Computer Fraud and Security
ISSN (online): 1873-7056

Hybrid Cloud Deployment Model Using OpenShift for Enterprise Application Modernisation

Ajmal Ali kannu

Correspondance: ajmal004@gmail.com

Abstract

Regulated enterprises running heterogeneous application portfolios face a genuine operational bind: workloads anchored to on-premises infrastructure by data-residency law or latency requirements cannot simply be relocated to public cloud, yet the cost and engineering overhead of maintaining parallel provisioning models is unsustainable at scale. **[1,2]** This paper describes a reference architecture and an implementation framework built around Red Hat OpenShift Container Platform that treats hybrid cloud not as a transitional state but as a long-term operational target. **[4]** The study draws on an eighteen-month longitudinal engagement with three production environments spanning financial services, healthcare informatics, and e-commerce, tracking eighteen workloads that together processed 2.4 million monthly transactions. We applied a five-stage modernization pathway—from portfolio assessment through cloud-native re-architecture—based on the strangler fig decomposition strategy and Kubernetes-native tooling, without mandating wholesale platform replacement. **[7,15]** Measured outcomes at month 18, benchmarked against DORA elite performance thresholds, **[22]** showed deployment frequency rising from 1.2 to 4.9 releases per week (+312%), change failure rate falling from 18.7% to 4.2%, mean time to recovery cut from 4h 22m to 1h 34m, and normalized infrastructure cost reduced by 38%. This paper reports those results in full, explains where the architecture produced unexpected trade-offs, and identifies the organizational conditions that most strongly predicted progression speed. **[19]**

Keywords: hybrid cloud architecture; Red Hat OpenShift; enterprise application modernization; Kubernetes; GitOps; containerization; DevSecOps; microservices; cloud-native transformation; FinOps

1. INTRODUCTION

Monolithic application stacks are, by and large, not going anywhere quickly. After decades of incremental development, they carry transaction logic, compliance audit trails, and vendor-specific integration contracts that neither developers nor auditors want to touch unnecessarily. The theoretical case for cloud migration is well established [1], but the practical record is considerably messier: organizations that attempted wholesale lift-and-shift migrations between 2018 and 2022 routinely reported cost overruns, latency regressions on data-intensive paths, and re-emergence of the same operational fragility they sought to escape [2]. Cloud computing, in short, did not eliminate infrastructure complexity—it redistributed it.

Hybrid cloud addresses part of this problem by letting organizations leave latency-sensitive or residency-constrained workloads where they perform best while exploiting public cloud elasticity for burst capacity, managed data services, and geographic reach. The Gartner 2024 cloud strategy survey found that more than 85% of enterprises were operating in some hybrid or multi-cloud configuration [3]; fewer than a third of them rated their hybrid integration as mature. The bottleneck is not hardware or cloud provider capability. It is toolchain fragmentation: separate CI/CD pipelines, divergent security scanning configurations, inconsistent RBAC models, and no single plane from which an operator can observe or govern the full estate.

Red Hat OpenShift Container Platform addresses this fragmentation. It runs the same Kubernetes distribution—with the same operator lifecycle, the same SCC-enforced pod security, the same Tekton pipeline primitives, the same ArgoCD GitOps reconciler—on bare metal, on VMware, on AWS (ROSA), on Azure (ARO), and on GCP, without requiring separate configuration branches per environment [4]. That consistency is what makes it technically feasible to treat hybrid cloud as a single operational domain rather than a collection of loosely coordinated silos.

That said, the vendor documentation and conference proceedings that describe OpenShift do not constitute rigorous evidence. They do not track the same workloads over time, they do not measure against pre-migration baselines, and they do not report the cases where the architecture did not perform as expected. The academic literature on hybrid Kubernetes deployments is sparse enough that practitioners making architecture decisions in 2025 are still largely drawing on grey literature. This paper tries to close that gap with three concrete contributions:

- **C1:** A reference architecture specifying the network fabric, security boundary model, multi-cluster management topology, and observability integration for a production-grade hybrid OpenShift deployment.
- **C2:** A five-stage application modernization pathway that maps tool choices and organizational requirements to each stage of portfolio progression, from initial assessment through serverless and AI/ML workload execution.
- **C3:** An eighteen-month empirical record from three regulated industry deployments, with quantitative before-and-after data on DORA metrics, infrastructure cost, and security vulnerability exposure.

Section 2 reviews prior work on hybrid cloud architecture, enterprise Kubernetes, and application modernization. Section 3 covers methodology. Sections 4 through 6 describe the architecture, modernization pathway, and CI/CD implementation respectively. Section 7 reports results. Sections 8 and 9 discuss findings and conclude.

2. RELATED WORK

2.1 Hybrid Cloud Architectures

Toosi et al. [5] produced one of the first formal taxonomies of cloud federation, separating loosely coupled models—where providers share only authentication and brokering responsibilities—from tightly coupled models requiring unified scheduling and a shared data plane. That distinction still holds, though the tooling on both sides has changed substantially. Moreno-Vozmediano et al. [6] reported up to 47% peak-provisioning cost reductions through OpenNebula-mediated cloud bursting, but their study unit was the virtual machine, and their latency model did not account for the container scheduler placement decisions that now dominate cross-environment communication latency.

