).
17. V. R. S. Potunuru, "Engineering Attack-Resistant Identity: A Layered, Automation-Enforced Framework for Hardening Hybrid Enterprise Access," manuscript in preparation (companion article, concurrent submission to a separate venue).
18. Md M. Hassan, J. Salvador, S. K. Karmaker Santu, and A. Rahman, "State Reconciliation Defects in Infrastructure as Code," Proc. ACM Softw. Eng., vol. 1, no. FSE, Art. 83, Jul. 2024, doi: 10.1145/3660790.
19. Z. Yang, H. Guan, V. Nicolet, B. Paulsen, J. Dodds, D. Kroening, and A. Chen, "Automated Cloud Infrastructure-as-Code Reconciliation with AI Agents," arXiv:2510.20211, 2025. Available: https://arxiv.org/abs/2510.20211
20. H. Gaggar, "Two Planes of Truth: Execution Fidelity and BehavioralCorrectness in Tool-Using LLM Agents," International Journal of Computational and Experimental Science and Engineering, vol. 10, no. 3, 2024. Available: https://doi.org/10.22399/ijcesen.5498
21. A. Bhatnagar and V. R. K. Bonagiri, "Operational Excellence through Cloud-Native Distributed Systems," International Journal of Computational and Experimental Science and Engineering, vol. 12, no. 2, 2026. Available: https://doi.org/10.22399/ijcesen.5207
22. K. Rekulapalli, S. R. Kagga, V. Ayyagari, A. Akula, and R. Akula, "AI in HRM: Enhancing Workforce Competency and Organizational Effectiveness," SSRN, 2026. Available: https://ssrn.com/abstract=6762998

23. H. Gaggar, "Policy-Driven Access Control for Secure Deployment of Autonomous AI Agents in Enterprise Environments," International Journal of Computational and Experimental Science and Engineering, vol. 11, no. 4, 2025. Available: https://doi.org/10.22399/ijcesen.5497
24. K. Rekulapalli, H. Nilagiri, and M. Vakkanthula, "A comparative evaluation of national FHIR Implementation Guides: Strategic methodologies for enhancing interoperability, regulatory alignment, and clinical comprehensiveness in global health IT," Health Systems, pp. 1–13, 2026.
25. R. K. Shaw, "Kerberos vs. Public Key Authentication in z/OS Environments: Performance, Scalability, and Security Trade-offs," in 2026 9th International Conference on Computing Methodologies and Communication (ICCMC), pp. 1305–1312, IEEE, 2026.