Modernizing GxP-Validated Systems with Microservices: A Compliance-Aware Framework for Performance, Scalability, and Continuous Validation

Rajashekar Reddy Gujjula
Senior Systems Engineer, Cloud and Data Center Technology
Fresenius Kabi USA, Lake Zurich, Illinois, USA
rajashekarreddy.gujjula@gmail.com

To Cite this Article
Rajashekar Reddy Gujjula, "Modernizing GxP-Validated Systems with Microservices: A Compliance-Aware Framework for Performance, Scalability, and Continuous Validation", Journal of Science Engineering Technology and Management Science, Vol. 01, Issue 01, October 2024, pp: 348-363, DOI: http://doi.org/10.64771/jsetms.2024.v01.i01.pp348-363
Submitted: 01-09-2026 *Accepted: 04-10-2026* *Published: 10-10-2026*

Abstract
Computerized systems that facilitate GxP activities in the pharmaceutical and life sciences industry, such as laboratory information management, batch records, and quality management systems, are often implemented as validated monoliths. Maintaining the validated status of these systems is expensive, which results in large infrequent releases, static capacity provisioning based on peak loads, and audit trails relying on database permissions rather than integrity guarantees. However, migration of these systems into microservices allows for elasticity and faster deployment, although it leads to fragmenting of the audit trail, expanded validation scope, and autoscaling behavior that is hard to justify to regulatory authorities. In this paper, we propose the Compliance Aware Microservice Framework (CAMF) that brings together four techniques: (i) transactional-outbox, hash-chained, and periodically anchored audit log that is used to generate tamper evidence that is consistent with ALCOA+ and 21 CFR Part 11; (ii) compliance-aware autoscaler that uses Holt forecast along with criticality-based utilization targets, replica floors, and image digest-based scaling; (iii) risk-based change impact analysis that is used to bound re-validation scope using dependency graph and consumer-driven contract tests; and (iv) validation-as-code pipeline. The CAMF approach was compared with the VM, the LSM approach and the MSA approach with respect to a queueing network simulation of a two-shift laboratory workload, tamper injection test on 20,000-record audit log, and a validation effort model on scale free dependency graph, each of which was run with 30 different random seeds. With baseline load, CAMF cut down the percentage of the time that violates the service level objective of p95 < 300 ms from 0.33% (MSA) to 0.04% and the request error rate from 0.46% to 0.03%, while being 30.1% more costly on a per-unit basis than MSA but 42.5% less costly than VM. Under eight times baseline load, the failure rate of the monolith was 53.4% while that of CAMF was 0.78%. All tampering activities could be detected by the hash-chained ledger, while only 51.6% of tampering could be detected by sequence checked relational audit tables, and the risk based validation scope helped reduce the re-validation effort by 96.5% than the monolith per release.

Keywords: GxP compliance; Microservices; Computerized system validation; Data integrity; Audit trail; Autoscaling; 21 CFR Part 11; Cloud-native architecture

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/
CC BY-NC-ND 4.0

1. Introduction

Pharmaceutical, biotech, and medical device companies rely on computer systems for generation, processing, and storage of the records that serve as the basis of decisions about product quality and patient safety. LIMS, electronic batch records, and eQMSs are GxP systems in that they are used for GxP-related activities, where GxP stands for good manufacturing practice, good laboratory practice, good clinical practice, or good distribution practice. They should be validated for their intended use and maintain data that are attributable, legible, contemporaneous, original, and accurate, along with the extended ALCOA+ characteristics of completeness, consistency, durability, and availability (Medicines and Healthcare products Regulatory Agency [MHRA], 2018; U.S. Food and Drug Administration [FDA], 2018; World Health Organization [WHO], 2016). The requirements were compiled into one inspectorate guide by the Pharmaceutical Inspection Co-operation Scheme (Pharmaceutical Inspection Co-operation

Scheme [PIC/S], 2021), and later extended to computer systems employed in clinical trials by the European Medicines Agency (European Medicines Agency [EMA], 2023).

Most of the GxP platforms currently available in use have been designed as monolithic platforms - one deployable unit relying on one common relational database that is validated together and modified by infrequent and heavily documented changes. Such an approach was convenient at first as far as the scale of the systems was relatively small, but it contradicts the current digital transformation strategy of the industry. Industry 4.0 requires the continuous flow of data from instruments and PAT into the decision-making concerning quality (Arden et al., 2021; Rantanen & Khinast, 2015), and the number of regulated data to be handled grows fast (Alosert et al., 2022; Hole et al., 2021). Monolithic system configured for peak load is costly in low load periods and still overloaded during batch release bursts, whereas the full regression for any modification leads to month-long lead times.

Applications based on microservice architectures consist of independently deployable services, which communicate with each other through lightweight interfaces, and are individually scalable (Dragoni et al., 2017; Jamshidi et al., 2018). Practical experience gained by industrial players in different industries, such as banking, demonstrates that such a transition could help to improve scalability and maintainability of critical tasks (Bucchiarone et al., 2018; Mazzara et al., 2021). Migration is not a cost-free process, though: industry experts face problems associated with data consistency, distributed debugging, and increased complexity (Fritzsch et al., 2019; Soldani et al., 2018; Taibi et al., 2017). In the context of GxP these problems have a legal aspect. For example, a transaction that writes a business object to a service and an audit object to another one might lose the audit object, the automatic launch of an unverified image leads to an unverified system state, and the dependency graph of tens of services makes it difficult to predict the effects of a change.

Regulatory mindsets have started to embrace such developments. The second version of GAMP 5 has explicitly incorporated agile methods, cloud computing, and critical thinking in the context of validation (International Society for Pharmaceutical Engineering [ISPE], 2022), while FDA draft guidance on computer software assurance has supported assurance activities based on risk assessments and automated records of tests as objective evidence (FDA, 2022). There has been research on the implementation of DevOps in regulated environments and has found traceability, documentation, and release approval as the key challenges in implementing continuous delivery (Laukkarinen et al., 2017, 2018), while activities of compliance can be integrated into delivery pipeline (Moyón et al., 2020). What is still lacking is an architectural framework which would consider the GxP controls as a first-order design element of a microservice architecture and evaluate their impact on performance, scalability, cost, data integrity, and validation efforts altogether.

This study addresses that gap. Its contributions are as follows:

- Layered CAMF that translates compliance needs for audit trail, electronic signatures, validated state, and change management into specific architectural elements (Figure 1 and Table 2).
- A tamper-proof audit ledger built by linking a transactional outbox pattern with SHA-256 hashing and periodic anchoring, thereby resolving the dilemma between synchronous write audits and complete audit coverage.
- A compliance-aware autoscaling algorithm based on Holt forecasting, criticality-based utilization targets, and replica floors, which scales up only qualified image digests.
- A risk-driven change impact model that limits the scope of re-validation in the service dependency graph and a validation-as-code pipeline which generates qualification proofs automatically.
- A rigorous empirical evaluation that includes 30 seeds per scenario and analyzes latency, SLO compliance, error rate, unit cost, scalability up to eight times baseline load, audit store outage, tamper proofing, and validation effort using ANOVA and Tukey tests.

The rest of the paper is structured as follows. Section 2 provides an overview of relevant literature. The framework and its formal model together with the experiment are presented in Section 3. Section 4 describes the findings, Section 5 provides the discussion, and Section 6 draws conclusions.

2. Related Work

2.1. Microservice architectures and migration

Systematic mapping has traced the microservices literature from its initial stage of conception to industrial application (Di Francesco et al., 2019; Pahl & Jamshidi, 2016) and has placed the use of microservices into the broader shift toward cloud-native architecture (Kratzke & Quint, 2017). Balalaie et al. (2016) demonstrated that microservices and DevOps systems mutually support one another during transition, and Waseem et al. (2020) conducted systematic mapping of the research on the topic. Studies comparing monolithic architectures and

microservice architecture indicate that microservices could reduce cost and increase scalability but require extra hops and complexity in comparison with monoliths (Al-Debagy & Martinek, 2018; Villamizar et al., 2015). Workload characterization has established that microservice performance depends on the level of overhead and runtime configuration (Ueda et al., 2016), while container virtualization incurs very small overhead in comparison with VMs (Felter et al., 2015).

Many ways have been suggested for the decomposition process. For example, Service Cutter identifies the service boundaries based on coupling requirements (Gysel et al., 2016). Furthermore, candidate microservices can be derived from version control history and code structure (Mazlami et al., 2017). Data management is one of the most challenging aspects of a migration process: Laigner et al. (2021) showed that practitioners face the challenge of inter-service consistency and often use event-driven patterns, while Overeem et al. (2021) described the concept of event sourcing in the context of industrial applications, especially with respect to issues related to schema evolution due to an append-only event log becoming the system of record.