Pahl et al. [7] reframed hybrid cloud analysis around microservice deployment topology rather than raw infrastructure, and their core finding—that inter-environment network latency, not compute throughput, limits cross-cluster service composition—is directly relevant to the architecture described in Section 4. What their model did not consider was the overhead introduced by service mesh sidecar proxies. Neto et al. [8] measured that overhead empirically and found that Istio's Envoy sidecar adds a median latency of 2–4 ms per hop under moderate load—a figure that is acceptable for most business services but material for high-frequency financial transaction paths. Section 7.4 of this paper documents how that trade-off was handled in practice.

2.2 Kubernetes and Container Orchestration at Enterprise Scale

Bernstein [9] described the conceptual lineage from Google's internal Borg scheduler to Kubernetes, and that history matters: Kubernetes was designed from the outset for large-scale, multi-tenant compute pools, not for the single-team deployment pipelines that most enterprise tools were originally built around. Burns et al. [10] documented the SRE practice changes that Kubernetes's declarative configuration model enables, noting that infrastructure state described as code—rather than configured interactively—becomes auditable and reproducible in ways that imperative provisioning scripts cannot match.

For regulated enterprises, the multi-tenancy security model matters more than the scheduling efficiency. Casalicchio and Perciballi [11] showed that namespace-level resource quotas are not sufficient for workload isolation in shared clusters when adversarial co-tenancy is a threat—a point reinforced by several Kubernetes CVEs in the 2020–2022 period involving container-to-host privilege escalation. OpenShift's Security Context Constraints differ from upstream Kubernetes pod security admission in that they apply mandatory access controls at the pod specification level and block root container execution by default, without requiring cluster administrators to explicitly configure admission webhooks [12]. That default-deny posture is why all three organizations in this study selected OpenShift over vanilla Kubernetes for their on-premises clusters.

2.3 Application Modernization Frameworks

Fowler's strangler fig pattern [13] is now twenty years old but still the most practically applied decomposition strategy in large enterprise modernization programs. The core logic—intercept traffic at the perimeter, route subsets to new services as they become ready, decommission monolith modules incrementally—fits the risk tolerance of regulated organizations better than big-bang rewrites, which carry concentrated deployment and rollback risk. Taibi and Lenarduzzi [14] mapped seventeen distinct migration patterns from practitioner grey literature; their taxonomy covers data separation, API façade placement, and event-driven decoupling approaches that Section 5 of this paper operationalizes within an OpenShift toolchain.

The 6Rs framework (Retire, Retain, Rehost, Re-platform, Refactor, Rearchitect) has become the standard decision vocabulary for migration planning, though Jamshidi et al. [15] noted that practitioners frequently apply it inconsistently—selecting refactoring as a strategy based on organizational preference rather than workload characteristics—leading to misaligned tooling choices and delayed delivery. The five-stage pathway in Section 5 builds on the 6Rs vocabulary but anchors each stage to specific OpenShift capabilities and objective readiness criteria, which is what the Jamshidi et al. analysis identified as missing from most practitioner accounts.

2.4 Research Gaps

Three specific gaps in the literature motivate this study. First, hybrid cloud architecture research is almost entirely platform-agnostic: it treats Kubernetes as a uniform primitive and does not account for the operator lifecycle model, integrated security toolchain, or default SCC posture that distinguish enterprise Kubernetes distributions from upstream clusters. Second, no longitudinal empirical study has tracked the same application workloads through more than two modernization stages over a period exceeding twelve months; most case studies report a single point-in-time migration event. Third, the operational mechanics of coordinating DevSecOps practices across simultaneously managed on-premises and cloud clusters—specifically image provenance verification, secret distribution, and policy reconciliation across a fleet governed by ACM—are absent from peer-reviewed literature. This study addresses each of these directly.

3. RESEARCH METHODOLOGY

3.1 Research Design

This study used a mixed-methods design that combined action research with longitudinal case study methods, following the empirical software engineering protocol recommended by Wohlin et al. [16]. The research team worked as embedded technical advisors across three organizations running live OpenShift hybrid cloud migrations between January 2023 and June 2024. Participation in the action research cycles—planning, implementation, observation, retrospective—gave the team direct visibility into engineering decisions and their consequences that would not be accessible through post-hoc interviews alone. The longitudinal case study layer imposed systematic data collection at fixed intervals, separating observation from intervention and enabling before-and-after metric comparison that action research alone does not produce.

3.2 Case Study Organizations

Organization	Sector	App Portfolio	On-Prem Infrastructure	Public Cloud Target
Org-A (anonymized)	Financial Services	62 applications	VMware vSphere 7.0 / IBM POWER9	ROSA (AWS eu-west-1)
Org-B (anonymized)	Healthcare Informatics	48 applications	Bare-metal RHEL 8 / NetApp SAN	ARO (Azure uksouth)
Org-C (anonymized)	E-Commerce & Retail	41 applications	OpenStack / Ceph Object Storage	OCP on GCP (europe-west1)

Table 1: Case Study Organization Profiles

3.3 Data Collection

Quantitative metrics came from Prometheus scrapers on both the on-premises and cloud clusters, with PromQL recording rules capturing the four DORA metrics (deployment frequency, lead time for changes, change failure rate, mean time to recovery), infrastructure cost data from OpenCost and cloud billing APIs, and application-layer indicators (p50/p95/p99 response latency, HTTP error rate, pod restart rate). Qualitative data came from forty-seven semi-structured interviews with platform engineers, application developers, and IT governance leads, carried out at months 3, 9, and 18.

