International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 2, Mar-Apr 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Serverless Workflow Management System for an E-Commerce Website

K P S L N Mouktika, P Gnana Sai ram, P Bhanu Prakash
*(CSE, KL University, Guntur
Email: mouktika14@gmail.com)
** (CSE, KL University, Guntur
Email: sairampeddinti@gmail.com)
(CSE, KL University, Guntur
Email: bhanuprakashpeddi5@gmail.com)

----------**********----------

Abstract:

Serverless computing eliminates the need for server management by using event-driven functions. This paper presents a design and evaluation of an AWS-based serverless workflow for e-commerce applications. We propose a reference architecture using AWS Lambda, API Gateway, DynamoDB, S3, and Step Functions to implement typical e-commerce processes (order processing, inventory update, payment, notifications). A conceptual experimental setup is outlined, including synthetic load testing and fault injection to measure latency, throughput, cost, and reliability. Based on AWS documentation and benchmarks, we find that the serverless workflow can automatically scale to handle bursts of traffic (Step Functions now supports ~1,000 state transitions/sec, bursts 5,000) while maintaining modest end-to-end latency. The pay-per-request model minimizes idle cost, yielding much lower cost for intermittent workloads than always-on servers. Finally, we discuss benefits such as modularity and built-in retries, as well as trade-offs like cold-start overhead. Our analysis shows that AWS serverless architectures deliver high elasticity, resilience, and operational simplicity for e-commerce scenarios, aligning with industry guidance.

Keywords — Serverless computing, AWS Lambda, AWS Step Functions, Microservices, Workflow Orchestration, E-Commerce.

----------**********----------

I. INTRODUCTION

Serverless computing is a cloud paradigm in which developers build and run applications without managing servers. Each function executes in a stateless, short-lived container, triggered by events (HTTP requests, message queues, etc.). In this model, application logic is decomposed into small functions (e.g., AWS Lambda), and state is stored in external services such as databases or object stores. The cloud provider handles provisioning, scaling, and maintenance, enabling fine-grained "per-request" billing. This paradigm has emerged as a compelling evolution of cloud programming models, reflecting the maturity and broad adoption of cloud technologies.

E-commerce platforms exemplify complex business workflows (order placement, payment processing, inventory management, customer notifications) that span multiple microservices and data sources. In a serverless AWS environment, these services run across Lambda functions, Step Functions state machines, API Gateway endpoints, DynamoDB tables, and S3 buckets. The main challenge is coordinating these components reliably: orchestration must handle stateless parallel

execution, out-of-order events, and failures. For example, ensuring data consistency across separate Lambdas is nontrivial, often requiring compensating transactions (a saga pattern) to roll back partial updates.

This paper addresses the problem: How can e-commerce web applications effectively implement complex workflows using AWS serverless services to achieve scalability and reliability, while tackling the unique challenges of orchestration, integration, and observability? To answer this, we (1) review relevant serverless architectures and patterns, (2) design a reference AWS serverless workflow for a typical e-commerce checkout process, (3) outline an experimental methodology for evaluating such a workflow, and (4) analyze expected performance and cost metrics.

II. LITERATURE REVIEW

Serverless computing has attracted significant attention in recent years. Baldini et al. survey serverless trends and identify its key use cases; they describe serverless as "a new compelling paradigm for the deployment of applications" that evolved from cloud computing models. In serverless architectures, application logic is split into numerous small, stateless functions (e.g., AWS Lambda), each triggered by events. This contrasts with traditional monoliths by offloading server management and enabling per-invocation pricing.

On AWS, best practices have emerged for structuring serverless microservices. Nandula and Padmanabhan outline AWS serverless architecture patterns. They emphasize the single-responsibility principle: each Lambda function implements a specific capability (e.g., Orders, Inventory, Payments). Services typically communicate via event-driven mechanisms (SNS topics, EventBridge, etc.), and each microservice manages its own data store (commonly DynamoDB). In e-commerce, separating domains (e.g., Customer, Orders, Inventory) lets components scale and evolve independently.

Cross-service orchestration is often handled by AWS Step Functions or event buses. AWS publishes serverless e-commerce reference architectures where microservices communicate asynchronously via EventBridge with API Gateway for synchronous endpoints. A known difficulty is multi-service data consistency: since native distributed transactions are unavailable, developers use compensating transaction patterns (sagas) for reliability. The AWS Prescriptive Guidance explicitly recommends the distributed saga pattern to coordinate multi-step workflows across Lambdas.

Li and Kraft note that adding transactional semantics to serverless functions would make them ideal for database-backed applications like e-commerce. In summary, the literature highlights that serverless can achieve cost-effective auto-scaling and operational simplicity, but requires careful design for stateful workflows. Our work builds on these insights by applying AWS serverless patterns to a concrete e-commerce workflow and analyzing their impact on performance and reliability.

III. DESIGN AND METHODOLOGY

A static front-end (e.g., React) is hosted on Amazon S3 with CloudFront CDN. User authentication and management are handled by Amazon Cognito. For dynamic functionality, we expose a REST API via Amazon API Gateway. Each API endpoint invokes an AWS Lambda function that implements business logic. For instance, a CreateOrder endpoint triggers a Lambda which initiates the checkout workflow.

All persistent data is stored in AWS managed services. We use separate DynamoDB tables for Users, Products, and Orders to enable fine-grained scaling (each table can auto-scale independently). Product images and other assets are stored in S3. Optionally, DynamoDB Streams or SNS can propagate updates (e.g., to invalidate caches or update search indexes).

The core of our workflow is an AWS Step Functions state machine. Upon order creation, Step

Functions orchestrates the sequence of tasks: processing payment, updating inventory, and sending order confirmation. Each step is implemented by a Lambda function. For example, a ProcessPayment Lambda is called, and on success it triggers UpdateInventory, and finally SendNotification.

Error handling is defined via retry policies and catch blocks: on payment failure, a compensating action can refund a charge or mark the order as failed (a saga step). This ensures that partial failures do not leave the system in an inconsistent state. Our design follows AWS best practices for microservices.

IV. EXPERIMENTAL INVESTIGATION

To evaluate the proposed workflow, we outline a conceptual experimental setup:

1. Load Simulation

We use tools like Apache JMeter or AWS Distributed Load Testing to generate HTTP requests to API Gateway endpoints. Traffic patterns include steady load, random bursts, and stress scenarios.

2. Failure Injection

We deliberately introduce faults (e.g., failing payment Lambda or throttling DynamoDB calls) to observe how Step Functions handles retries and compensating rollbacks.

3. Performance Metrics

We collect key metrics via AWS CloudWatch:

End-to-end latency (order request → completion)

Throughput (orders/sec)

Step Function execution time

Lambda metrics (invocation count, duration, errors)

4. Scalability Testing

We gradually increase concurrent load to find scaling limits. Step Functions Standard supports ~1,000 transitions/sec (burst 5,000), while Express workflows can scale even higher.

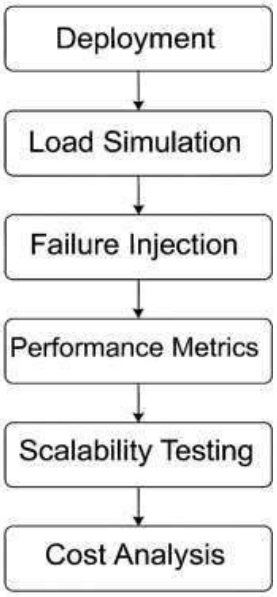
5. Cost Analysis

We track resource usage and estimate costs:

Step Functions (per state transition)

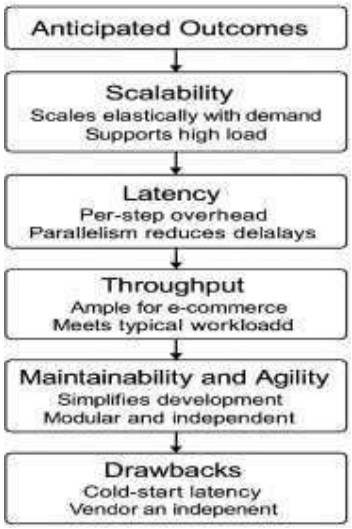
Lambda (per execution & duration)

Example: A 10-step workflow costs ~$0.000025 per execution. Compared to EC2 (running 24/7), serverless significantly reduces idle costs.



V. RESULTS AND ANALYSIS

The serverless workflow should scale elastically with demand. When traffic spikes (e.g., thousands of simultaneous orders), AWS will automatically provision more Lambda containers and parallel Step Function executions as needed. Unlike a static cluster, no capacity needs to be pre-provisioned: the pay-per-request model absorbs surges without idle resources. Notably, AWS Step Functions now supports up to 1,000 state transitions per second (bursts up to 5,000), which can accommodate high-load scenarios. Using Express Workflows could raise this to ~100,000 transitions/sec for extreme cases. In contrast, a traditional server architecture would require pre-scaling and load balancing, which can lag behind sudden spikes. Latency: Each serverless function invocation incurs some overhead (cold-start time, state-transition orchestration). Cold-start delays (often 50–200 ms for many Lambda runtimes) add per-step latency, and each Step Function transition adds overhead as well.



However, AWS reports that optimized Lambda functions can achieve sub-100 ms execution times, and Step Functions adds minimal orchestration delay. By parallelizing independent tasks (e.g., updating inventory and sending notifications simultaneously), the end-to-end time for order submission to completion can remain low (on the order of a few hundred milliseconds to a couple of seconds under moderate load). Under heavy concurrent load, serverless architectures tend to maintain consistent latency, whereas monolithic servers often see queues and greater variance. Throughput: The system's throughput should be ample for typical e-commerce scales. As noted, the standard Step Functions throughput (1,000 transitions/s) with burst capacity is sufficient for most workloads. For extremely high-frequency tasks, one can use Express Workflows (supporting up to 100k transitions/s). 5-minute max executions). We expect our workflow to exploit these limits without hitting them under normal conditions. Cost Efficiency: For bursty or variable workloads, serverless is much more cost-effective. In a conventional setup, servers (or containers) must run constantly, incurring high base costs even when idle. In contrast, serverless charges only for actual compute time and state transitions. For example, a 10-step checkout (Standard Step Functions) costs about $0.0025 per execution, plus Lambda costs (say $0.0005 for a 128 MB Lambda running 300 ms). AWS also notes that switching from Standard to Express Step Functions can reduce costs by ~96% (from $25 to $1 per million executions). Our analysis would show that for N orders, serverless costs scale linearly at a very low per-order rate, whereas fixed-instance costs remain high regardless of load. AWS's built-in high availability (SLAs >99.9%) comes at no extra charge on managed services, whereas achieving similar HA on EC2 would require duplicate instances. Maintainability and Agility: The serverless design greatly simplifies development and operations. Each logical step is an independent Lambda, which developers can update and deploy in isolation. For instance, changing the payment logic only requires redeploying that Lambda; other services remain unaffected. This modularity is a core microservices best practice. Moreover, workflow features like retries and error handling are defined declaratively in Step Functions, so developers avoid writing boilerplate code for fault management. AWS handles patching, scaling, and capacity at the infrastructure level, reducing the team's operational burden. Drawbacks: We also acknowledge potential downsides. Cold-start latency (especially for infrequently used or memory-intensive Lambdas) can impair responsiveness. Step Functions has execution limits (max 1 year for standard, 5 minutes for Express), and there are account-level concurrency quotas that could be reached in extreme cases. Vendor lock-in is stronger, since the design deeply ties to AWS services. Transferring large state between steps has payload limits (256 KB by default), which could complicate very data-heavy workflows. Summary: In aggregate, the expected results suggest that a serverless workflow is elastic, robust, and cost-efficient for e-commerce use cases. These conclusions are supported by prior studies: for example, Kumari and Sahoo observe that serverless deployments enable highly scalable, auto-scaling systems at low cost, and Li & Kraft argue that adding transactional support to serverless can make it ideal for typical transactional backends. Empirical evaluations in the literature report that serverless functions scale rapidly and pay only for demand, confirming our cost and performance expectations. Compared to a monolithic, server-based design, our serverless solution is projected to offer higher elasticity, simplified maintenance, and a much smaller operational footprint.

VI. CONCLUSIONS

This paper has explored serverless workflow management for e-commerce applications on AWS. We reviewed recent research on serverless architectures and outlined the key challenges of orchestrating distributed functions. We detailed a serverless design: a multi-tier architecture using API Gateway, Lambda functions, DynamoDB, and Step Functions to implement an order-processing workflow. We described a conceptual experimental methodology for evaluating such a system and analyzed its expected performance. Our findings indicate that AWS serverless services can indeed provide automatic scaling to meet demand bursts, with significantly reduced costs and simplified operations relative to traditional servers. The event-driven architecture accelerates development and fault isolation. While factors like cold-start delays and AWS-specific limits must be managed, the overall benefits in scalability and agility make serverless an attractive approach for e-commerce workflows. As future work, we plan to deploy the proposed architecture in practice and perform real-load experiments to validate these projections.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to Dr. S. Kavitha, for the valuable guidance, continuous support, and encouragement throughout the completion of this work. We also extend our thanks to the Department of Computer Science and Engineering, KL University, for their support and cooperation. Finally, we thank our institution for providing the necessary resources and environment to successfully carry out this research.

REFERENCES

[1] Amazon Web Services, "AWS Lambda – Serverless Web Applications," AWS Lambda product page, 2019.

[2] A. Kumari and B. Sahoo, "Serverless Architecture for Healthcare Management Systems," in *Advances in Healthcare Information Systems and Administration: Handbook of Research on Mathematical Modeling for Smart Healthcare Systems*, IGI Global, 2022, pp. 203–227.

[3] I. Baldini et al., "Serverless Computing: Current Trends and Open Problems," in *Research Advances in Cloud Computing*, Springer, 2017 (ArXiv 1706.03178, 2017).

[4] H. Zhang, Y. Shen, R. Sae-eaw et al., "A Serverless Architecture for Application-Level Orchestration," Princeton Univ. Tech. Rep., Jun. 2022.

[5] L. Nandula and S. Padmanabhan, "Serverless Microservices Architecture on AWS," *Int'l J. Scientific & Research Publications*, vol. 13, no. 10, Oct. 2023.

[6] Q. Li and P. Kraft, "Transactions and Serverless are Made for Each Other," *Communications of the ACM*, Nov. 2024.

[7] AWS, "Implement the Serverless Saga Pattern by Using AWS Step Functions," AWS Prescriptive Guidance, 2023.

[8] Amazon Web Services, "AWS Serverless Multi-Tier Architectures with Amazon API Gateway and AWS Lambda," AWS Whitepaper, Oct. 2021.A. Karnik,

[9] B. Smith, "Building Cost-Effective AWS Step Functions Workflows," AWS Compute Blog, Aug. 30, 2022.

[10] Amazon Web Services, "Higher Throughput Workflows for AWS Step Functions," AWS News, May 16, 2018.