2.2. Performance, autoscaling, and benchmarks

Reproducibility of microservices performance studies has been made possible by open benchmarks and traces. DeathStarBench offers end-to-end systems for investigating tail latencies and clustering (Gan et al., 2019), while TrainTicket offers a benchmark for analyzing faults (Zhou et al., 2021). Analysis of Alibaba production traces has found that the system had heavy tailed and tree-shaped call graphs with hot-spot microservices (Luo et al., 2021). The 2019 Google cluster trace describes scheduling in a large shared cluster (Tirmazi et al., 2020). Kubernetes is the most popular orchestrator, following the design principles of Borg and Omega (Burns et al., 2016).

Autoscaling techniques have been thoroughly studied (Qu et al., 2018). The Kubernetes Horizontal Pod Autoscaler takes into account the utilization observed and may fail to cope with bursts in demand (Nguyen et al., 2020); multilevel decision-making with infrastructure and application performance indicators yields better results (Taherizadeh & Stankovski, 2019); Autopilot utilizes history of recommendations to cut down the overhead at Google scale (Rzadca et al., 2020); and FIRM is an algorithm based on machine learning for dealing with SLO violations in microservices' graphs (Qiu et al., 2020). However, none of these methods account for regulatory restrictions like the necessity to keep each deployed instance qualified or to protect the capacity for critical processes.

2.3. Data integrity, audit trails, and regulated delivery

Tamper-evident logging has also been considered using blockchain and hash-chain techniques. Sutton and Samavi (2017) logged integrity proofs for privacy audit logs using the blockchain technology, Steinwandter and Herwig (2019) combined versioning with Ethereum smart contract to prove the pharmaceutical data is not altered, while Leal et al. (2021) used blockchain and data-quality agents to guarantee end-to-end traceability in drug manufacturing. The aforementioned research confirms cryptographic chaining could be used to produce audit evidence. However, the ledger is considered to be another system, whereas consistency of logging in the distributed environment by distributed services is not covered. Security mapping research of microservices concentrates on authentication, authorization, and secure communication rather than record integrity (Hannousse & Yahiouche, 2021).

As for the process, CI/CD practices have been well researched (Shahin et al., 2017), and its adoption for regulated software has been studied for medical device (Laukkarinen et al., 2017, 2018) and industrial control system (Moyón et al., 2020). While those papers prove compliance evidence could be produced within CI/CD pipelines, they do not address validation scope. Table 1 contains the most representative publications and gaps that inspire the development of CAMF.

Table 1: Literature matrix of representative studies relevant to GxP microservice modernization.

Study	Technique / focus	Contribution	Limitation for GxP systems
Balalaie et al. (2016)	Migration with DevOps	Incremental migration patterns	No regulatory or audit considerations
Mazzara et al. (2021)	Mission-critical migration	Scalability gains in banking FX core	Finance domain; no validation model
Gan et al. (2019)	DeathStarBench	Tail-latency and cluster effects	Benchmark only; no compliance controls
Luo et al. (2021)	Alibaba trace analysis	Heavy-tailed call graphs, hot spots	Characterization; no design guidance
Nguyen et al. (2020)	Kubernetes HPA study	Metric choice and scaling lag	Reactive; ignores qualified state
Qiu et al. (2020)	FIRM (ML resource mgmt.)	Lower SLO violations	Model training; hard to validate

Study	Technique / focus	Contribution	Limitation for GxP systems
Laigner et al. (2021)	Data management survey	Consistency practice and gaps	No audit-trail integrity analysis
Sutton & Samavi (2017)	Blockchain audit logs	Tamper-proof privacy logs	External chain; no service integration
Steinwandter & Herwig (2019)	VCS + smart contract	Provable pharma data integrity	Batch data; no runtime architecture
Leal et al. (2021)	Blockchain + agents	End-to-end traceability	Manufacturing data; no scaling model
Laukkarinen et al. (2018)	Regulated DevOps	Mapping DevOps to standards	Process view; no architecture or metrics
Moyón et al. (2020)	Security standards in CI/CD	Compliance activities in pipelines	Security focus; no validation scope model
This study (CAMF)	Compliance-aware microservices	Integrated audit, scaling, validation	Simulation-based evaluation

3. Methods

3.1. Framework overview

The CAMF architecture consists of four layers (Figure 1). Layer L1 manages access control – API gateway with mutual TLS and rate limiting, identity service providing role-based access control, and electronic signature service, which ties the signature to the signer, timestamp, and meaning of the signature in accordance with 21 CFR 11.50 and 11.200. The GxP domain services are contained in Layer L2. In the case of the laboratory and quality system under evaluation, they are sample management, results and calculations, batch record, deviation and CAPA, document control, reporting (optimized for reading), and instrumentation integration services.

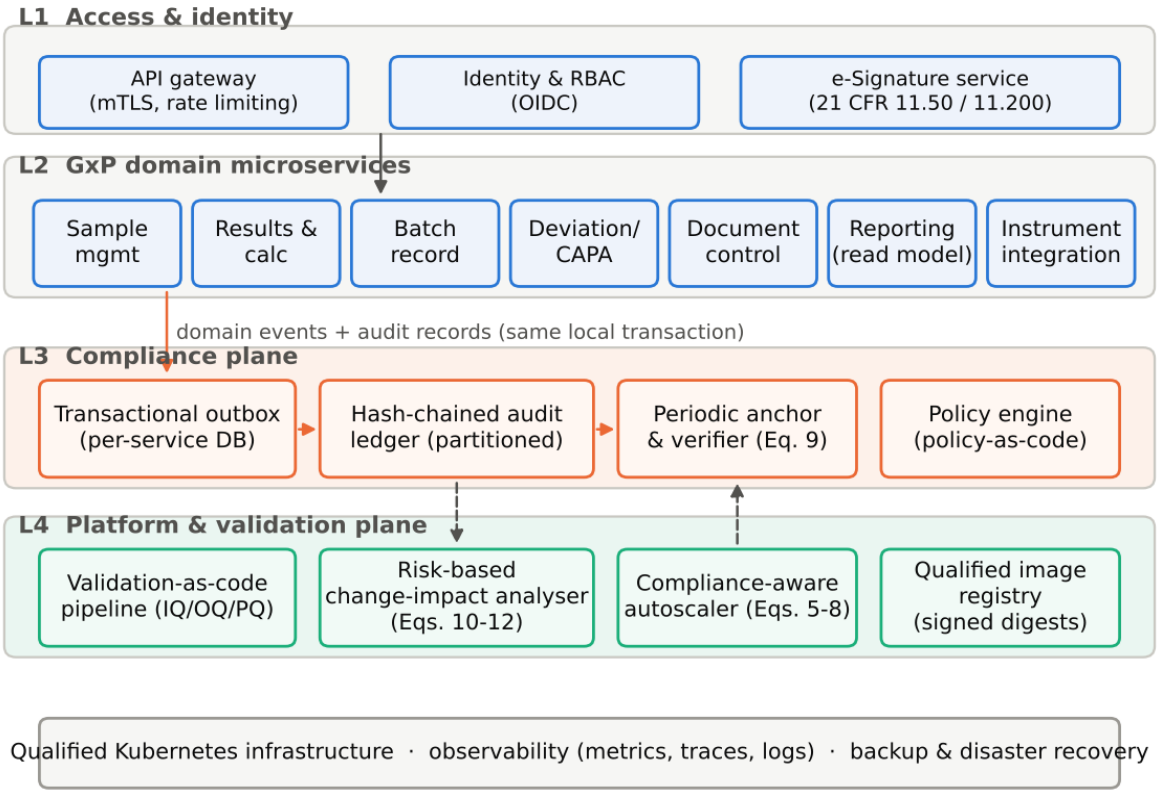

Figure 1: Layered architecture of the proposed Compliance-Aware Microservice Framework (CAMF).

Layer L3, the compliance plane, is where we diverge most from typical microservice architectures. The domain service will save both the business event and its audit log entry in a local outbox table in a single transaction, so that any business event cannot be committed unless its audit entry is committed as well. The relay will publish outbox messages into a distributed, append-only audit log, where each message is linked to the previous one by SHA-256, and a validator will periodically revalidate the chain using anchors in a write-once storage. A policy engine will apply release and runtime policies, like ensuring that only signed and qualified image digests can be used in GxP

namespaces, as executable policies. Layer L4, the platform and validation plane, consists of the validation-as-code pipeline, the risk-based change impact analysis tool, the compliance-aware autoscaler and the qualified image registry running on qualified Kubernetes. Table 2 shows how the key regulatory requirements map onto the CAMF mechanisms.

Table 2: Mapping of regulatory expectations to CAMF architectural controls.