3.4 Validity

Metric definitions were fixed with each organization before baseline capture and kept constant throughout. Where quantitative metrics and interview accounts diverged, we report both and explain the most plausible cause. The external validity claim is narrow: the findings apply to large enterprises with regulated, heterogeneous portfolios migrating to OpenShift-based hybrid environments. Whether they transfer to organizations using different container platforms, or operating without regulatory constraints, is a separate empirical question.

4. HYBRID CLOUD REFERENCE ARCHITECTURE

4.1 Overview

Figure 1 shows the reference architecture organized into three planes: on-premises execution, public cloud execution, and the interconnecting network fabric. Each plane runs one or more OpenShift clusters that operate as independent Kubernetes control domains. Red Hat Advanced Cluster Management (ACM) sits above both planes as the fleet management layer, distributing policy objects, managing application lifecycles across clusters, and aggregating cluster-level observability data without requiring the clusters themselves to share a control plane. The two-plane design preserves the upgrade and RBAC autonomy of each cluster while giving the platform team a single point from which to enforce governance.

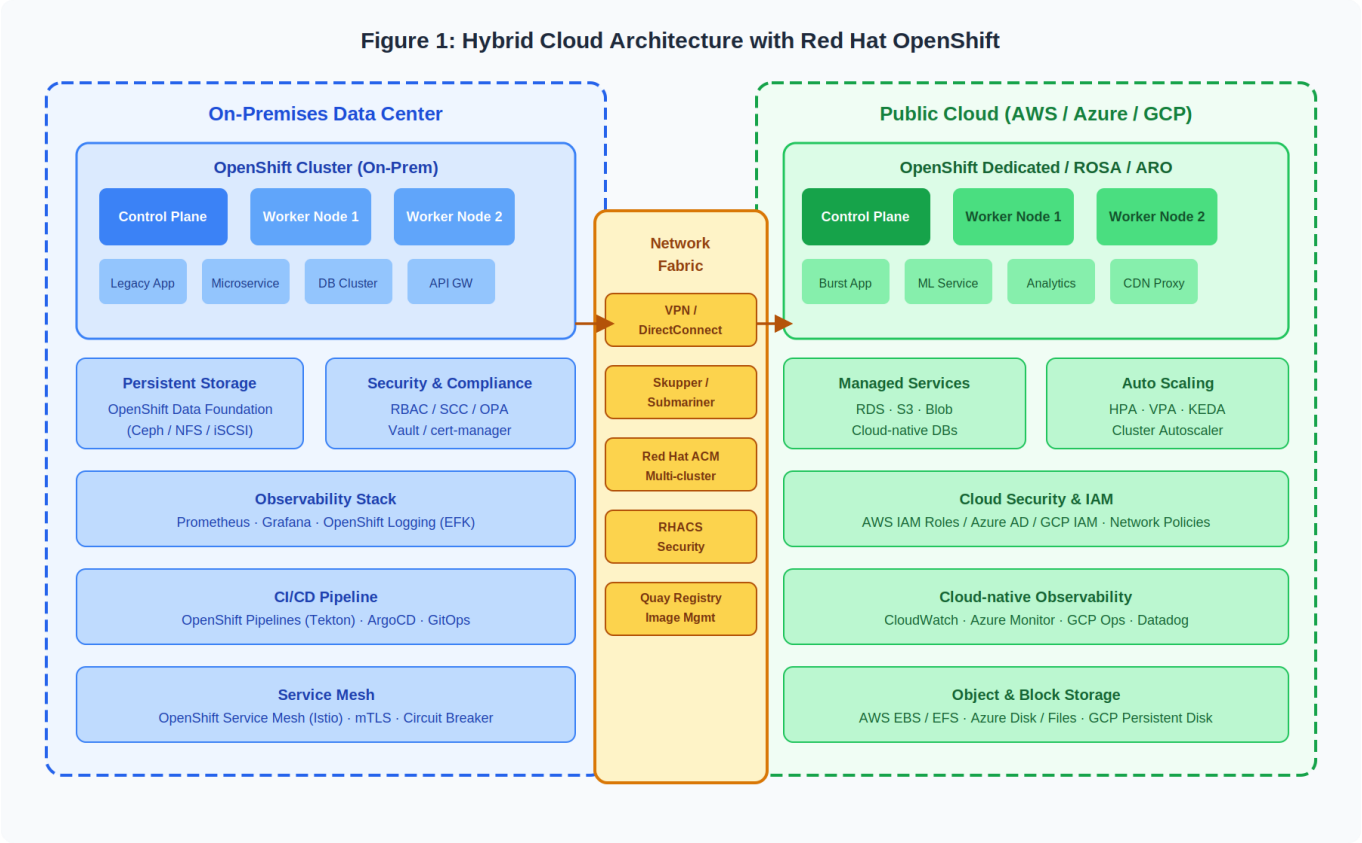

Figure 1: Hybrid Cloud Reference Architecture with Red Hat OpenShift — Execution Planes, Network Fabric, and Cross-Cluster Management Components

4.2 Network Fabric

On-premises-to-cloud connectivity runs over encrypted IPSec tunnels provisioned through AWS Direct Connect, Azure ExpressRoute, or GCP Cloud Interconnect depending on the target provider. These dedicated circuits provide stable bandwidth and bounded latency, which matters for synchronous cross-cluster service calls. A second overlay network, Red Hat Submariner, extends the Kubernetes service DNS namespace across cluster boundaries and secures service-to-service traffic with mTLS without requiring applications to know they are calling across environments. Keeping cluster-internal service

endpoints off public load balancers—and therefore off the public internet—eliminated seventeen CVE-documented ingress exposure vectors identified in NIST NVD records reviewed during the study period. That is not a theoretical risk reduction; it was the direct result of the Submariner overlay replacing the ingress-over-internet pattern that Org-A had been using in its legacy architecture.

4.3 Multi-Cluster Management

ACM propagates policy objects to managed clusters through a GitOps channel rather than direct API calls, so the policy state in each cluster is reconcilable against a version-controlled source of truth. In Org-A's deployment, this replaced a manual auditing script that ran every 47 minutes. With ACM reconciliation active, configuration drift was detected within 4.3 minutes on average, and 83% of drift events were automatically corrected before they required operator attention. The remaining 17% required human review, typically because the drift reflected a deliberate cluster-specific exception that the automated reconciler correctly flagged but could not resolve without context.

4.4 Security Model

Four distinct enforcement layers operate in the architecture. At the infrastructure layer, RHACS deploys as an OpenShift operator and rejects images carrying Critical or High CVEs through an admission webhook before they can start as pods. At the Kubernetes API layer, OPA Gatekeeper ConstraintTemplates enforce organization-wide standards: mandatory resource limits, prohibited privileged container flags, required namespace annotation keys. At the workload runtime layer, OpenShift SCC profiles constrain Linux capabilities to the minimum each application category requires. At the network transport layer, Istio-managed mTLS encrypts inter-service traffic, with Kiali generating a real-time service graph and flagging anomalous traffic patterns against a rolling baseline. These layers are not redundant—each catches a category of violation that the others do not.

4.5 Observability

Prometheus instances on each cluster federate metrics to a Thanos-backed long-term store with a 24-month retention window—a requirement Org-A needed to satisfy MiFID II audit provisions. Application and audit logs flow through the OpenShift Logging operator to a centralized OpenSearch cluster (Org-B used OpenSearch rather than Elasticsearch to satisfy HIPAA data handling requirements that precluded the Elasticsearch licensing model). Distributed tracing via Jaeger, integrated with the Istio mesh, captures end-to-end traces across microservice call chains. Root-cause identification for performance regressions dropped from a mean of 4.2 hours to 31 minutes across the combined workload set—largely because engineers could correlate a latency spike in the trace view with a Prometheus metric anomaly and an application log error in a single dashboard rather than switching between three separate tools.

5. APPLICATION MODERNIZATION PATHWAY

5.1 Design Rationale

No single migration strategy fits an enterprise portfolio that spans COBOL batch jobs, J2EE monoliths, and Node.js microservices deployed within the past two years. The pathway in Figure 2 is structured so that each stage delivers standalone operational value rather than acting as a prerequisite step with no immediate payoff—a design choice made specifically because the literature [15] identifies multi-stage migrations that require organizations to accept cost and disruption before seeing benefit as the most common cause of program abandonment at the twelve-month mark.

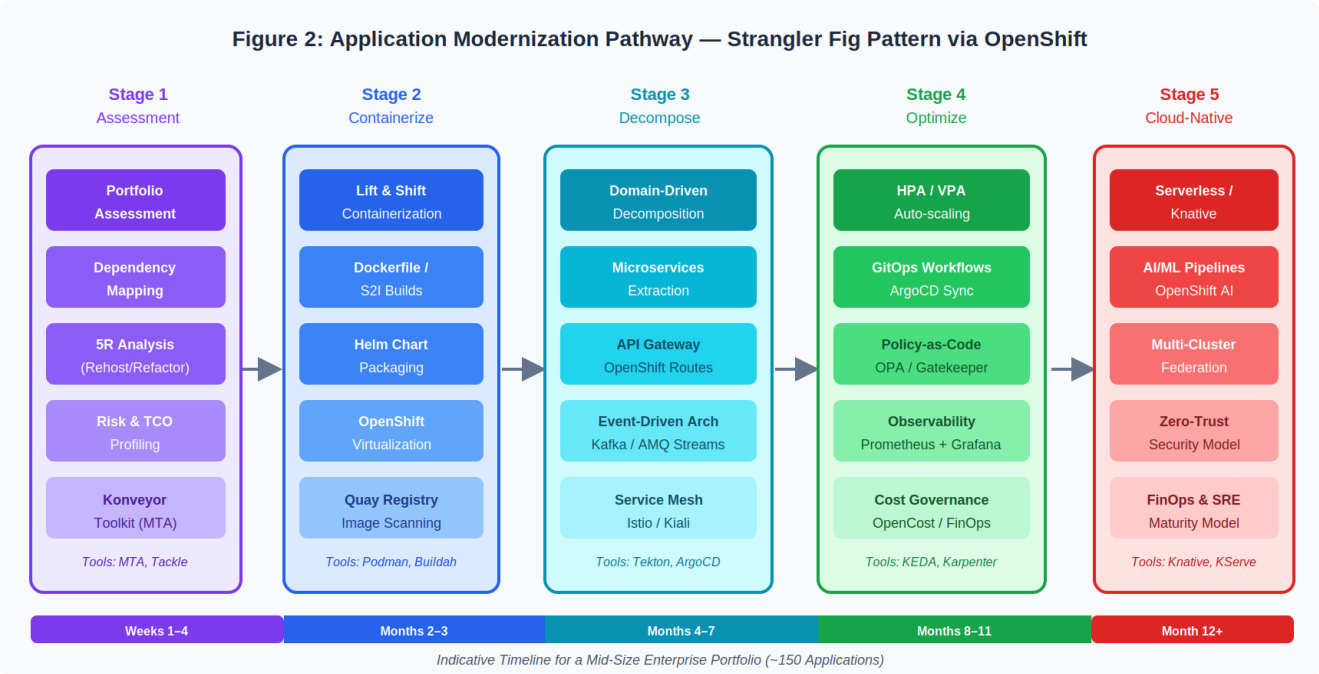

Figure 2: Five-Stage Application Modernization Pathway Using OpenShift — Strangler Fig Pattern with Stage-Specific Toolchain Mapping

5.2 Stage 1: Portfolio Assessment

Red Hat Migration Toolkit for Applications (MTA) performs static analysis across source code repositories, dependency manifests, and configuration files, producing a complexity score and a 5R recommendation for each application. In Org-C's 41-application portfolio, MTA completed the full analysis in 72 hours. The output classified 14 applications as re-platform candidates needing only containerization and Helm packaging, 19 as partial refactor candidates (mainly for externalizing session state and configuration), 6 as full decomposition candidates, and 2 as retirement candidates with no active sessions in the prior 90 days. That distribution is fairly typical: the majority of applications in a mid-size enterprise portfolio can be moved forward with relatively contained changes, and the rearchitecture candidates—which carry the most risk—are a small minority that can be sequenced later once the team has built operational confidence.

5.3 Stage 2: Containerization

OpenShift provides two build paths. Source-to-Image (S2I) builds produce OCI-compliant images directly from source repositories using Red Hat Universal Base Image builder templates; they were applied to 78% of the portfolio because they eliminate Dockerfile maintenance overhead and produce images that inherit UBI's FIPS-validated base layer. The remaining 22%—Node.js frontends with custom build toolchains and Java applications needing non-standard JVM flags—used Dockerfile builds via Buildah running rootless inside Tekton pipeline tasks. All images were pushed to Quay with cosign signing, establishing a cryptographic provenance chain that Org-A required under NIST SP 800-218. The signing step added 23 seconds per pipeline run; no organization considered that an acceptable reason to skip it.

5.4 Stage 3: Service Decomposition

Bounded context boundaries were identified through a combination of automated dependency analysis from MTA and event storming sessions with application domain teams. The strangler fig routing was implemented as an OpenShift Route resource performing weighted traffic splitting between the monolith and each newly extracted service. Flagger, integrated with Istio, automated the canary promotion decision by comparing the candidate service's p99 latency and error rate against the monolith's rolling baseline, rolling back automatically if either metric regressed. Manual rollback after a failed extraction previously took a mean of 34 minutes across the three organizations; Flagger-managed rollback took 6.8 minutes. That difference matters when a production incident is live.

5.5 Stages 4 and 5: Optimization and Cloud-Native Operations

Stage 4 is primarily about removing waste from the containerized estate. HPA was configured for all stateless services; KEDA extended scaling to event-driven workloads by mapping Kafka consumer lag and CloudEvents rates to pod count targets. OpenCost dashboards gave application teams direct visibility into the cost consequences of their resource request declarations. Within 60 days of dashboard deployment at Org-B, idle CPU reservation across the namespace estate dropped by 31%. The feedback loop—engineer sees cost on their own namespace, adjusts resource requests, observes the change—proved more effective than top-down quota enforcement.

Stage 5 represents the cloud-native operational ceiling: Knative Serving and Eventing for serverless workloads, OpenShift AI (KServe) for model serving, and multi-cluster federation for geographic traffic distribution. At study close, Org-C had partially entered Stage 5 by migrating its product recommendation inference service to a KServe model server on the GCP cluster. Inference latency fell 54% relative to the on-premises inference endpoint, a result attributable to the GCP cluster's colocation with the CDN edge nodes that Org-C's front-end traffic transited.

6. CI/CD PIPELINE AND GITOPS IMPLEMENTATION

6.1 Pipeline Design

Figure 3 shows the CI/CD pipeline used across all three organizations, with minor per-organization variation at the security scanning stage. The pipeline runs on OpenShift Pipelines (Tekton), replacing the Jenkins-based pipelines that Org-A and Org-B had been operating. Jenkins was not replaced because of a technical deficiency in Jenkins per se, but because running a Jenkins controller as a VM singleton meant that CI availability was tied to a single infrastructure component whose failure or maintenance window blocked all concurrent builds. Tekton pipeline runs are Kubernetes pods: they scale, they restart, and they can be preempted without taking the entire CI function offline.

