iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** III **Month of publication:** March 2026

DOI: https://doi.org/10.22214/ijraset.2026.78878

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

A Generalized Serverless Event-Driven Architecture for Automated Receipt Processing and Data Extraction

Shravan Digambar Komejwar[1], Tushar Ashok Khedkar[2], Ganesh Vijay Kadam[3], Prajyot Randhir Barsing[4], Nitin Talhar[5]
Department of Computer Engineering, AISSMS College of Engineering, Pune, India

Abstract: ***Receipt volumes don't follow a schedule. Quiet for two days, then thirty submissions at once because someone's expense report is due. A server sitting on 24/7 for that is mostly wasted cost. The approach in this paper is based on serverless computing principles — resources remain inactive between executions. Upon data arrival, processing is triggered automatically, structured data is stored in a database system, users are notified. This represents the overall processing flow.***
The architecture consists of multiple functional layers. IPO notation tracks what enters and exits each stage — keeps the flow auditable without overcomplicating it. Paying per invocation beat committing to reserved instances; idle capacity between bursts doesn't bill, which is the whole point. Subtotal aggregation, tax application, and final transaction computation are all spelled out in the methodology. Full runtime benchmarking isn't something this paper gets into — that needs its own controlled test environment. This work focuses on the architectural design, while detailed implementation and performance evaluation are considered as part of future work.
Keywords: Serverless Computing, Event-Driven Architecture, Receipt Processing, Cloud Computing, Cost Optimization, Transaction Processing, IPO Model.

I. INTRODUCTION

Receipt processing at scale has a few recurring problems. Vendors don't format documents consistently. Submission volumes aren't steady — they cluster around month-end, expense reporting windows, tax periods. Systems built for flat, predictable load aren't a good fit. Manual entry falls behind fast. Fixed-capacity servers either sit idle most of the time or fall over when things bunch up.
Storage becomes a problem quickly. Receipt A has a line-item table, a tax breakdown, and a delivery charge. Receipt B is a photo with a single amount and a merchant name.
A relational schema that works for one won't fit the other without preprocessing [1]. Document-style storage that lets each record hold whatever fields it actually contains skips that step entirely.
Serverless platforms follow a pay-per-execution model, eliminating idle resource consumption [2]. That billing model fits receipt processing directly because the load isn't predictable. NoSQL databases complement such systems by enabling flexible, schema-less data storage , so a receipt with ten fields and one with three fields both write without any massaging [3].
OCR processing introduces latency in document pipelines . Run it synchronously and users sit waiting while both recognition and field extraction complete — the extraction step isn't fast on its own, and raw OCR output usually needs a correction pass before it's usable [4]. Using asynchronous processing helps reduce user-perceived delay; upload confirms, processing follows.
Research on RFID billing pipelines showed this type of event-chained cloud architecture handles real-time financial data reliably at scale [5]. Such systems operate using event-driven triggers, where stages are activated automatically, no coordinator in the middle — adapted for document ingestion rather than hardware-generated events.

II. LITERATURE REVIEW

A. Privacy of Business Data in Accounting Systems

Tally Solutions (2023) put out a case study on financial record security in accounting platforms. The systems they looked at handle invoices, client records, transactions — standard stuff, but sensitive. They cover authentication, access control, encryption. The timing argument is the interesting part: building privacy constraints into the design from the start consistently outperforms adding them later [6]. Not a controversial claim, but easy to skip when the team is heads-down trying to ship.

B. Security and Privacy of Cloud Storage Services

Saeed et al. (2019) ran a security comparison of S3 and Azure Blob Storage. Both handle it fine but make different choices in how access policy works. What the data actually showed: most real breaches in cloud storage weren't from platform vulnerabilities — they came from permissions being misconfigured [7]. So picking the 'more secure' platform matters less than whether whoever set it up actually got the config right.

C. Electronic Receipt Management System

Wadsworth et al. (2010) describe a digital receipt system that hooks into existing point-of-sale infrastructure through a protocol layer and stores records centrally. Merchants, consumers, and banks each pull from the same backend. The environmental paper reduction argument takes up some space. More relevant here is the reconciliation angle [8] — chasing down separate copies held by each party is the kind of thing that wastes hours; one shared record mostly eliminates
that.

