ESP-JETA
ESP Journal of Engineering & Technology Advancements
ISSN: 2583-2646 / Volume 1 Issue 1, September, 2021
Paper Id: JETA-V1I1P132/ Page No: 298-301

Research Article

Performance Testing Strategy for Cloud Migration on AWS

Gunjan Chaudhari
Performance Engineering, QualityKiosk Technologies, Mumbai, India

Received Date: 12 July 2021 Revised Date: 10 August 2021 Accepted Date: 08 September 2021

Abstract: *As enterprises transition legacy workloads to Amazon Web Services (AWS), the shift from static on-premises infrastructure to dynamic, software-defined environments introduces unique performance risks. This paper outlines a comprehensive performance testing strategy designed to mitigate latency, throughput, and scalability bottlenecks during the migration lifecycle. Grounded in foundational cloud-computing scholarship and the AWS Well-Architected Framework, the strategy integrates "Shift-Left" testing methodologies with AWS-native observability tooling so that migrated applications meet or exceed their pre-migration baseline performance metrics. We examine the roles of load simulation, resource right-sizing, and post-cutover optimization in sustaining operational excellence, and we present a three-phase testing lifecycle — Pre-Migration Baselining, During-Migration Validation, and Post-Migration Optimization — together with a set of measurable key performance indicators (KPIs) that organizations can use to validate migration success.*

Keywords: *Auto Scaling, AWS Cloud Migration, AWS CloudWatch, AWS Well-Architected Framework, AWS X-Ray, Capacity Planning, Cloud Migration Validation, Cloud Performance Optimization, Cloud-Native Applications, Continuous Performance Testing, Elastic Scalability, Infrastructure as Code (IaC), Latency Optimization, Load Testing, Observability, Performance Benchmarking, Performance Testing, Resource Right-Sizing, Scalability Testing, Shift-Left Testing, Throughput Analysis.*

I. INTRODUCTION

The migration of enterprise applications to the cloud is no longer a simple "lift-and-shift" operation. Contemporary workloads increasingly rely on high-performance requirements and complex, distributed microservices architectures that demand a rigorous performance testing strategy to prevent post-migration regression. Cloud computing was formally characterized by the National Institute of Standards and Technology as a model enabling on-demand, elastic access to a shared pool of configurable computing resources that can be rapidly provisioned and released [1]. This elasticity is precisely what distinguishes AWS environments from the static, dedicated hardware of traditional data centers, and it is also the source of new performance variables that classical, on-premises performance testing practices were never designed to address.

Armbrust et al. observed early in the development of cloud computing that the illusion of infinite, on-demand resources fundamentally changes how systems should be engineered, tested, and operated, shifting the emphasis from static capacity planning toward continuous elasticity management [2]. Performance testing for AWS migration must therefore account for shared-resource contention, network virtualization, and the elasticity of the underlying cloud fabric — concerns that extend well beyond the functional correctness testing that dominates traditional software quality assurance. Weyuker and Vokolos note that performance testing, unlike functional testing, is concerned with whether a system continues to behave correctly and efficiently under realistic and peak operating conditions, and that performance defects discovered late in a project are disproportionately expensive to correct [3]. This observation motivates the "Shift-Left" principle adopted throughout the strategy proposed in this paper: performance validation should begin before a single workload is migrated, not after cutover.

This paper presents a structured, three-phase performance testing strategy for AWS cloud migration. Section II describes the migration performance problem space together with the materials and methods used to address it, including the three-phase testing lifecycle and the key performance indicators used to evaluate migration success. Section III presents the results and discussion, covering tools, load-simulation methodology, and mitigation strategies for common migration challenges. Section IV concludes the paper.

II. MATERIALS AND METHODS

Migrating workloads to AWS introduces performance variables that are largely absent, or present in a very different form, in on-premises environments: (a) Network Latency — differences in throughput between on-premises hardware and virtualized cloud networks (Virtual Private Cloud, or VPC); (b) Auto-Scaling Lag — improperly tuned scaling policies can cause service spikes or performance drops during sudden traffic influxes; (c) Cost Optimization versus Performance — over-

provisioning leads to wasted spend, while under-provisioning leads to latency and availability risk; and (d) Resource Contention — multi-tenancy and shared underlying infrastructure require different optimization approaches than dedicated physical hardware. Wu, Garg, and Buyya similarly emphasize that in a multi-tenant, Software-as-a-Service style delivery model, resource allocation must be governed by explicit service-level agreements (SLAs), because the elasticity that makes cloud infrastructure cost-efficient also makes performance far more sensitive to allocation and scheduling decisions than it is on dedicated hardware [10].

A robust performance testing strategy for AWS migration is organized into three distinct phases: Pre-Migration Baselining, During-Migration Validation, and Post-Migration Optimization, as illustrated in Figure 1. This phased structure mirrors the phase-driven migration methodology described in AWS's own migration guidance, which recommends assessing, piloting, and iteratively migrating workloads rather than attempting a single, monolithic cutover [5].

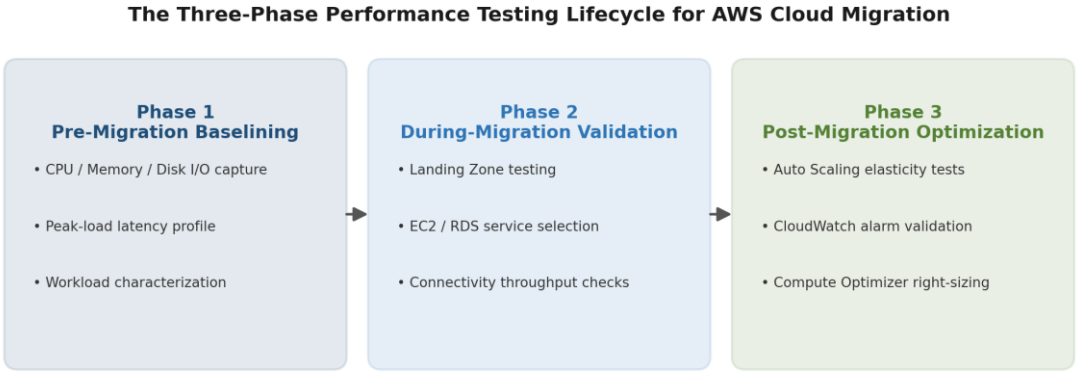

Figure 1: The Three-Phase Performance Testing Lifecycle for AWS Cloud Migration.

A. Phase 1: Pre-Migration Baselining

Before any workload or dataset is moved, it is critical to establish a "Gold Standard" of current, on-premises performance against which all post-migration results will be compared. This includes Metric Collection — capturing CPU utilization, memory footprint, disk I/O (IOPS), and network latency under peak loads — and Workload Characterization — identifying "zombie" or idle applications to avoid migrating unnecessary technical debt into the target environment.

B. Phase 2: During-Migration Validation

This phase involves testing the application in a "Landing Zone" on AWS that mimics the intended production environment as closely as possible. Service Selection validates that the chosen EC2 instance types or RDS database engines match the performance requirements defined during Phase 1 baselining, while Connectivity Testing measures throughput between on-premises databases and AWS-based application tiers during any hybrid or phased cutover window.

C. Phase 3: Post-Migration Optimization

Once cutover is complete, the focus shifts from parity validation to cloud-native scaling behavior. Elasticity Testing validates that Auto Scaling groups trigger correctly based on Amazon CloudWatch alarms under simulated traffic ramps, and the Cost-Performance Tradeoff is assessed using right-sizing tooling to identify over-provisioned resources without violating the performance thresholds established in Phase 1.

D. Key Performance Indicators (KPIs)

Performance success is measured across four primary domains — compute, storage, network, and database — as summarized in Table 1. Each metric is paired with a target or success criterion derived from the Phase 1 baseline, consistent with the principle that performance requirements should be measurable and testable rather than qualitative [6].

Table 1: Performance Metrics and Thresholds

Domain	Metric	Target / Success Criteria
Compute	CPU / Memory Stability	< 70% utilization during peak
Storage	EBS Latency	< 10 ms for transactional databases
Network	TTFB (Time to First Byte)	Within 5% of on-premises baseline
Database	Query Response Time	No regression from legacy SQL/NoSQL

III. RESULTS AND DISCUSSION

A. Tools and Methodology

Amazon CloudWatch provides real-time telemetry collection and log aggregation across compute, storage, and network resources [11]. AWS Distributed Load Testing is an automated solution used to simulate thousands of concurrent users against a target environment, and AWS X-Ray provides distributed tracing used to identify latency bottlenecks across microservice call chains. We recommend a "Step-Load" approach, in which concurrency is increased incrementally to identify the "breaking point" of the migrated architecture, consistent with classical performance testing methodology for identifying the operational envelope of a system under test [3]. The relationship between throughput (T) and concurrency (C) can be modeled as T = C / R, where R is the average response time. In an elastic cloud environment, R should remain approximately stable as C increases, provided that horizontal scaling keeps pace with demand.

B. Load Simulation and Throughput Analysis

Figure 2 illustrates the throughput–concurrency relationship under two scenarios: one in which Auto Scaling responds correctly and response time remains stable, and one in which scaling lag causes response time — and therefore throughput — to degrade beyond a threshold level of concurrency. Autonomous, demand-driven resource provisioning of the kind modeled here has long been recognized in the distributed-systems literature as necessary for multi-tier web applications to sustain throughput under bursty load [8].

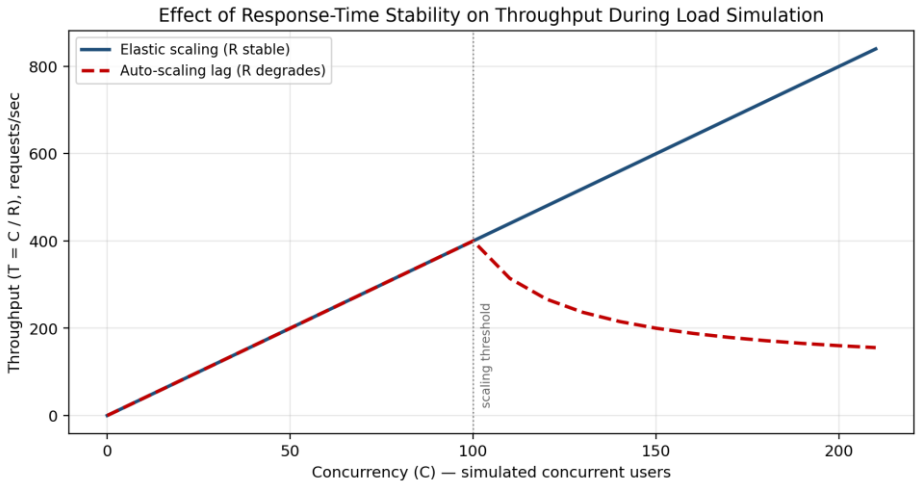

Figure 2: Effect of Response-Time Stability on Throughput as Simulated Concurrency Increases

As shown in Figure 2, when the average response time R remains stable through correctly tuned horizontal scaling, throughput continues to grow linearly with concurrency. When Auto Scaling policies lag behind demand, however, R begins to climb once a threshold concurrency level is reached, and throughput plateaus and then declines even as offered load continues to increase. This is precisely the auto-scaling lag failure mode described in Section II, and it is the primary reason elasticity testing is treated as a distinct, mandatory activity in Phase 3 rather than an incidental outcome of functional testing.

C. Challenges and Mitigation

Two recurring challenges emerged from this analysis. Data Gravity: large datasets can cause significant latency during synchronization between on-premises and cloud environments; the recommended mitigation is to use AWS Database Migration Service (DMS) with Change Data Capture (CDC) to minimize synchronization windows. Cold Starts: serverless functions (AWS Lambda) may experience latency on infrequently invoked code paths; the recommended mitigation is to implement Provisioned Concurrency for performance-critical functions. Both challenges illustrate a broader theme in cloud performance engineering: many of the most damaging performance defects in a migrated system are not defects in the application code itself, but defects in how the application interacts with the elastic, multi-tenant, and event-driven characteristics of the platform it now runs on [2], [9]. A performance testing strategy that only re-runs pre-migration functional and load tests without specifically targeting these cloud-native failure modes will systematically miss them.

IV. CONCLUSION

A successful AWS migration is not measured by the speed of the move, but by the stability of the application in its new environment. By adopting a structured, three-phase performance testing strategy that emphasizes early baselining, rigorous during-migration validation, and continuous post-migration observability, organizations can leverage the full elasticity and cost efficiency of the cloud without compromising the user experience. The KPI framework and load-simulation

methodology presented in this paper give practitioners a repeatable, measurable basis for validating that a migrated workload meets or exceeds its pre-migration performance baseline, consistent with the operational excellence and performance efficiency pillars of the AWS Well-Architected Framework [4].

- **Interest Conflicts:** The author(s) declare that there is no conflict of interest concerning the publication of this paper.
- **Funding Statement:** This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.
- **Acknowledgments:** The author gratefully acknowledges the AWS Well-Architected Framework documentation and the foundational cloud-computing and performance-testing literature cited in this paper for informing the strategy presented here.

V. REFERENCES

[1] P. Mell and T. Grance, The NIST Definition of Cloud Computing, National Institute of Standards and Technology, Special Publication 800-145, Gaithersburg, MD (2011).

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, A View of Cloud Computing, Communications of the ACM 53(4) (2010) 50–58.

[3] E. J. Weyuker and F. I. Vokolos, Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study, IEEE Transactions on Software Engineering 26(12) (2000) 1147–1156.

[4] Amazon Web Services, AWS Well-Architected Framework, Amazon Web Services Whitepaper (2019).

[5] J. Varia, Migrating your Existing Applications to the AWS Cloud: A Phase-driven Approach to Cloud Migration, Amazon Web Services Whitepaper (2010).

[6] IEEE Standard for Software and System Test Documentation, IEEE Std 829-2008, Institute of Electrical and Electronics Engineers (2008).

[7] EEE Standard for Software Unit Testing, IEEE Std 1008-1987, Institute of Electrical and Electronics Engineers (1986).

[8] D. Jiang and G. Pierre, Autonomous Resource Provisioning for Multi-Service Web Applications, in Proc. 19th International Conference on World Wide Web (WWW '10), Raleigh, NC (2010) 471–480.

[9] B. Furht and A. Escalante, Eds., Handbook of Cloud Computing, Springer, New York, NY (2010).

[10] L. Wu, S. K. Garg, and R. Buyya, SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments, in Proc. 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Newport Beach, CA (2011) 195–204.

[11] Amazon Web Services, Amazon CloudWatch User Guide, Amazon Web Services Documentation (2019).