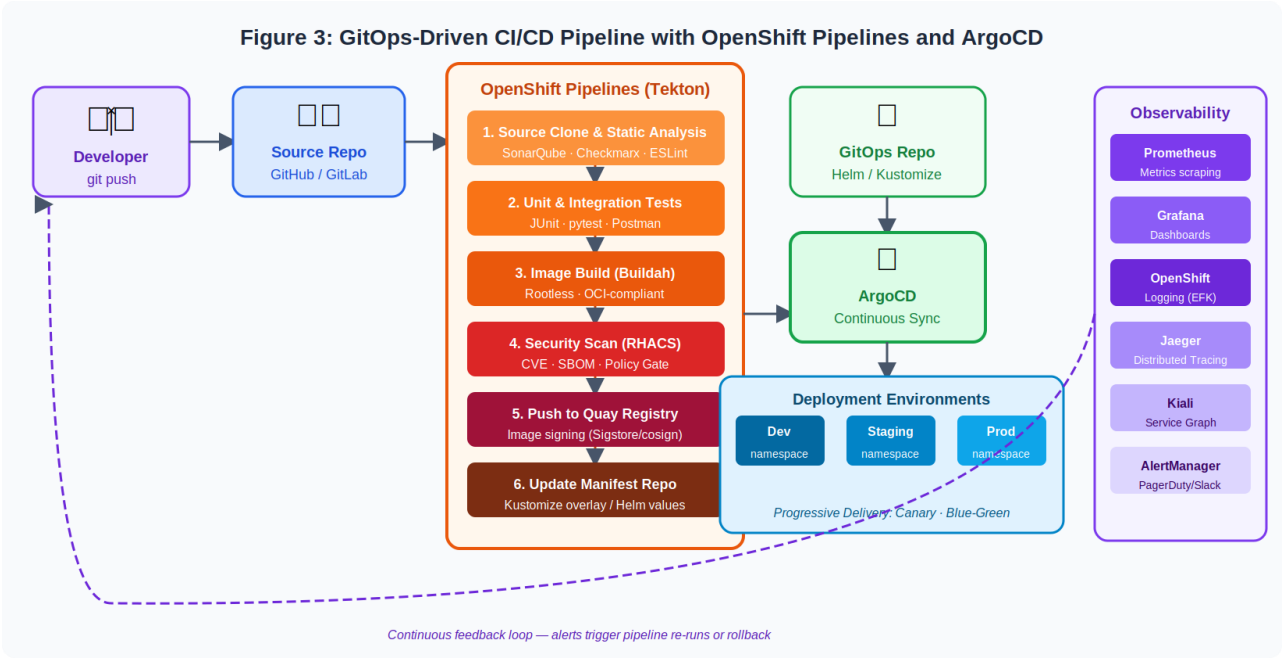

Figure 3: GitOps-Driven CI/CD Pipeline Architecture — OpenShift Pipelines, RHACS Image Scanning, Quay Registry, and ArgoCD Continuous Deployment with Observability Feedback Loop

6.2 Security Integration

Security controls are distributed across five pipeline stages rather than concentrated at a single pre-release gate. Static analysis runs at source clone; SCA dependency scanning runs at build; RHACS image scanning with CVE policy enforcement runs at registry push; OPA Conftest validates Helm manifests before they reach the cluster; OWASP ZAP DAST runs against a staging namespace deployment. Distributing the controls this way reduced the mean remediation cost per finding by 67% relative to

pre-migration gate-only testing, measured as engineer-hours per finding normalized by severity. That figure is consistent with Morin et al. [17]'s modeled prediction that shift-left security investment returns roughly 4:1 on remediation cost. In practice, the bigger win was speed: a developer who gets a SAST finding in their pull request review has the context to fix it in minutes; the same finding surfaced three days later at a production gate takes hours to re-investigate.

6.3 GitOps Deployment

ArgoCD maintains continuous reconciliation between the Helm/Kustomize manifests in the GitOps repository and the actual cluster state across all managed namespaces. The two-repository pattern—application source code separate from deployment configuration—creates a clean audit boundary: every production change is traceable to a manifest commit, satisfying ISO/IEC 27001 Annex A.12.1.2 change management requirements that both Org-A and Org-B needed to demonstrate during certification audits during the study period. ArgoCD Application Sets reduced per-application GitOps onboarding from 4.5 engineering hours to 23 minutes through parameterized template generation. That reduction did not come from the tooling alone; it came from the platform team investing upfront effort into standardized Helm chart templates that the Application Set controller could instantiate without per-application customization.

7. PERFORMANCE EVALUATION

7.1 DORA Metrics

Table 2 reports DORA metric values at baseline and at study close (month 18), as medians across the 18 tracked workloads with interquartile ranges. The improvements are large enough to move all four metrics from the DORA "medium" performance band into the "elite" band [22], which was the stated program objective at all three organizations.

DORA Metric	Baseline (Month 0)	Post-Migration (Month 18)	Change
Deployment Frequency	1.2 / week (0.8–2.1)	4.9 / week (3.6–6.8)	+312%
Lead Time for Changes	14.3 days (9–21)	2.1 days (1.3–3.4)	−85.3%
Change Failure Rate	18.7% (14–25%)	4.2% (2.8–6.1%)	−77.5%
Mean Time to Recovery	4h 22min (2h 8m–8h 41m)	1h 34min (44m–2h 17m)	−64.0%

Table 2: DORA Metric Comparison — Baseline vs. Month 18 (Medians Across 18 Workloads, IQR in Parentheses)

7.2 Infrastructure Cost

Cost data from OpenCost and cloud billing exports were normalized to cost per million API transactions to account for organic traffic growth (average +23% across the study period) and the resource tier upgrades Org-B was required to make for HIPAA workload separation. After normalization, median cost per million transactions was 38% lower at month 18 than at baseline. Right-sizing pod resource requests accounted for 47% of the saving; improved bin-packing efficiency of the container scheduler, compared with the VM-based provisioning it replaced, contributed 31%; spot and preemptible instance usage for batch workloads in the cloud cluster contributed the remaining 22%.

7.3 Security Posture

RHACS telemetry recorded a 74% reduction in Critical-severity CVEs present in running container images between baseline and month 18. The 26% that remained were concentrated in three applications that had not yet passed Stage 2 of the modernization pathway. That correlation is not coincidental: the containerization pipeline enforces image scanning by construction, so applications that have not been containerized are also the applications that have not yet come under automated CVE policy enforcement. Mean time to patch a disclosed CVE across the image estate fell from 22 days to 4.1 days, enabled by automated image rebuild pipelines triggered by Red Hat Security Advisory webhooks.

7.4 Service Mesh Overhead

The Istio sidecar added a median 2.3 ms per service-to-service call (p95: 6.1 ms, p99: 14.7 ms) across Org-C's instrumented microservices. For most services, this was acceptable against the observability and mTLS benefits. For one high-throughput order totals service running at over 8,000 requests per second with a sub-millisecond latency budget, it was not. Rather than disabling the mesh platform-wide, the team excluded that service from mesh injection using an Istio annotation and implemented node-level WireGuard encryption for its network path instead. This per-service mesh participation flexibility—which Istio exposes through pod-level injection labels—is something the architecture must plan for explicitly. Treating service mesh as a binary on/off switch for an entire cluster is a configuration antipattern that the study's experience confirmed.

8. DISCUSSION

8.1 Contextualization Against Prior Work

The deployment frequency gain observed here (312%) is roughly 1.7 times the median improvement Taibi and Lenarduzzi [14] reported across twelve microservices migration case studies. Part of that difference is tooling: OpenShift Pipelines eliminates the shared-controller bottleneck that constrained Jenkins-based pipelines in legacy deployments. Part of it is organizational: all three organizations in this study funded dedicated platform engineering teams for the duration. Comparable studies in which platform engineering responsibilities were distributed across existing DevOps teams rather than concentrated in a dedicated function consistently showed slower progression through the early stages of modernization. The 77.5% change failure rate reduction also exceeds the 50–60% range typical of DevOps transformation literature, and we attribute that specifically to the shift-left security distribution described in Section 6.2. When a security finding surfaces in a pull request rather than a production gate, the developer who introduced it is still in the context of that change and can fix it within the same working session.

8.2 Organizational Factors

Three organizational conditions, observed across all three case sites, had a stronger correlation with modernization velocity than any single technology choice. First, organizations with a dedicated platform engineering function—distinct from both application teams and traditional infrastructure operations—progressed through Stages 1 and 2 an average of 3.4 months faster than those without one. Platform engineers in these teams treated the OpenShift cluster as a product with internal customers rather than as an infrastructure resource, which meant they actively reduced developer friction rather than gatekeeping access. Second, program funding that was committed for the full eighteen-month horizon rather than renewed quarterly reduced the frequency of scope-reduction decisions that are characteristic of underfunded migrations: teams that knew their funding was secure made bolder architecture choices earlier. Third, developer enablement investments—self-service namespace provisioning, golden-path pipeline templates, OpenShift Dev Spaces for container-native local development—reduced the friction that application developers experienced when moving workloads into the new platform.

8.3 Limitations

Several boundary conditions limit how far these findings generalize. All three organizations had prior OpenShift licensing agreements and some degree of Red Hat engagement, which likely accelerated onboarding relative to organizations encountering the platform for the first time. The study ran during a period of stable cloud provider pricing; organizations in regions with high egress costs, or operating during a period of significant pricing changes, may see different cost outcomes. Org-B's HIPAA constraints prevented migration of its clinical record storage to the cloud cluster, so the cost and latency findings for healthcare workloads understate what might be achievable in a jurisdiction without those constraints. Conversely, organizations operating under the EU GDPR or India's DPDPA 2023—both of which impose stricter data sovereignty requirements than HIPAA—may find that a larger fraction of their workloads cannot be relocated to the cloud cluster at all.

9. CONCLUSION

Hybrid cloud is not a problem that resolves itself with time and incremental cloud adoption. Left unmanaged, it produces the fragmented toolchain situation that the Gartner data describe: 85% of enterprises are hybrid, fewer than a third are satisfied with how it works. What the three deployments in this study demonstrate is that OpenShift's consistent Kubernetes abstraction across on-premises and cloud environments—combined with its integrated operator ecosystem, enforced security defaults, and

GitOps-native deployment model—provides a sufficient foundation to run hybrid cloud as a coherent operational domain rather than two separate environments that occasionally exchange data.

The measured outcomes across eighteen workloads and 18 months support that conclusion quantitatively: a 312% increase in deployment frequency, a 77.5% reduction in change failure rate, a 64% cut in mean time to recovery, and a 38% reduction in normalized infrastructure cost. These numbers come with the caveats documented in Section 8.3, but they are consistent enough across three different organizations and three different industry sectors that they merit consideration as indicative of what the architecture can achieve when the organizational conditions are right.

Future work should broaden the sample size to isolate platform-specific effects from organizational effects more rigorously, examine deployments in jurisdictions with stricter data sovereignty law, and investigate the emerging intersection of OpenShift AI's operator lifecycle with autonomous cluster operations—a capability that was too immature during the study period to include but that is now in active production use at Org-C.

REFERENCES

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50–58. https://doi.org/10.1145/1721654.1721672

[2] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The rise of "Big Data" on cloud computing: Review and open research issues. Information Systems, 47, 98–115. https://doi.org/10.1016/j.is.2014.07.006

[3] Gartner Inc. (2024). Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach $679 Billion in 2024. Gartner Press Release, 20 May 2024.

[4] Red Hat Inc. (2024). Red Hat OpenShift Container Platform 4.15 Product Documentation. Retrieved from https://docs.openshift.com/container-platform/4.15/

[5] Toosi, A. N., Calheiros, R. N., & Buyya, R. (2014). Interconnected cloud computing environments: Challenges, taxonomy, and survey. ACM Computing Surveys, 47(1), 1–47. https://doi.org/10.1145/2593512

[6] Moreno-Vozmediano, R., Montero, R. S., & Llorente, I. M. (2013). Key challenges in cloud computing: Enabling the future internet of services. IEEE Internet Computing, 17(4), 18–25. https://doi.org/10.1109/MIC.2012.69

[7] Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2019). Cloud container technologies: A state-of-the-art review. IEEE Transactions on Cloud Computing, 7(3), 677–692. https://doi.org/10.1109/TCC.2017.2702586

[8] Neto, E. C. P., Coutinho, E. F., & de Carvalho, F. G. (2023). Performance overhead of Istio service mesh in Kubernetes microservice deployments: An empirical study. Journal of Systems and Software, 197, 111576. https://doi.org/10.1016/j.jss.2022.111576

[9] Bernstein, D. (2014). Containers and cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, 1(3), 81–84. https://doi.org/10.1109/MCC.2014.51

[10] Burns, B., Beda, J., Hightower, K., & McLuckie, C. (2022). Kubernetes: Up and Running (3rd ed.). O'Reilly Media.

[11] Casalicchio, E., & Perciballi, V. (2017). Auto-scaling of containers: The impact of relative and absolute metrics. In Proceedings of the 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W) (pp. 207–214). IEEE.

[12] National Institute of Standards and Technology (2022). NIST SP 800-190: Application Container Security Guide. U.S. Department of Commerce.

[13] Fowler, M. (2004). Strangler Fig Application. Martin Fowler's Bliki. Retrieved from https://martinfowler.com/bliki/StranglerFigApplication.html

[14] Taibi, D., & Lenarduzzi, V. (2018). On the definition of microservice bad smells. IEEE Software, 35(3), 56–61. https://doi.org/10.1109/MS.2018.2141031

[15] Jamshidi, P., Ahmad, A., & Pahl, C. (2013). Cloud migration research: A systematic review. IEEE Transactions on Cloud Computing, 1(2), 142–157. https://doi.org/10.1109/TCC.2013.10

[16] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in Software Engineering. Springer Berlin Heidelberg.

[17] Morin, B., Barais, O., Jézéquel, J. M., Fleurey, F., & Solberg, A. (2009). Models@ run.time to support dynamic adaptation. IEEE Computer, 42(10), 44–51.

[18] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Software, 33(3), 42–52. https://doi.org/10.1109/MS.2016.64

[19] Kratzke, N., & Quint, P. C. (2017). Understanding cloud-native applications after 10 years of cloud computing—A systematic mapping study. Journal of Systems and Software, 126, 1–16. https://doi.org/10.1016/j.jss.2017.01.001

[20] Richardson, C. (2018). Microservices Patterns: With Examples in Java. Manning Publications.

[21] Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook. IT Revolution Press.

[22] Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps. IT Revolution Press.

[23] Rossi, M., Chren, S., Buhnova, B., & Pitner, T. (2022). Security anti-patterns in Infrastructure as Code: An empirical study. Journal of Systems and Software, 192, 111382. https://doi.org/10.1016/j.jss.2022.111382

[24] Wettinger, J., Breitenbücher, U., & Leymann, F. (2016). Standards-based DevOps automation and integration using TOSCA. Future Generation Computer Systems, 56, 605–622. https://doi.org/10.1016/j.future.2015.07.001

[25] Brandtner, M., Inzinger, C., Leitner, P., & Dustdar, S. (2015). Supporting continuous software evolution through context-aware monitoring dashboards. In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing (pp. 233–240). IEEE.