D. Performance Analysis of AWS Cloud Infrastructure

A passive measurement study on AWS tracked EC2, S3, and CloudFront under real usage. Distance from the data source adds latency — everyone knows that, but the measured values were worse than most rule-of-thumb estimates suggest [9]. CloudFront's caching cuts it down, assuming content placement was actually thought through. If the root cause is geographic, more compute doesn't help.

E. Chunk Processing in HTTP Authorization Protocols

Surkov (2020) looked at large payload handling in HTTP authorization. Sending a big file as one unit is slow, and it opens up more surface for denial-of-service attempts. The study tested chunked transmission — split the payload, process each piece on its own — and it outperformed full-unit handling [10]. For this system specifically, users upload receipt images regularly. That's not a theoretical edge case; it's the normal operation.

However, existing studies primarily focus on individual components, and a unified serverless framework for receipt processing remains relatively unexplored.

III. SYSTEM ARCHITECTURE OVERVIEW

Serverless was picked because OCR demand isn't steady. Quiet periods, then sudden bursts tied to submission deadlines. A provisioned server is either oversized and expensive during the quiet times or undersized when load spikes. Serverless adjusts automatically — cost follows usage, and there's no upfront capacity decision to get wrong.

The event-driven side solves something different. Continuous polling mechanisms are not required in such architectures. When a file is stored in a cloud storage system, it can automatically trigger downstream processing — not a scheduled check, not a queue poll, just that arrival event. So the user's upload confirms right away. What happens after — OCR, field extraction, the DB write — runs in the background on its own time. Backend processing is decoupled from frontend response time.

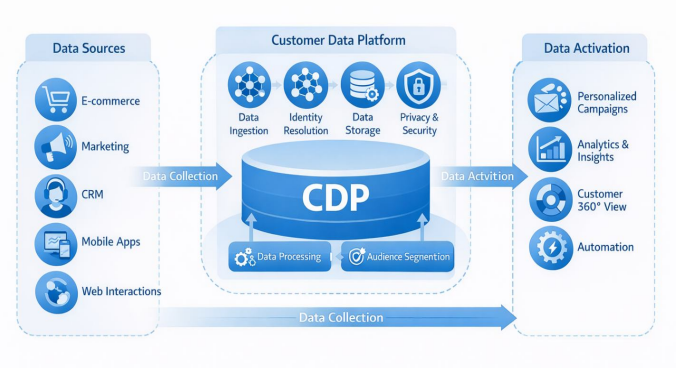


Fig. 1 Generalized Architecture for Receipt Processing

As illustrated in Figure 1, the architecture leverages event-driven execution and modular design to enable scalable, efficient, and asynchronous processing of document-based data.

One thing that came out of the micro-operation structure unexpectedly: system maintenance and debugging become more manageable. Something breaks in OCR, you fix that function and redeploy it. Notification still works. Nothing else touched. A monolithic pipeline doesn't give you that — a failure in one step tends to drag the rest down with it. That isolation is genuinely useful once the system is live.

Table I
Comparison Between Traditional Server-Based Systems And Serverless Architecture

Feature	Traditional Server-Based Systems	Serverless Framework
Resource Utilization	Servers run continuously	Resources used only during execution
Cost Model	Fixed / Pre-allocated	Pay-per-use
Scalability	Limited, requires manual scaling	Automatic and dynamic scaling
Maintenance	High (server management required)	Low (managed services)
Processing Type	Mostly synchronous	Asynchronous and event-driven
Fault Isolation	Low (failure affects system)	High (modular components)
Response Time	Affected by backend load	Decoupled frontend and backend

Table 1 summarizes the key differences between traditional server-based systems and serverless architecture.

IV. METHODOLOGY

Everything runs event-to-event. One stage finishes, the next one starts — nothing sits polling or waiting around. Lambda is built for exactly this: short stateless functions, fired per event, no memory between invocations [2]. The architecture splits into several layers, each doing one job.

First one catches incoming client requests and manages the upload process. Behind that sits a remote storage layer — just holds raw files, nothing else. Then there's a coordination layer built from stateless compute units that don't do heavy processing themselves; they take each job and forward it to wherever it needs to go next. The fourth layer performs document interpretation using OCR techniques, on a managed ML service we don't host ourselves. The final layer handles analytics and notification mechanisms — computes the financial totals, tells the user when their receipt is processed. Keeping these separate means Layer 4 slowing down doesn't stall the others [1].