Requirement (source)	Expectation	CAMF mechanism
Audit trail (21 CFR 11.10(e); PIC/S, 2021)	Secure, computer-generated, time-stamped record of create/modify/delete	Transactional outbox + hash-chained ledger (Eq. 9)
Data integrity, ALCOA+ (MHRA, 2018; FDA, 2018; WHO, 2016)	Complete, original, enduring, available records	Same-transaction audit write; anchored verification; partition replication
Electronic signatures (21 CFR 11.50, 11.200)	Signature linked to record, signer, time, and meaning	Dedicated e-signature service; signature manifest in ledger
Validated state (ISPE, 2022)	System remains in a state of control through change	Qualified image registry; autoscaler restricted to qualified digests
Risk-based assurance (ISPE, 2022; FDA, 2022)	Assurance effort proportionate to risk	Change-impact analyser with GxP criticality (Eqs. 10-12)
Computerised systems in trials (EMA, 2023)	Validation evidence, audit-trail review, access control	Validation-as-code evidence packages; RBAC; periodic ledger review

3.2. Performance model

Service s is represented by the M/M/c queue with cs(t) = m * ns(t) servers, where ns(t) is the number of replicas at time t and m=4 is the number of servers per replica. The end-to-end latency of request of class r is calculated as the total of the residence time at the stations in its call path Pr plus network overhead δ per hop, as given in Equation (1):

$$T_r = \sum_{s \in P_r} (W_s + S_s + \delta) \tag{1}$$

where W_s is the queueing delay and S_s is the service time, drawn from an exponential distribution with mean $1/\mu_s$. The probability that a request must wait is given by the Erlang-C formula in Equation (2), with offered load $a = \lambda_s/\mu_s$:

$$C(c,a) = \frac{\frac{a^c}{c!}\frac{c}{c-a}}{\sum_{k=0}^{c-1}\frac{a^k}{k!} + \frac{a^c}{c!}\frac{c}{c-a}} \tag{2}$$

and the conditional waiting time is exponential, so that the tail of the waiting-time distribution follows Equation (3):

$$P(W_s > t) = C(c,a)e^{-(c\mu_s - \lambda_s)t} \tag{3}$$

When λ_s exceeds capacity, the excess accumulates as a backlog that adds a deterministic queueing delay; requests whose queueing delay would exceed a 5 s client timeout are shed and counted as errors. The provisioning objective is to minimize resource cost while meeting the latency objective and respecting GxP replica floors, as formulated in Equation (4):

$$min \ \sum_t \sum_s v_s n_s(t) \ \ s.t. \ p95_t \le L_{SLO}, \ n_s(t) \ge n_{min}(g_s) \tag{4}$$

where v_s is the number of vCPUs per replica, $g_s \in \{1, 2, 3\}$ is the GxP criticality of service s (low, medium, high), and L_{SLO} = 300 ms is the p95 objective for interactive requests.

3.3. Compliance-aware autoscaling

Reactive autoscalers act on the most recent observation and typically restrict scale-up frequency, so a burst is served by too few replicas for several control intervals (Nguyen et al., 2020). CAMF forecasts the arrival rate of each service with Holt's linear exponential smoothing, whose level and trend updates are given in Equations (5) and (6):

$$\ell_t = \alpha\lambda_{t-1} + (1-\alpha)(\ell_{t-1} + b_{t-1}) \tag{5}$$

$$b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \tag{6}$$

with α = 0.5 and β = 0.3. The desired replica count combines the forecast with the last observation and a criticality-dependent utilisation target $u^*(g_s)$, as given in Equation (7):

$$n_s^*(t) = max\left(n_{min}(g_s), \left\lceil \frac{max(\lambda_{t-1}, \ell_t + hb_t)}{m\mu_s \, u^*(g_s)} \right\rceil\right) \tag{7}$$

where h = 2 is the forecast horizon in minutes, $u^*(3) = 0.60$, $u^*(2) = 0.65$, and $u^*(1) = 0.75$, and $n_{min} = 3$ for high-criticality services and 2 otherwise. Scale-up is applied immediately because new replicas are started from pre-

pulled, pre-qualified image digests; scale-down is damped by retaining the maximum recommendation over a window of $w = 8$ minutes, as expressed in Equation (8):

$$n_s(t) = max\big(n_s(t-1), n_s^*(t)\big)\ if\ up;\ \ max_{\tau\in[t-w,t]}\, n_s^*(\tau)\ otherwise \tag{8}$$

The qualified-digest restriction matters for compliance as well as performance. Each replica started by the autoscaler is an instance of a configuration item whose installation and operational qualification evidence already exists, so scaling does not create an unvalidated state. The policy engine rejects any pod in a GxP namespace whose image digest is not present in the qualified registry.

3.4. Tamper-evident audit ledger

Let r_i be the canonical serialization of the i-th audit record of a ledger partition, containing the record identifier, user, action, object, value before and after the change, reason for change, and a trusted timestamp. The ledger stores the chain value defined in Equation (9):

$$h_i = SHA-256(h_{i-1} \parallel r_i),\ \ h_0 = 0^{256},\ \ A_k = h_{kB} \tag{9}$$

where $\parallel$ stands for concatenation and Ak represents an anchor written to write once storage after every B = 1,000 records. Since any modification, deletion, insertion or reordering of a record will affect all successive values in the chain, any validator which computes the value for a segment using the anchor Ak−1 and then compares the computed value to Ak can detect the change and pinpoint its source to a single segment. Since the write operation to the outbox takes place in the same local transaction as the business change, an audit store failure only delays publication without losing any records.

3.5. Risk-based change impact and validation effort

The service landscape is represented as a directed dependency graph $G = (S, E)$, in which an edge (c, s) indicates that consumer c calls provider s. Let $C(s)$ be the direct consumers of s, $C^*(\Delta)$ the transitive closure of consumers of a change set Δ, and N_s the number of qualification test scripts of service s. CAMF bounds the test scope of a change set using the criticality weights φ and ψ and a platform smoke-test fraction ε, as defined in Equation (10):

$$S_{CAMF}(\Delta) = \sum_{s\in\Delta} \varphi(g_s) N_s + \sum_{s\in\Delta} \sum_{c\in C(s)} \psi(g_c) N_c + \varepsilon \sum_{s\in S} N_s \tag{10}$$

with $\varphi(1, 2, 3) = (0.60, 0.85, 1.00)$, $\psi(g) = 0.25$ for $g < 3$ and 0.50 for $g = 3$ (consumer-driven contract tests rather than full suites), and $\varepsilon = 0.03$. Microservice baselines without contract testing must re-test every transitive consumer in full, as expressed in Equation (11), and the monolith must re-test the entire suite ΣN_s:

$$S_{trans}(\Delta) = \sum_{s\in\Delta\cup C^*(\Delta)} N_s \tag{11}$$

Validation effort per release, in person-hours, comprises a fixed documentation component E_{fix} (validation plan, summary report, quality approval) and an execution component that depends on the automated fraction a, as shown in Equation (12):

$$E(\Delta) = E_{fix} + S(\Delta)[a\, e_{auto} + (1-a) e_{man}] \tag{12}$$

where $e_{man} = 0.75$ h is the effort to execute and review a manual scripted test and $e_{auto} = 0.08$ h is the effort to review automatically generated evidence. The values of E_{fix} and a used for each architecture are listed in Table 5. The procedure that integrates these mechanisms is summarized in Algorithm 1 (Table 6) and the corresponding workflow is depicted in Figure 2.

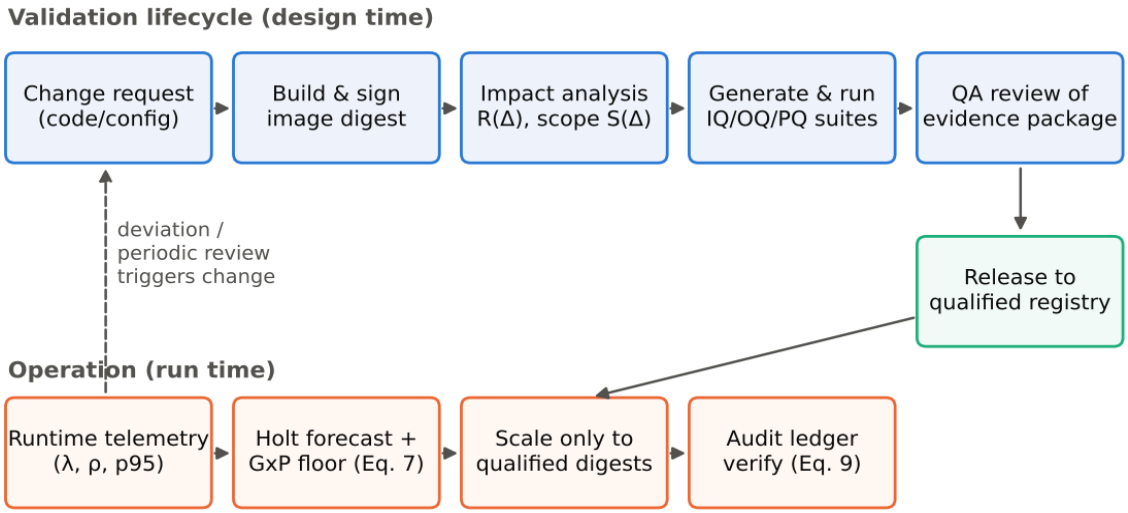

Figure 2: Design-time validation lifecycle and run-time operation workflow of CAMF.

3.6. Experimental design

Workload. There being no publicly available trace of a laboratory that operates with a regulated lab or quality system, the workload had to be constructed based on features of production microservices and tuned to a two-shift laboratory schedule. The arrival rate exhibits a peak during the day shift at 11:00 and another peak, less pronounced, for the evening shift, at 19:30 in the nighttime, having a mean value of 260 requests per second at load multiplier 1. Each simulated day has a Poisson number of bursts (mean 9), lasting 4-15 minutes and multiplying the arrival rate 1.6-2.6 times to reflect the batch-release and shift-end spikes, with minute-scale lognormal noise ($\sigma = 0.08$) superimposed. Such bursty arrivals and hot-spots are consistent with the Alibaba and Google traces (Luo et al., 2021; Tirmazi et al., 2020). Figure 3(a) illustrates an example of the workload.

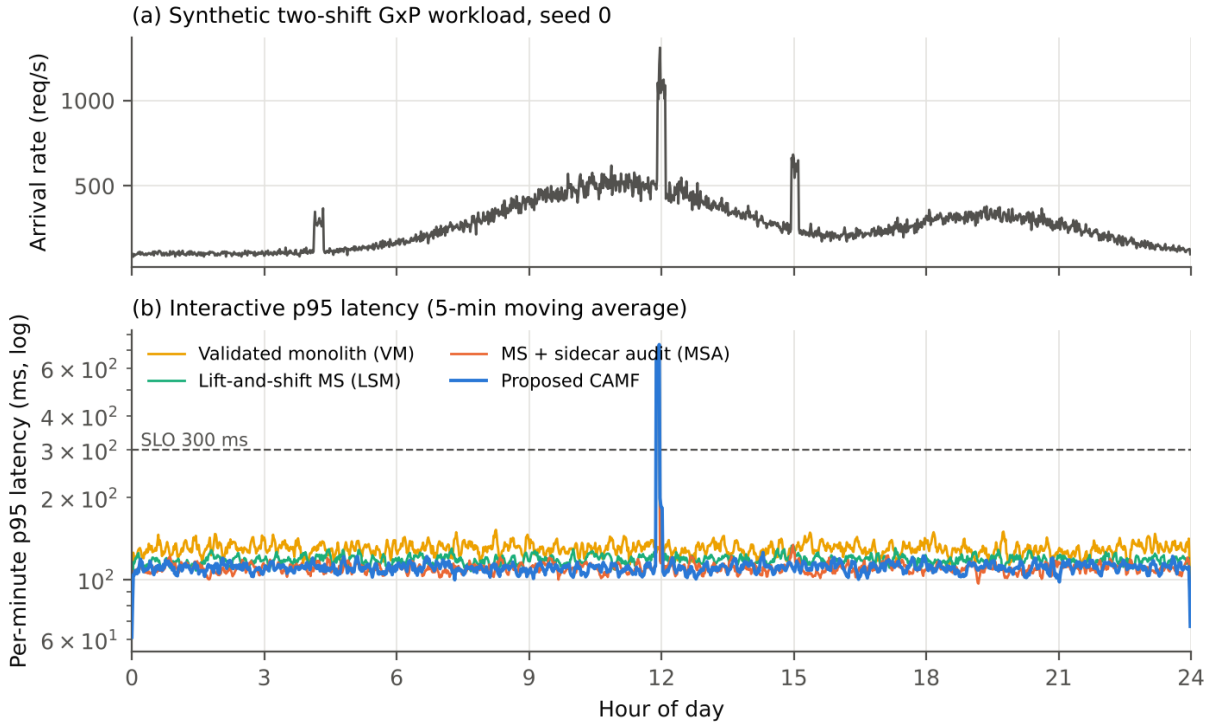

Figure 3: (a) Arrival rate of one workload realization; (b) per-minute interactive p95 latency of each architecture on the same day.

Services and Requests. Table 3 shows the thirteen services, their respective average service times, vCPU utilization and GxP criticality level, and Table 4 shows the six request types with their traffic fractions and calling pattern. The

magnitude of service time is of the same order as that of microservices benchmarking results (Gan et al., 2019; Ueda et al., 2016).

Table 3: Service catalogue of the simulated GxP laboratory and quality system.

Service	Mean service time (ms)	vCPU per replica	GxP criticality
API gateway	2.0	1	Medium
Identity / RBAC	4.0	1	High
e-Signature	9.0	1	High
Sample management	12.0	2	High
Results & calculation	15.0	2	High
Calculation engine	18.0	2	High
Batch record	22.0	2	High
Document control	14.0	2	Medium
Deviation / CAPA	16.0	2	High
Reporting (read model)	35.0	2	Low
Notification	5.0	1	Low
Instrument integration	10.0	2	Medium
Audit ledger / outbox	6.0	1	High

Table 4: Request types, traffic shares, and call paths.

Request type	Share	Interactive	Call path
Result entry with e-signature	26%	Yes	gateway → identity → e-sign → results → calc → audit
Sample login	20%	Yes	gateway → identity → sample → audit
Batch record review	10%	Yes	gateway → identity → e-sign → batch → results → document → audit
Report query	14%	Yes	gateway → identity → reporting → results
Deviation record	6%	Yes	gateway → identity → deviation → notify → audit
Instrument data ingest	24%	No	gateway → integration → results → audit

Architectures. Four architectures have been benchmarked in the same way under equal workloads and seeds, as shown in Table 5. The validated monolith (VM) is deployed on six dedicated nodes that have 16 vCPUs each; in-process calls do not incur a network hop, and therefore, total work amounts to 85% of the sum of the service times, but all requests use the same pool of workers and a relational database with increasing lock contention with utilisation. The lift-and-shift microservices deployment (LSM) employs a reactive autoscaler similar to the Kubernetes Horizontal Pod Autoscaler (targeting 70% utilisation, scale up after every third time interval, five minutes of scale down stabilisation), and it writes audit records synchronously to a centralised audit database with twelve writer connections. The microservices deployment with sidecar audit (MSA) employs the same autoscaler but writes audit records asynchronously using a sidecar with in-memory buffer and no chaining. CAMF leverages the techniques introduced in Sections 3.3-3.5. Newly spun up replicas experience a cold-start penalty (average 2.5 s for images pulled on demand, 0.6 s for CAMF pre-pulled digests) for a portion of requests during the first minute of execution.

Table 5: Simulation parameters and architecture configurations.

Parameter	VM	LSM	MSA	CAMF
Capacity model	Fixed 96 vCPU	Reactive HPA	Reactive HPA	Holt + GxP floors
Utilisation target	-	0.70	0.70	0.60 / 0.65 / 0.75
Minimum replicas	-	2	2	3 (high) / 2
Audit path	Same DB table	Sync. central DB	Async. sidecar	Outbox + hash chain
Cold-start mean (ms)	-	2,500	2,500	600
Automated test fraction a	0.20	0.55	0.70	0.85
Fixed validation effort E_{fix} (h)	40	24	24	12
Test scope per change	Full suite	Eq. (11)	Eq. (11)	Eq. (10)

Experimentation. The experiments simulate 24 hours of execution at 1 minute control intervals for 40 requests of each request type per minute, where the cost incurred is USD 0.048 per vCPU-hour. Five experiments were carried out. (E1) Performance under base conditions of load multiplier 1 with 30 seeds. (E2) Scalability under load multipliers 1, 2, 4, 6, and 8 with 10 seeds each. (E3) Failure of audit-store for 15 minutes during the day-shift peak period (10:00-10:15) with 30 seeds; LSM does not process GxP transactions if the audit-write fails, and MSA loses 3.1% of the buffered records because of pod crashes, the former assumption representing in-memory buffers without any persistent storage. (E4) Tampering-injection experiment in which 40 tampering incidents (modifications, deletions, back-date insertions, and re-ordering) were performed on a 20,000 records audit log for each of 30 seeds, and the following were measured: Detection by Hash Chain Verification (CAMF), detection by Sequence Gap and Duplicate Key checks plus a periodic manual verification at 5% (VM and LSM), and the wall clock time to construct the chain plus verification (MSA). (E5) An experiment for validating an effort on scale-free dependency graphs of 60 services created via preferential attachment that replicates the heavy-tailed fan-in effect found in production call graphs (Luo et al., 2021) in which there are 24 releases per seed in which 1-5 services get changed, and the test suites consisted of Poisson(45) + 10 scripts per service.

Metrics & Statistics. Metrics used included p50, p95, and p99 latency of interactive requests, the percentage of minutes where the interactive p95 is above 300ms (minutes with SLO-violation); the error rate (requests that got shed, rejected or timed-out); throughput; unit cost in USD per million served requests; audit record loss; tampering detection rate; validation person-hours per release; and the change lead time in days. Results are presented as mean ± standard deviation for seeds, and 95% confidence interval in the figures. The differences between architectures were tested using one-way ANOVA followed by Tukey's honest significance difference test at $\alpha = 0.05$. All the models were coded in Python 3 with NumPy and SciPy; for the chain experiment, SHA-256 from the standard library is used.

Table 6: Algorithm 1: Compliance-aware operation and change control in CAMF.

Step	Procedure
Input	Service set S, dependency graph G, criticality g_s, qualified registry Q, telemetry stream $\lambda_s(t)$
1	for each change set Δ: build image, sign digest, compute $R(\Delta) = \Sigma_{s\in\Delta}\, g_s$ and $S_{CAMF}(\Delta)$ by Eq. (10)
2	generate and execute IQ/OQ/PQ suites for $S_{CAMF}(\Delta)$; attach evidence to change record
3	if all suites pass and QA approves evidence: add digest to Q; else reject change
4	every minute, for each service s: update ℓ_t, b_t by Eqs. (5)-(6); compute $n_s^*(t)$ by Eq. (7)
5	apply Eq. (8); start new replicas only from digests in Q (policy engine enforces)
6	for each business transaction: commit business data and audit record to the local outbox atomically
7	relay outbox entries to the ledger; compute h_i by Eq. (9); write anchor A_k every B records
8	periodically verify each segment against anchors; raise a deviation and trigger change control on mismatch
Output	Qualified, elastically scaled system state with a verifiable audit trail and evidence per change

4. Results and Discussion

4.1. Baseline performance

The findings at baseline load across 30 seeds are presented in Table 7 below, while Figure 4 shows the metrics of latency, SLO and errors. At baseline load, the monolith architecture exhibited the lowest median latency (25.6 ms) since there are in-process calls that avoid network hops, but it exhibited the highest p95 and p99 latencies (131.0 ms and 215.4 ms) because of the database lock at peak times. The monolith was provisioned at peak, hence it did not violate any SLO and experienced no errors at this load.

Table 7: Baseline performance at load multiplier 1 (mean ± SD over 30 seeds).

Metric	VM	LSM	MSA	CAMF
p50 latency (ms)	25.6 ± 0.1	48.1 ± 0.1	40.2 ± 0.1	40.7 ± 0.1
p95 latency (ms)	131.0 ± 0.2	120.5 ± 1.2	112.1 ± 0.8	111.5 ± 0.3
p99 latency (ms)	215.4 ± 0.9	173.9 ± 26.8	162.1 ± 4.7	156.6 ± 1.2
SLO-violating minutes (%)	0.00 ± 0.00	0.33 ± 0.27	0.33 ± 0.23	0.04 ± 0.06
Error rate (%)	0.00 ± 0.00	0.49 ± 0.47	0.46 ± 0.40	0.03 ± 0.06

Metric	VM	LSM	MSA	CAMF
Throughput (req/s)	277.1 ± 6.9	276.2 ± 6.2	276.2 ± 6.3	277.0 ± 6.8
Daily cost (USD)	110.59 ± 0.00	48.75 ± 0.11	48.75 ± 0.11	63.61 ± 0.17
Unit cost (USD / 10^6 req)	4.62 ± 0.11	2.04 ± 0.04	2.04 ± 0.04	2.66 ± 0.06

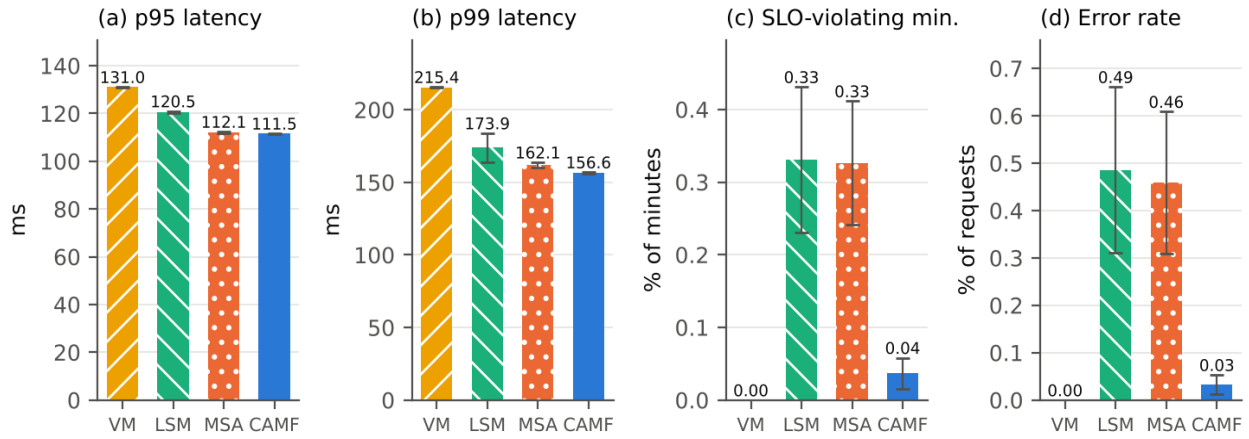

Figure 4: Baseline latency, SLO compliance, and error rate by architecture (error bars: 95% confidence intervals, 30 seeds).

Of the three elastic architectures, CAMF had the smallest tail latency (p95 111.5 ms, p99 156.6 ms), and decreased SLO-violating minutes by 88.7%, as well as error rate by 92.9%, compared to MSA. This was due to predictive scale-up with headroom depending on the criticality of the task, because the reactive autoscalers of LSM and MSA were not able to keep up with the bursts and thus the beginning minutes of every burst were processed by too few replicas, while CAMF had more replicas ready before the burst. However, LSM suffered an extra latency of about 8.4 ms on p95, since every GxP transaction required synchronous commit to the central audit database. Figure 3(b) presents simulation of one single day (seed 0). During this day, all three elastic architectures had exceeded the objective at the beginning of the biggest midday burst, and CAMF experienced one additional violating minute due to cold starts of the numerous replicas created during the burst. Thus, CAMF's performance advantage in Table 7 is an outcome of the distribution over 30 different seeds and not of every day individually.

The reliability improvement in CAMF comes at a price. Its higher floor capacity and lower utilisation requirements for high-criticality services resulted in an increased unit cost of USD 63.61 compared with MSA and LSM costs of USD 48.75, a reliability premium of 30.1% per request. The premium is justified. In a GxP environment, rejecting a result entry or timing out on a batch record review leads to rework and potentially a deviation. As such, it makes sense from a design perspective to preserve capacity for critical functionalities. Despite that premium, CAMF had a 42.5% lower cost per unit than the monolithic architecture.

Table 8 shows the results of the statistical tests. ANOVA revealed that there were statistically significant differences between architectures for all four metrics investigated ($p < 0.001$). In the Tukey post-hoc test, CAMF differed from LSM and MSA architectures in p95 latency, number of minutes violating SLOs, and error rate and from all three architectures in cost per unit and validation effort. There were no statistically significant differences between CAMF and the over-provisioned monolith architecture in number of minutes violating SLOs ($p = 0.852$) and error rate ($p = 0.977$).

Table 8: One-way ANOVA and Tukey HSD p-values for CAMF versus each baseline (30 seeds).

Metric	F	ANOVA p	vs VM	vs LSM	vs MSA
p95 latency	4,865.0	< 0.001	< 0.001	< 0.001	0.012
SLO-violating minutes	30.5	< 0.001	0.852	< 0.001	< 0.001
Error rate	21.8	< 0.001	0.977	< 0.001	< 0.001
Unit cost	9,023.7	< 0.001	< 0.001	< 0.001	< 0.001
Validation effort	5,898.2	< 0.001	< 0.001	< 0.001	< 0.001

4.2. Scalability

Figure 5 and Table 9 depict the performance behavior of the architectures as the load went from one to eight times the base. The monolith with a fixed amount of capacity could meet the demand up to two times the base but was already saturated at four times the base with p95 latency of 440.2 ms and 6.57% errors and 53.4% errors at eight times the base and throughput of 1,508 req/s. LSM architecture failed at a higher rate due to the fact that the central synchronous audit database became a bottleneck at four times the base as it was being used in all GxP transactions.

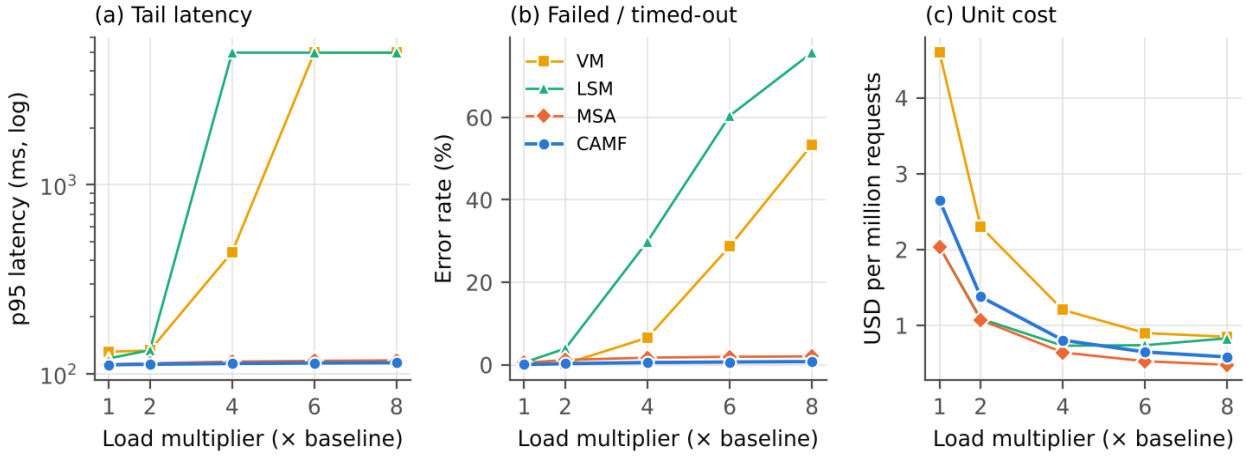

Figure 5: Scalability from one to eight times baseline load: (a) p95 latency (log scale), (b) error rate, and (c) unit cost (10 seeds per point).

Table 9: Scalability results: p95 latency (ms) / error rate (%) / throughput (req/s), mean of 10 seeds.

Load	VM	LSM	MSA	CAMF
1×	131 / 0.00 / 279	121 / 0.55 / 278	112 / 0.48 / 278	112 / 0.06 / 279
2×	133 / 0.25 / 557	133 / 3.94 / 545	114 / 1.17 / 553	112 / 0.30 / 556
4×	440 / 6.57 / 1,063	5,000 / 29.93 / 970	116 / 1.74 / 1,104	113 / 0.55 / 1,111
6×	5,000 / 28.71 / 1,424	5,000 / 60.40 / 1,182	117 / 1.94 / 1,654	114 / 0.68 / 1,666
8×	5,000 / 53.37 / 1,508	5,000 / 75.70 / 1,277	118 / 2.05 / 2,204	115 / 0.78 / 2,220

Both MSA and CAMF showed nearly linear scaling, handling 2,200 req/s at eight times the baseline workload, while the p95 latency was below 120 ms. CAMF experienced a failure rate of 0.78% at eight times the baseline workload compared to 2.05% for MSA and an SLO violation of 1.11% minutes compared to 1.82% for MSA. Unit cost decreased with the workload for all elastic architectures due to overheads being averaged (see Figure 5(c)), and CAMF incurred a cost of USD 0.59 per million requests versus USD 0.48 for MSA at eight times baseline, reducing the premium by 22%.

4.3. Audit-trail integrity and resilience

Table 10 provides an overview of the tamper-injection experiment, and Figure 6(c) depicts the detection rates. The hash-chain verification against anchors could detect 100.0% of the injected events, including reordering and back-dated insertion, for all the seeds. On the other hand, the relational audit table with monotonic sequence identifiers could detect deletions using sequence gaps and insertions using duplicate identifiers, while it could only detect modification if the tampered record turned out to be a part of the 5% sample that was periodically reviewed; reordering could not be detected in any way, and the total detection rate was 51.6% ± 6.3%. Without sequence identifiers and chaining, the sidecar log store could not do better than 1.8% based solely on sampling. Construction and verification of a chain of 20,000 records required 10.6 and 11.2 ms, respectively, which is about 1.9 million records per second.

Table 10: Audit-trail tamper detection (40 injected events per 20,000-record log; 30 seeds).

Audit design	Used by	Detection mechanism	Detected (%)
Relational table with sequence IDs	VM, LSM	Gap / duplicate-key check + 5% review	51.6 ± 6.3
Sidecar log store	MSA	5% periodic review only	1.8 ± 1.9
Hash-chained ledger with anchors	CAMF	Segment re-verification (Eq. 9)	100.0 ± 0.0

Audit-outage test for LSM and MSA microservices (Table 11) demonstrated the opposite types of failure modes of these two standard microservices. For LSM microservices, where the audit message goes by the synchronous path, all of the GxP transactions attempted during the 15 minutes of the outage period were blocked, which is the right response from the data integrity perspective; however, it increased the daily error rate to 2.10%, compared to the previous level of 0.49%. For MSA microservices, where the audit message goes by the asynchronous sidecar, transactions could proceed successfully, yet 0.058% of audit messages of the day were lost, which is the direct ALCOA+ completeness issue that cannot be compensated by performance.

Table 11: Effect of a 15-minute audit-store outage at peak (mean of 30 seeds).

Architecture	Audit write path	Error rate (%)	Audit records lost (%)	p95 (ms)
LSM	Synchronous, central	2.10	0.000	120.5
MSA	Async. sidecar buffer	0.46	0.058	112.1
CAMF	Transactional outbox	0.03	0.000	111.5

4.4. Validation effort and change lead time

The validation effort and lead times for 24 releases per seed is presented in Table 12 and Figures 6(a) and 6(b). The monolith needs full regression of about 3,300 test scripts for each release, taking 2,073 person hours and 117.4 days of lead time. Decoupling by itself was beneficial; the LSM and MSA tested the impacted services and its transitive consumers, which were about 1,148 scripts, resulting in 462 and 347 person hours of effort, respectively. High standard deviation for the above architectures (as seen in Table 12) is because of the heavy tailed dependency graph, where change in one of the hub services impacts the entire landscape.

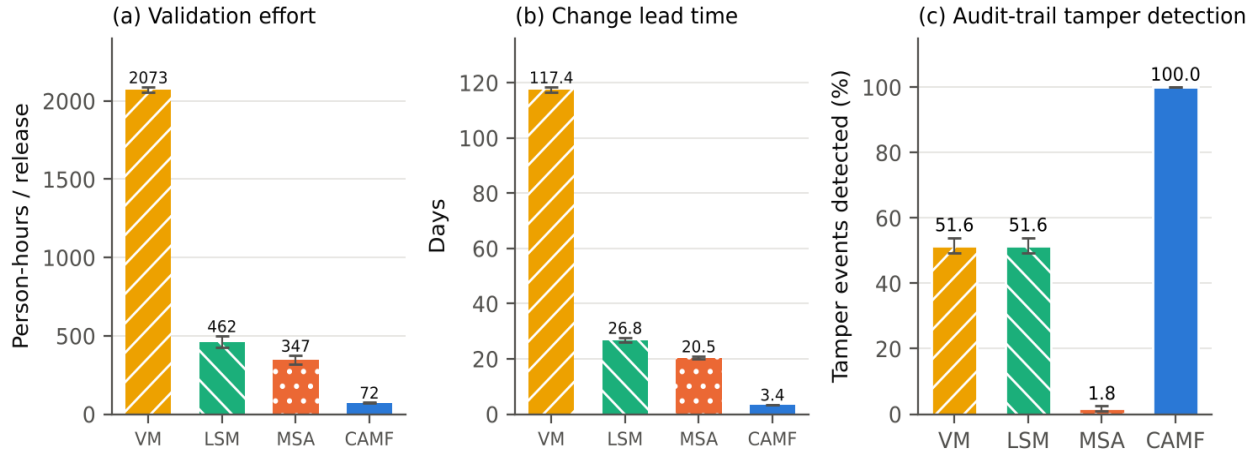

Figure 6: (a) Validation effort per release, (b) change lead time, and (c) audit-trail tamper-detection rate by architecture (error bars: 95% confidence intervals).

Table 12: Validation effort and lead time per release (mean ± SD over 30 seeds, 24 releases each).

Metric	VM	LSM	MSA	CAMF
Test scripts executed	3,300 ± 65	1,148 ± 259	1,148 ± 259	332 ± 33
Validation effort (person-hours)	2,073 ± 40	462 ± 99	347 ± 73	72 ± 6
Change lead time (days)	117.4 ± 2.5	26.8 ± 2.3	20.5 ± 1.4	3.4 ± 0.2

The risk-based scope of CAMF led to the change from full retesting of transitive customers to contract testing for direct customers, which decreased the number of executed tests to 332, and together with an 85% automatic fraction to 72 person-hours per release. This is a decrease of 96.5% compared to the monolith and 79.2% compared to MSA, where lead time is equal to 3.4 days. Moreover, such a bounded scope allowed to estimate the effort precisely, with a standard deviation of 6.0 person-hours compared with 72.7 for MSA.

4.5. Sensitivity analysis

Utilisation targets as defined in Equation (7) define the reliability-cost trade-off of CAMF. A constant shift of Δu was used in targets in the range -0.15 to +0.15 at four times baseline load (10 seeds per setting). As shown in Figure 7 and Table 13, there is a trade-off in CAMF with a monotonic relationship: decreasing targets by 0.15

decreased the error rate from 0.55% to 0.19% with 12.3% increase in unit cost, and increasing targets by 0.15 decreased the cost by 6.3% but increased the error rate to 0.87%. In addition, the p95 latency changed only by 3 ms. This means that the utilisation targets primarily influence the behaviour during bursts. Hence, organisations can adjust the utilisation targets according to their risk profile.

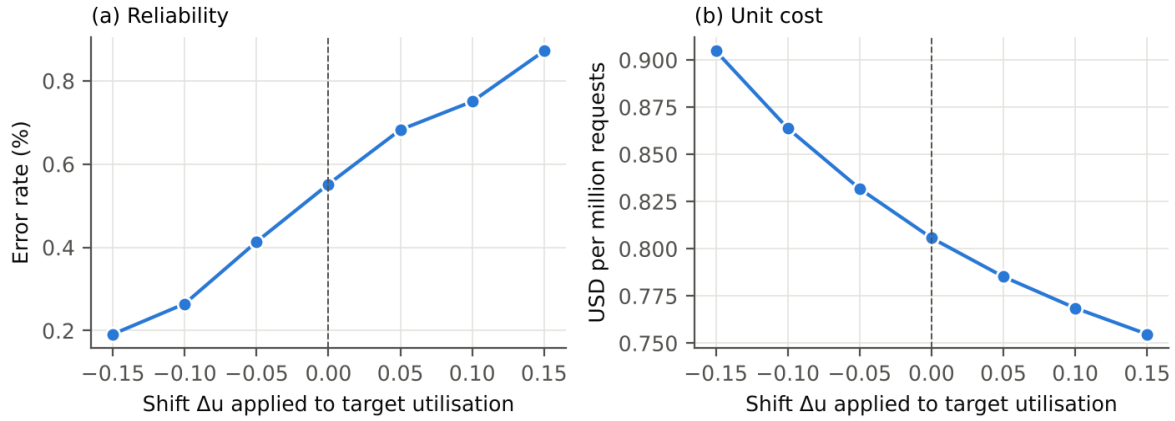

Figure 7: Sensitivity of CAMF to a uniform shift of the utilisation targets at four times baseline load: (a) error rate and (b) unit cost.

Table 13: Sensitivity of CAMF to the utilisation-target shift Δu at four times baseline load (mean of 10 seeds).

Δu	p95 (ms)	p99 (ms)	SLO-violating min. (%)	Error rate (%)	Unit cost (USD / 10^6 req)
-0.15	112.7	162.3	0.71	0.19	0.905
-0.10	113.0	164.8	0.76	0.26	0.864
-0.05	113.2	165.7	0.70	0.41	0.832
0.00	113.5	167.8	0.74	0.55	0.806
+0.05	113.7	169.1	0.71	0.68	0.785
+0.10	114.1	170.5	0.74	0.75	0.769
+0.15	115.0	173.6	0.78	0.87	0.755

5. Discussion

5.1. Implications for practice

From the results obtained, three implications can be drawn concerning the GxP system modernization process for organizations. First of all, the choice of the audit path plays a pivotal role. An LSM approach to auditing limited scalability and rendered the audit store incidents into business interruptions, while the MSA pattern sacrificed audit integrity but provided high system performance. Finally, the TX outbox approach eliminated this dilemma and provided an additional feature of tampering evidence, which was missing in the relational designs, aligning with the patterns of consistency in event-driven processes discovered by Laigner et al. (2021) and Overeem et al. (2021). Second, the autoscaling in the context of regulation should be predictive and controlled. The limitation of the scaling operation to qualified digests provides elasticity of a system in conjunction with the requirement of the constant validated status of a system (ISPE, 2022). Third, the reduction of effort from microservice decomposition can be achieved only in the case of a limited scope of a change. Absent the usage of contract tests, the heavy-tailed dependency structure of realistic microservice landscapes (Luo et al., 2021) implies that changes in hub services lead to near-global re-testing. Risk-based assurance promoted by the FDA (2022) applied to the dependency graph transforms the decomposition into a consistent reduction in effort.

From the perspective of auditors and quality units, the use of CAMF technology brings about a shift in the type of evidence available. Rather than reviewing a selected sample of audit records periodically manually, the ledger offers a technical integrity test of all audit records, whereas the validation process ties machine-generated execution records to each change. Execution records comply with the requirements for evidence to be attributable and concurrent (MHRA, 2018; PIC/S, 2021).

5.2. Limitations and threats to validity

Since the evaluation is based on a simulation and not a production GxP system, there are certain constraints involved. The workload and service characteristics have been generated since no publicly available trace of a

regulated laboratory or quality system is available; these have been used to replicate the qualitative features of a published production trace, while the latencies and costs cannot be considered as predictions of a particular system. The M/M/c modeling does not take into account CPU interference in case of co-located containers, as pointed by Luo et al. (2021). The validation-effort model is based on assumptions regarding automated fractions, effort per test, and effort in documentation (Table 5); the ordering of architectures is resilient to small changes in these figures, although the advantage of CAMF architecture is dependent upon achieving higher levels of test automation that demands investments. The tampering experiment finally presupposes that the anchors are kept at a place that is not tamperable by an adversary capable of tampering with the ledger. This will require that anchors be maintained in write-once storage outside the control of the adversary. Future research will involve verifying the framework using a testbed that runs containers with realistic LIMS traffic, and also doing a case study involving a regulated business, with input from the inspectors regarding the evidence packages.

6. Conclusion

The CAMF framework was proposed in this study as a compliance-aware microservices framework to modernize GxP-validated systems and compared against a validated monolithic system and two traditional microservice architectures. As audit trail integrity, validated state, and change control were considered architectural concerns, CAMF allowed for combining elasticity with data integrity guarantees. In simulations, the framework delivered the reliability level of an over-provisioned monolith at the baseline load while saving 42.5% in terms of unit cost, handled eight times the baseline load while maintaining a failure rate less than 1%, identified all introduced audit trail tampering, suffered from no audit log loss in the case of an audit store failure, and cut the effort for re-validation by 96.5% compared to the monolith per release. However, the price for such guarantees is the 30% compliance premium in terms of unit cost above a non-constrained microservice architecture at baseline load, decreasing as the load increases and adjustable via the criticality-dependent utilization targets. Further research will include the physical testbed experiment on Kubernetes, interference-aware performance model, learning-based forecasting within the qualified state constraint, and acceptance of ledger as evidence by regulators.

References

Al-Debagy, O., & Martinek, P. (2018). A comparative review of microservices and monolithic architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)* (pp. 149–154). IEEE. https://doi.org/10.1109/CINTI.2018.8928192

Alosert, H., Savery, J., Rheaume, J., Cheeks, M., Turner, R., Spencer, C., Farid, S. S., & Goldrick, S. (2022). Data integrity within the biopharmaceutical sector in the era of Industry 4.0. *Biotechnology Journal, 17*(6), Article 2100609. https://doi.org/10.1002/biot.202100609

Arden, N. S., Fisher, A. C., Tyner, K., Yu, L. X., Lee, S. L., & Kopcha, M. (2021). Industry 4.0 for pharmaceutical manufacturing: Preparing for the smart factories of the future. *International Journal of Pharmaceutics, 602*, Article 120554. https://doi.org/10.1016/j.ijpharm.2021.120554

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

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T., & Mazzara, M. (2018). From monolithic to microservices: An experience report from the banking domain. *IEEE Software, 35*(3), 50–55. https://doi.org/10.1109/MS.2018.2141026

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. *Communications of the ACM, 59*(5), 50–57. https://doi.org/10.1145/2890784

Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software, 150*, 77–97. https://doi.org/10.1016/j.jss.2019.01.001

Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In M. Mazzara & B. Meyer (Eds.), *Present and ulterior software engineering* (pp. 195–216). Springer. https://doi.org/10.1007/978-3-319-67425-4_12

European Medicines Agency. (2023). *Guideline on computerised systems and electronic data in clinical trials* (EMA/INS/GCP/112288/2023). https://www.ema.europa.eu/en/documents/regulatory-procedural-guideline/guideline-computerised-systems-electronic-data-clinical-trials_en.pdf

Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015). An updated performance comparison of virtual machines and Linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 171–172). IEEE. https://doi.org/10.1109/ISPASS.2015.7095802

Fritzsch, J., Bogner, J., Wagner, S., & Zimmermann, A. (2019). Microservices migration in industry: Intentions, strategies, and challenges. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 481–490). IEEE. https://doi.org/10.1109/ICSME.2019.00081

Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., Hu, K., Pancholi, M., He, Y., Clancy, B., Colen, C., Wen, F., Leung, C., Wang, S., Zaruvinsky, L., . . . Delimitrou, C. (2019). An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)* (pp. 3–18). ACM. https://doi.org/10.1145/3297858.3304013

Gysel, M., Kölbener, L., Giersche, W., & Zimmermann, O. (2016). Service Cutter: A systematic approach to service decomposition. In *Service-Oriented and Cloud Computing (ESOCC 2016)* (Lecture Notes in Computer Science, Vol. 9846, pp. 185–200). Springer. https://doi.org/10.1007/978-3-319-44482-6_12

Hannousse, A., & Yahiouche, S. (2021). Securing microservices and microservice architectures: A systematic mapping study. *Computer Science Review, 41*, Article 100415. https://doi.org/10.1016/j.cosrev.2021.100415

Hole, G., Hole, A. S., & McFalone-Shaw, I. (2021). Digitalization in pharmaceutical industry: What to focus on under the digital implementation process? *International Journal of Pharmaceutics: X, 3*, Article 100095. https://doi.org/10.1016/j.ijpx.2021.100095

International Society for Pharmaceutical Engineering. (2022). *ISPE GAMP® 5 guide: A risk-based approach to compliant GxP computerized systems* (2nd ed.). ISPE. https://ispe.org/publications/guidance-documents/gamp-5-guide-2nd-edition

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software, 35*(3), 24–35. https://doi.org/10.1109/MS.2018.2141039

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

Laigner, R., Zhou, Y., Salles, M. A. V., Liu, Y., & Kalinowski, M. (2021). Data management in microservices: State of the practice, challenges, and research directions. *Proceedings of the VLDB Endowment, 14*(13), 3348–3361. https://doi.org/10.14778/3484224.3484232

Laukkarinen, T., Kuusinen, K., & Mikkonen, T. (2017). DevOps in regulated software development: Case medical devices. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)* (pp. 15–18). IEEE. https://doi.org/10.1109/ICSE-NIER.2017.20

Laukkarinen, T., Kuusinen, K., & Mikkonen, T. (2018). Regulated software meets DevOps. *Information and Software Technology, 97*, 176–178. https://doi.org/10.1016/j.infsof.2018.01.011

Leal, F., Chis, A. E., Caton, S., González-Vélez, H., García-Gómez, J. M., Durá, M., Sánchez-García, A., Sáez, C., Karageorgos, A., Gerogiannis, V. C., Xenakis, A., Lallas, E., Ntounas, T., Vasileiou, E., Mountzouris, G., Otti, B., Pucci, P., Papini, R., Cerrai, D., & Mier, M. (2021). Smart pharmaceutical manufacturing: Ensuring end-to-end traceability and data integrity in medicine production. *Big Data Research, 24*, Article 100172. https://doi.org/10.1016/j.bdr.2020.100172

Luo, S., Xu, H., Lu, C., Ye, K., Xu, G., Zhang, L., Ding, Y., He, J., & Xu, C. (2021). Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '21)* (pp. 412–426). ACM. https://doi.org/10.1145/3472883.3487003

Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)* (pp. 524–531). IEEE. https://doi.org/10.1109/ICWS.2017.61

Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A., Larsen, S. T., & Dustdar, S. (2021). Microservices: Migration of a mission critical system. *IEEE Transactions on Services Computing, 14*(5), 1464–1477. https://doi.org/10.1109/TSC.2018.2889087

Medicines and Healthcare products Regulatory Agency. (2018). *'GXP' data integrity guidance and definitions* (Revision 1). https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/687246/MHRA_GxP_data_integrity_guide_March_edited_Final.pdf

Moyón, F., Soares, R., Pinto-Albuquerque, M., Mendez, D., & Beckers, K. (2020). Integration of security standards in DevOps pipelines: An industry case study. In M. Morisio, M. Torchiano, & A. Jedlitschka (Eds.), *Product-Focused Software Process Improvement (PROFES 2020)* (Lecture Notes in Computer Science, Vol. 12562). Springer. https://doi.org/10.1007/978-3-030-64148-1_27

Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., & Kim, S. (2020). Horizontal pod autoscaling in Kubernetes for elastic container orchestration. *Sensors, 20*(16), Article 4621. https://doi.org/10.3390/s20164621

Overeem, M., Spoor, M., Jansen, S., & Brinkkemper, S. (2021). An empirical characterization of event sourced systems and their schema evolution—Lessons from industry. *Journal of Systems and Software, 178*, Article 110970. https://doi.org/10.1016/j.jss.2021.110970

Pahl, C., & Jamshidi, P. (2016). Microservices: A systematic mapping study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016)* (pp. 137–146). SciTePress. https://doi.org/10.5220/0005785501370146

Pharmaceutical Inspection Co-operation Scheme. (2021). *Good practices for data management and integrity in regulated GMP/GDP environments* (PI 041-1). https://picscheme.org/docview/4234

Qiu, H., Banerjee, S. S., Jha, S., Kalbarczyk, Z. T., & Iyer, R. K. (2020). FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices. In *14th USENIX Symposium on Operating Systems Design and*

Implementation (OSDI 20) (pp. 805–825). USENIX Association. https://www.usenix.org/conference/osdi20/presentation/qiu

Qu, C., Calheiros, R. N., & Buyya, R. (2018). Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys, 51*(4), Article 73. https://doi.org/10.1145/3148149

Rantanen, J., & Khinast, J. (2015). The future of pharmaceutical manufacturing sciences. *Journal of Pharmaceutical Sciences, 104*(11), 3612–3638. https://doi.org/10.1002/jps.24594

Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmierek, J., Nowak, P., Strack, B., Witusowski, P., Hand, S., & Wilkes, J. (2020). Autopilot: Workload autoscaling at Google. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)* (Article 16). ACM. https://doi.org/10.1145/3342195.3387524

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access, 5*, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

Soldani, J., Tamburri, D. A., & Van Den Heuvel, W.-J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software, 146*, 215–232. https://doi.org/10.1016/j.jss.2018.09.082

Steinwandter, V., & Herwig, C. (2019). Provable data integrity in the pharmaceutical industry based on version control systems and the blockchain. *PDA Journal of Pharmaceutical Science and Technology, 73*(4), 373–390. https://doi.org/10.5731/pdajpst.2018.009407

Sutton, A., & Samavi, R. (2017). Blockchain enabled privacy audit logs. In *The Semantic Web – ISWC 2017* (Lecture Notes in Computer Science, Vol. 10587, pp. 645–660). Springer. https://doi.org/10.1007/978-3-319-68288-4_38

Taherizadeh, S., & Stankovski, V. (2019). Dynamic multi-level auto-scaling rules for containerized applications. *The Computer Journal, 62*(2), 174–197. https://doi.org/10.1093/comjnl/bxy043

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing, 4*(5), 22–32. https://doi.org/10.1109/MCC.2017.4250931

Tirmazi, M., Barker, A., Deng, N., Haque, M. E., Qin, Z. G., Hand, S., Harchol-Balter, M., & Wilkes, J. (2020). Borg: The next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)* (Article 30). ACM. https://doi.org/10.1145/3342195.3387517

U.S. Food and Drug Administration. (2018). *Data integrity and compliance with drug CGMP: Questions and answers—Guidance for industry*. https://www.fda.gov/media/119267/download

U.S. Food and Drug Administration. (2022). *Computer software assurance for production and quality system software: Draft guidance for industry and Food and Drug Administration staff*. https://www.fda.gov/media/161521/download

Ueda, T., Nakaike, T., & Ohara, M. (2016). Workload characterization for microservices. In *2016 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 85–94). IEEE. https://doi.org/10.1109/IISWC.2016.7581269

Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)* (pp. 583–590). IEEE. https://doi.org/10.1109/ColumbianCC.2015.7333476

Waseem, M., Liang, P., & Shahin, M. (2020). A systematic mapping study on microservices architecture in DevOps. *Journal of Systems and Software, 170*, Article 110798. https://doi.org/10.1016/j.jss.2020.110798

World Health Organization. (2016). Annex 5: Guidance on good data and record management practices. In *WHO Expert Committee on Specifications for Pharmaceutical Preparations: Fiftieth report* (WHO Technical Report Series, No. 996). https://www.who.int/medicines/publications/pharmprep/WHO_TRS_996_annex05.pdf

Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Li, W., & Ding, D. (2021). Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering, 47*(2), 243–260. https://doi.org/10.1109/TSE.2018.2887384