Uploads skip the backend completely. The system issues a pre-signed URL per request — expires quickly, has S3 write permission baked in already. Client sends the file directly to S3, our servers aren't involved [11]:

$$\text{Input} \rightarrow \text{Processing} \rightarrow \text{Output} \quad (1)$$

Client request (Input) triggers auth and URL generation (Processing), receipt ends up in S3 (Output). That write event fires the next stage on its own — no polling.

Processing picks up Input from storage and hands it off to the Interpretation layer. OCR runs on it, output is text R. That text is messy — you can't just parse it directly, there's an error correction step first before field extraction works reliably [4]. After that, the structured result goes into DynamoDB.

Schema-free, so a receipt with 5 fields and one with 12 both write fine [3]:

$$\text{Raw Data} \rightarrow \text{Extraction} \rightarrow \text{Structured Data} \rightarrow \text{Storage} \quad (2)$$

Async notification goes out after the write confirms — nothing upstream waits on it. Same real-time async pattern used in cloud billing systems [5]. Retrieval is a direct query:

$$\text{Input} \rightarrow \text{Query} \rightarrow \text{Output} \quad (3)$$

A financial analytics component computes key transactional metrics on extracted data. Per-item subtotal:

$$\text{Subtotal} = \Sigma(\text{price}_i \times \text{quantity}_i) \quad (4)$$

Tax on subtotal:

$$\text{Tax} = \text{Subtotal} \times \text{taxRate} \quad (5)$$

Full transaction value:

$$\text{Total} = \text{Subtotal} + \text{Tax} \quad (6)$$

Business-level profit:

$$\text{Profit} = \text{Revenue} - \text{Expenses} \quad (7)$$

V. CONCLUSION

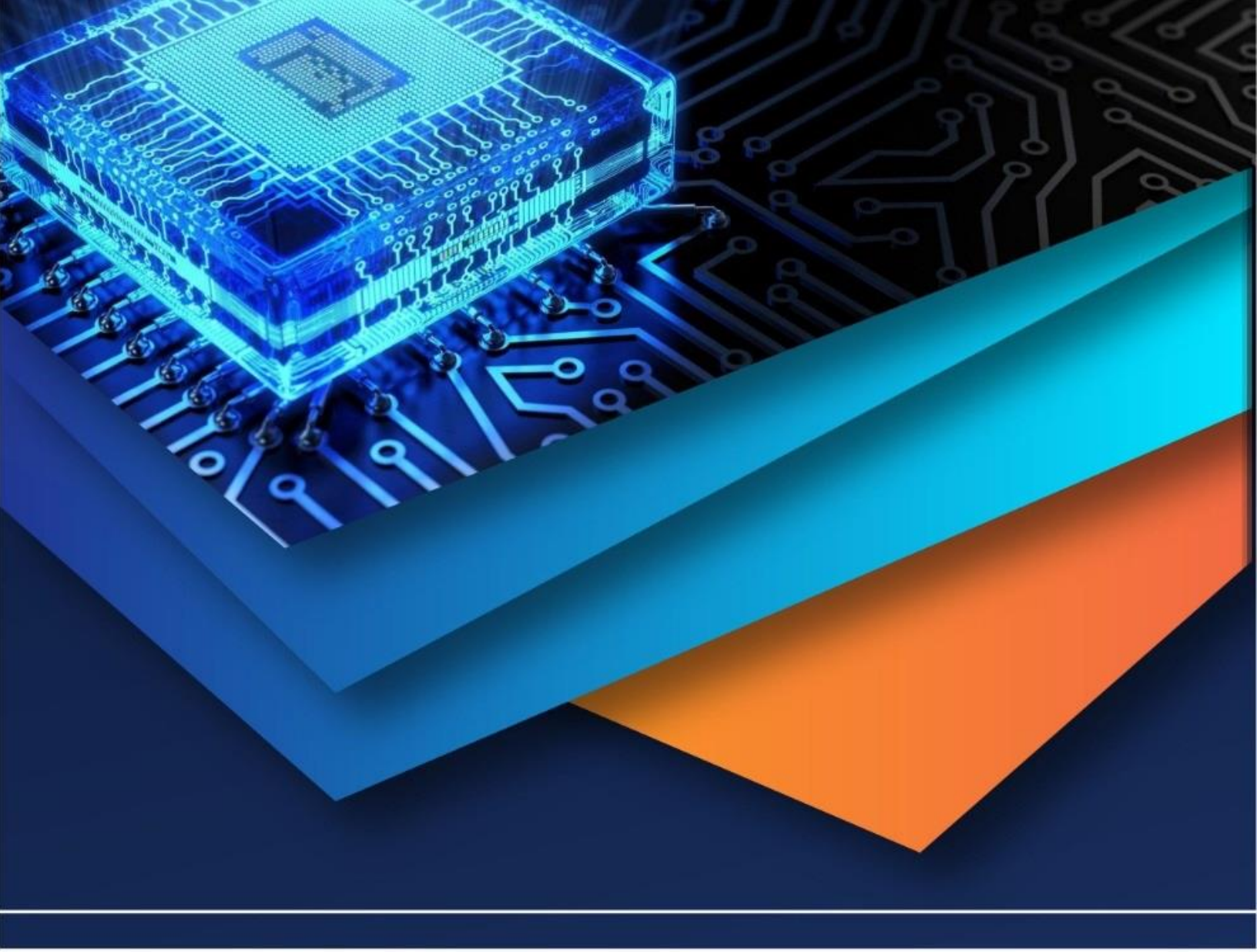
Traditional receipt pipelines have two expensive problems: servers that run idle and OCR that holds up user responses. This approach addresses both challenges. Pay-per-use compute means you're not paying for downtime. Async OCR means the upload confirms before processing even starts. Getting those two things to work together reliably is where the actual implementation effort goes.

The proposed design can be generalized to other document processing applications. Irregular document volumes with variable formats come up all over the place — expense management, invoice processing, document archiving. Stateless functions, schema-free storage, managed ML — that combination generalises to most of those problems without much reworking.

Further validation can be conducted through real-world evaluation .

REFERENCES

[1] An innovation in paper receipts: the electronic receipt management system Katherine T. Wadsworth, Michael T. Guido, John F. Griffin, and Arcan Mandil, University of Virginia, Charlottesville, VA, USA, April 23, 2010.
[2] Passive Measurement Study of AWS EC2, S3, and CloudFront, Cloud Network Measurement Dataset, Year Not Specified.
[3] S. Surkov, “Model and Method of Chunk Processing of Payload for HTTP Authorization Protocols,” International Cybersecurity Research Proceedings, 2020.
[4] Tally Solutions Pvt. Ltd., “Privacy of Business Data — Tally Case Study,” Tally Corporate Case Study Series, 2023. [Online]. Available: https://tallysolutions.com/tally/privacy-of-businessdata-tally-case-study/
[5] W. Y. Mok, “A Feasible Schema Design Strategy for Amazon DynamoDB: A Nested Normal Form Approach,” in Proc. IEEE Int. Conf. Industrial Engineering and Engineering Management (IEEM), 2020.
[6] I. Saeed, S. Baras, and H. Hajjdiab, “Security and Privacy of AWS S3 and Azure Blob Storage Services,” in Proc. IEEE Int. Conf. Computer and Communication Systems (ICCCS), 2019.
[7] Amazon Web Services (AWS), Choosing an AWS Database Service — AWS Decision Guide, AWS Whitepaper Series, 2024.
[8] R. R. Khande, S. Rajapurkar, P. Barde, H. Balsara, and A. Datkhile, “Data Security in AWS S3 Cloud Storage,” in Proc. 14th IEEE ICCCNT, 2023.
[9] Amazon Web Services (AWS), AWS Fargate or AWS Lambda? — AWS Compute Decision Guide, 2024.
[10] D. Sinha, K. Cottur, K. Bhat, C. Guruprasad, and B. Nath, “Automated Billing System Using RFID and Cloud,” in Proc. IEEE Innovations in Power and Advanced Computing Technologies (i-PACT), 2019.
[11] T. T. H. Nguyen, A. Jatowt, M. Coustaty, and A. Doucet, “Survey of Post-OCR Processing Approaches,” ACM/IEEE Joint OCR Research Survey, 2021.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY