iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.81880

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 V May 2026- Available at www.ijraset.com

Cloud-Based Multi-Channel Notification Delivery-System

Mrs.V.Veena, C. Jaiwanth David, CH.Sruthik, Y. Nachiketh Reddy
IT Department Mahatma Gandhi Institute of Technology Hyderabad, Telangana

Abstract: ***Inmodernapplications,reliablenotificationdeliveryiscriticalacrossmultiple domainssuchasfinance,healthcare, andlogistics. However,mostexistingsystemsaredomain-specificandlackflexibility,extensibility,andfault-tolerance. Thisprojectproposesageneralizednotificationplatform builtonaqueue-basedarchitecturethatensuresscalability,reliability,andmulti-channeldelivery. Thesystemfollowsa Producer-Queue-Consumer Deliverymodel,wherenotificationsarequeuedforasynchronousprocessingand guaranteeddelivery. Itsupportsmultipledeliverychannels(e.g.,email,WhatsApp)andincludessmartretrymechanismswith DeadLetterQueue(DLQ)handling. Ifanotificationfailsononechannel,thesystemcanretrywithfallbackstrategies. Additionally,the platformprovide scentralizedmonitoring, makingiteasiertotrackmessageflow,systemhealth,andfailurepoints. Thearchitectureisdesignedtobeplug-and-play,enablingseamlessintegrationwithexistingenterprisesystems. Itsmodularstructureallowsnewchannelsorrulestobeaddedwithoutimpactingthecorepipeline.Theproposedsolutionimprovesoverallsystemresilience,reduces operationaloverhead, andensuresconsistentmessagedeliveryevenunderhighload.Overall,thisplatformdemonstratesarobust,domain-agnosti capproach tobuilding scalable andobservablecommunicationinfrastructure.***

I. INTRODUCTION

The current digital world relies heavily on efficient and robust communication systems for achieving smooth functionality and creating trust among users. Notifications serve the important task of informing users about important events such as transactions, alerts, shipping and delivery information, and security alerts. For industries like finance, health care, e-commerce, and logistics, communications are an absolute necessity, with failure potentially causing financial losses, inefficiency, and even risk tothe health of the individual.

However, despite the vital importance of communications, the available notification systems are limited and specific to certain areas of use. Such systems make use of one type of communication channel, be it SMS or email, and thus are prone to failure if that channel fails to function. Moreover, such systems are not scalable, flexible, or fault-tolerant, which makes it impossible for them to accommodate the growing number of users and high frequency of events in the modern world.

Yet another drawback is the coupling of traditional systems with the application logic, thus making reuse difficult for different industries and adding to development complexity. In con section with the above-mentioned trend, there is an increasing need for a standardized and reusable notification infrastructure capable of handling multiple applications and business domains without substantial re-design and development work.

To solve these problems, this project offers the development of the Cloud-Based Multi-Channel Notification Delivery System which utilizes the advantages of modern cloud computing and distributed systems. This system will be based on the queue-based architecture (Producer - Queue - Consumer - Delivery) to ensure asynchronous execution and improve reliability by decoupling application components. Integration of several channels, including Email, WhatsApp, SMS, and Push Notifications, makes it possible to guarantee message delivery through effective routing and failover mechanisms.

The solution also benefits from incorporating cloud services for scaling purposes and performance improvements. In addition, there are built-in mechanisms for monitoring and logging which help to gain insight into the system behavior and quickly fix any issues. Microservices architecture enables extensibility, making it possible to introduce new channels or integrations.With such an arrangement, where notifications are managed from one platform, the system ensures that there is efficiency in its working. There will be ease of maintainability in the system and no redundancies. The system ensures that its performance remains the same when integrated into other applications. Moreover, such a platform paves the way for improvements in the future, as there can be AI-driven personalization and decision-making through analysis.

II. LITERATURE SURVEY

Chiang Liang Kok et al. (2025) proposed an innovative energy-efficient IoT hazard detection system with adaptive monitoring and multiple channels of alerts (Telegram and SMS), which improved reliability and reduced cost, but faced problems with burst events management and network connectivity issues. The possible improvement direction was further development of machine learning techniques to enable predictive detection [1].

A proposal of IoT-enabled healthcare monitoring system providing real-time notifications via emails, SMS, and messengers was made by Marilena Ianculescu and Victor-Ştefan Constantin (2025), which had high reliability but suffered from issues with managing large amounts of data and latencies, suggesting edge computing and predictive analytics as possible improvements [2].

A. R. Al-Ali et al. (2024)The implementation of a bridge monitoring system based on the Producer–Queue–Consumer paradigm along with notification features and retry capabilities was done, which performed well but was influenced negatively by network traffic congestion, recommending improvement in fault tolerance of edge nodes [3].

M. Barua et al. (2024) discussed a microservices architecture with high availability provided using circuit breakers, retries, and rate limiting techniques. But fine-tuning the system to increase resilience was complicated and required improvements [4].

Z. Ahmed et al. (2024) implemented an agnostic cloud messaging application using exponential backoff retry strategy and proved its performance efficiency during peak periods. At the same time, the increase in the number of retries results in more overheads in the networks, which suggests the use of ML for optimization [5].

T. Williams et al. (2024) developed an architecture based on Kubernetes, Istio, and AWS messaging services, which is event-driven and cloud-native. Despite all its benefits, configuring the service mesh may become challenging [6].

Kumar et al. (2023) introduced a Kafka-based framework for prioritizing the most crucial messages for their fast delivery. Unfortunately, maintaining the balance between delivering urgent messages and less important information poses difficulties [7].

L. Johnson et al. (2024) suggested applying AI-based scheduling methods for the management of message queues to optimize their throughput. The main limitation of the proposed system includes additional computational expenses [8].

S. Kim et al. (2024) discussed the differences between AWS Lambda and Fargate regarding the implementation of event-driven architecture and found it preferable to utilize AWS Lambda and Fargate in combination [9].

D. Nakamura et al. (2024) offered various queue optimization methods, such as batching and improved scheduling policies, which can help achieve low latency. Nevertheless, inter-region latency remained problematic [10].

V. Desai et al. (2024) proposed a multiple provider notification approach with intelligent failover that increased reliability and cost-efficiency. However, to maintain real-time information of providers, monitoring had to continue [11].

Ashwin Chavan (2024) researched fault-tolerant event-driven systems with an emphasis on multi-channel routing and failover mechanisms to ensure reliability of such systems. The study found that system complexity had increased and recommended automation in resilient systems testing as the way forward [12].

Raghav Agarwal (2025) examined event-driven architecture regarding multi-provider notification routes and failover approaches. While improvements were achieved in terms of reliability, alert synchronization within a distributed environment became difficult, requiring consistency-based event streaming in future work [13].

Sharma et al. (2025) recommended a migration from monolithic applications to event-driven microservices with multi-channel routing. While modularity had been improved, compatibility with legacy systems became problematic [14].

III. METHODOLOGY

1) EventGeneration

The entire process starts with the creation of a notification request triggered by a particular event from the client app, which may be an eCommerce site, banking software, or even healthcare application. Such an event can be related to the confirmation of an order, alert regarding transactions, or an appointment reminder. The notification request carries all the vital data required like the user data, type of message, template ID, and preferred delivery channel.

2) .Queue-BasedProcessing

Upon the creation of the notification request, it gets put into an SQS queue. The queue serves as an intermediary to act as a buffer for the producer and processing services to facilitate asynchronous communication. As a result, there will be no overload on any of the components even with many requests coming into the system. Even if the processing services are not available at a certain point, all the messages will still be safe within the queue until they get processed.

3) NotificationProcessing

The consumer service that runs either on AWS Lambda or via a backend service such as Django regularly polls the queue to fetch and process messages. The service is responsible for validating the incoming message data and fetching the corresponding template from the database. It then combines the dynamic variables such as the name of the user, order ID, and other variables to create the final message.

4) Multi-ChannelRouting

The system selects the most appropriate mode of delivery for the message after processing it. The choice of the delivery mode will depend on factors such as user preference, type of message, and how the system is configured to function. The possible modes of delivery are email, WhatsApp, SMS, and push notifications.

5) NotificationDeliveryviaExternalServices

After choosing the communication channel, the notification message gets transmitted via third-party services. In this way, for instance, email notifications get transmitted through AWS SES, WhatsApp messages get transmitted via the Meta API, Push notifications via Firebase Cloud Messaging, and SMS notifications through the Gateway API. The channels work separately from each other using their own service provider.

6) RetryandFallbackMechanism

To deliver notifications in a reliable manner, the system uses retry and fallback mechanisms. In case a failure occurs during delivery using the main channel, then the system tries to send the message again with exponential back-off mechanism to prevent overloading of the external systems. Should this not work out, the system falls back to a secondary channel, which ensures that eventually the message will be delivered successfully.

7) MonitoringandLogging

This system has been designed with full monitoring and logging capabilities that ensure logging for all activities performed by notifications from creation to finalization. Some of the information logged includes messages of delivery, time stamps, errors, and responses from the service providers. Monitoring provides real-time data on the status of the system.

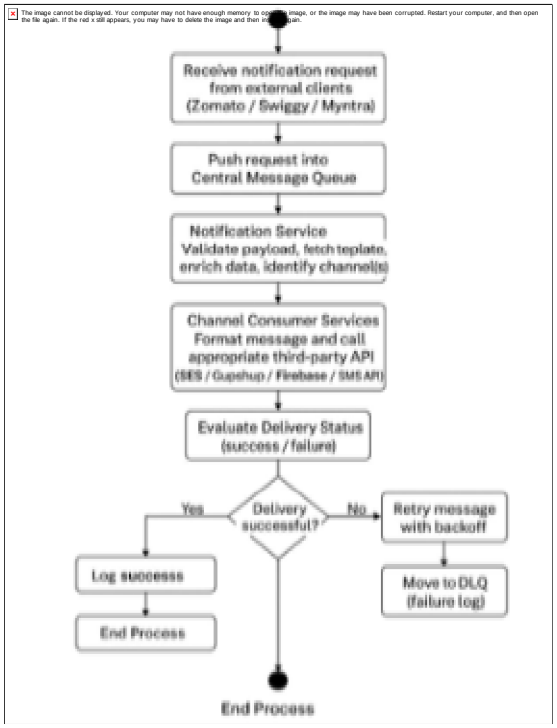


Fig.1:WorkflowDiagram.

*Fig1.*The flow starts with the receipt of notifications from third-party application clients and placing them in a queue for asynchronous processing. The notification service processes the message and validates the payload. It picks up the correct template, adds data to it, and decides on the appropriate delivery channel. The message is passed on to channel-specific consumer services, where it gets formatted and delivered to external providers like email, SMS, WhatsApp, and push notifications. Once delivery is done, the system assesses the delivery status of the message. In case the message was delivered successfully, the outcome is stored, and the process terminates. However, in the case of failed delivery, the system attempts multiple re-deliveries using an exponential backoff approach. The failed messages are placed in a DLQ for further processing.

The method ensures the reliability and scalability of the system because of component separation and asynchronous communication. Moreover, it eliminates the problem of losing messages in case of system crashes or high loads on the system. The usage of several channels makes the delivery of messages more likely to succeed. Finally, logging and monitoring allow tracking the system's performance and rapid debugging. All these features ensure successful notification delivery through the workflow.

A. ProposedArchitecture

The suggested system uses a systematic and scalable architecture for multi-channel notification delivery. Notifications are sent from different external sources, such as Zomato, Swiggy, and Myntra, with clients sending structured payloads that contain information like order alerts and status messages into the application. These requests are received through the API gateway and are pushed into the central message queue, which is used as a reliability layer to ensure that there will be no data loss when the load is high and facilitate asynchronous processing.

Notifications Service, which works as the producer layer, pulls messages from the central queue, and preprocesses them. In the process, notifications are validated, the channels they should go through are determined, and relevant metadata is added to the payload. Once the preprocessing stage is complete, notifications are placed in channel-specific queues according to the delivery channels (email, WhatsApp, SMS, push notifications).

At the consumer level, dedicated notifications providers retrieve these messages from the respective channel queues and prepare them in the specific channel format, such as HTML email, WhatsApp message template, or even mobile app push notifications payload. After being formatted appropriately, these notifications are routed via external third-party providers for each of the respective channels. The system provides plug-and-play integration that permits addition of new providers with little effort.

In terms of reliability, the system makes use of a comprehensive mechanism of handling failures as well as re-trying failed notifications. In case of a failed delivery attempt, the message would go back to its respective queue for subsequent re-routing, using an exponential delay strategy to avoid server overload and repeated delivery failures. Also included is a Template Configuration Database where reusable templates could be created. This enables dynamic generation of templates using variable information such as names, order details, or status.

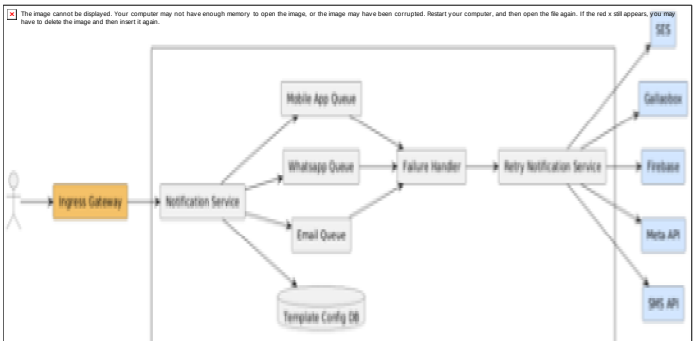


Fig.2:ProposedArchitectureDiagram

B. ProposedWork

This project aims at developing a Cloud-Based Multi-Channel Notification Delivery System for facilitating reliable and scalable communication over different domains. This has been implemented using a Queue-based, Event-driven Architecture to make asynchronous processing possible.

In this work, the major aim of designing a notification system is that it must have a generalized form which can be integrated with various types of clients such as ecommerce sites, logistics services, healthcare services and financial systems. The notification requests in this system are accepted via API calls and processed based on queue implementation.

Multi-channel communication capability is provided using this system through Email, WhatsApp, SMS, and Push Notifications. This system can identify the optimal delivery channel based on certain configuration criteria.Moreover, the framework has a built-in fallback system, which means that if one of the channels does not work, the message will be automatically transferred to another channel to ensure its delivery.

For the sake of reliability, the proposed research also includes a mechanism that uses retry strategies with exponential backoff and Dead Letter Queues (DLQ), where failed messages can be processed further. Cloud computing through AWS SQS, AWS SES, and serverless computing makes it easy for the system to expand without being costly and accessible. Furthermore, there is a template management system to make it easier for dynamic messages to be created from templates. The logging and monitoring functions ensure that message delivery and performance can be analyzed.

The objective of this study is to develop a notification system that is flexible and reliable, regardless of domain specificity.

IV. OUTPUT

System Interface Performance

The notification manager software went through functional testing and interactions by the user with the entire system. It comprises multiple modules such as Notifications Templates, Multi-Channel Delivery (Email, WhatsApp, and Push), Credentials Management, and Notification Logs. This helps users manage communication processes from one central place.

The application shows responsive structure in terms of user interaction, which makes navigating through modules easy.

User requests go through backend APIs that validate payloads, parse the template, and send the message over to the proper communication channel. It also has capabilities for integrating third-party services like cloud email, push notification, and messaging services.

API authentication and secure token-based access are implemented to enhance operational security of the platform. As per the evaluation, the software manages to achieve message delivery, use templates in messages, and log them into logs easily. Moreover, its service-oriented architecture enables future scalability.

Web Interface

To improve usability and administrative efficiency, a web-based interface was implemented for the platform. The interface allows users to create and manage templates, trigger notifications, and monitor delivery outcomes from a single environment.

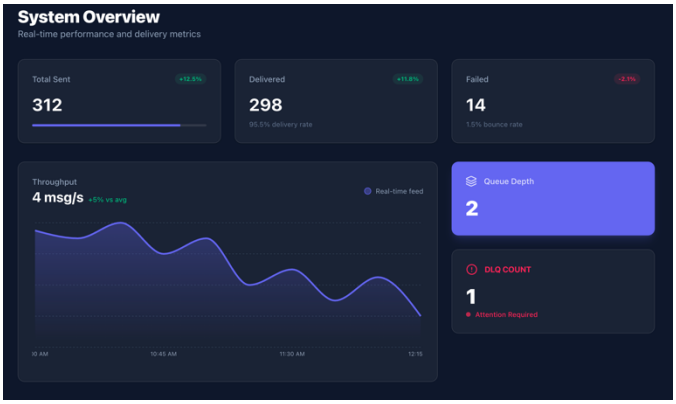


Fig. 3. Dashboard

The Dashboard is the main gateway to the platform. It gives direct access to important functions, including template customization, send notification process, and delivery statistics. The dashboard design ensures easy navigation and efficiency for users to find important functions quickly. It boosts work efficiency by minimizing time spent on notifications management.

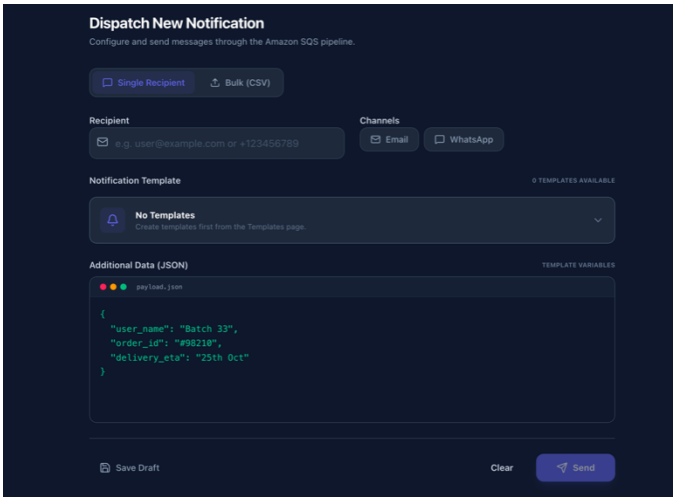


Fig. 4. Notification Send Interface

pipeline provider. This component helps in maintaining consistency in the message format and prevents any manual errors during notification creation. The Send Interface allows users to create notification requests based on the channel, template, and recipient information. Validation of mandatory fields is done, along with processing of dynamic template variables.

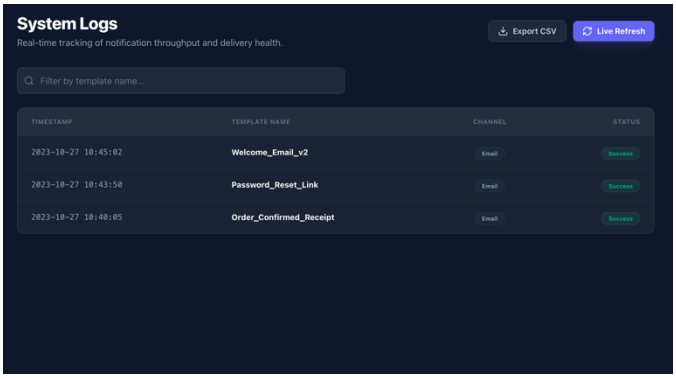


Fig. 5. Notification Logs and Updates

The Logs feature gives near real-time insight into notification operations, such as status tracking, timestamp recording, and outcomes at the level of each channel. It aids users in identifying failures, successful delivery attempts, and previous history of communication operations. This approach facilitates the reliability analysis of the system as well as helps debug errors.

V. CONCLUSION

The described Cloud-Based Multi-Channel Notification Delivery System is a viable solution that addresses the mentioned problems of existing notification systems as well as pro-vides a proper and reliable approach to sending messages to modern devices. Due to the asynchronous architecture that involves a queue and events, the system successfully processes huge loads of requests while minimizing the risk of message loss.

The involvement of multiple communication channels such as Email, WhatsApp, SMS, and Push notifications increases reliability and ensures high user satisfaction, since the use of alternative communication channels guarantees delivery of a message. Various techniques used to increase reliability of the system ensure successful delivery of messages.

Additionally, using cloud-based technology ensures greater flexibility of the system, since the size of the system depends on industry. Template management, logging, and monitoring features make the system more usable.

In general, this design presents an adaptable and universal framework that can be easily incorporated within diverse business settings. This system provides a robust platform for further improvements, including intelligence-based routing, optimization through analytics, and AI-driven personalization, thus contributing to the evolution of advanced and versatile notification systems.

VI. FUTURE ENHANCEMENT

The proposed notification delivery mechanism serves as a robust framework for providing reliable and scalable notification delivery services; however, there are various enhancements that could be added to improve the efficiency of this system. An important aspect of this is to include intelligent routing capabilities based on AI and machine learning. The system will be able to analyze the past performance of users, delivery rates, and channel performance to identify the most efficient communication channel.

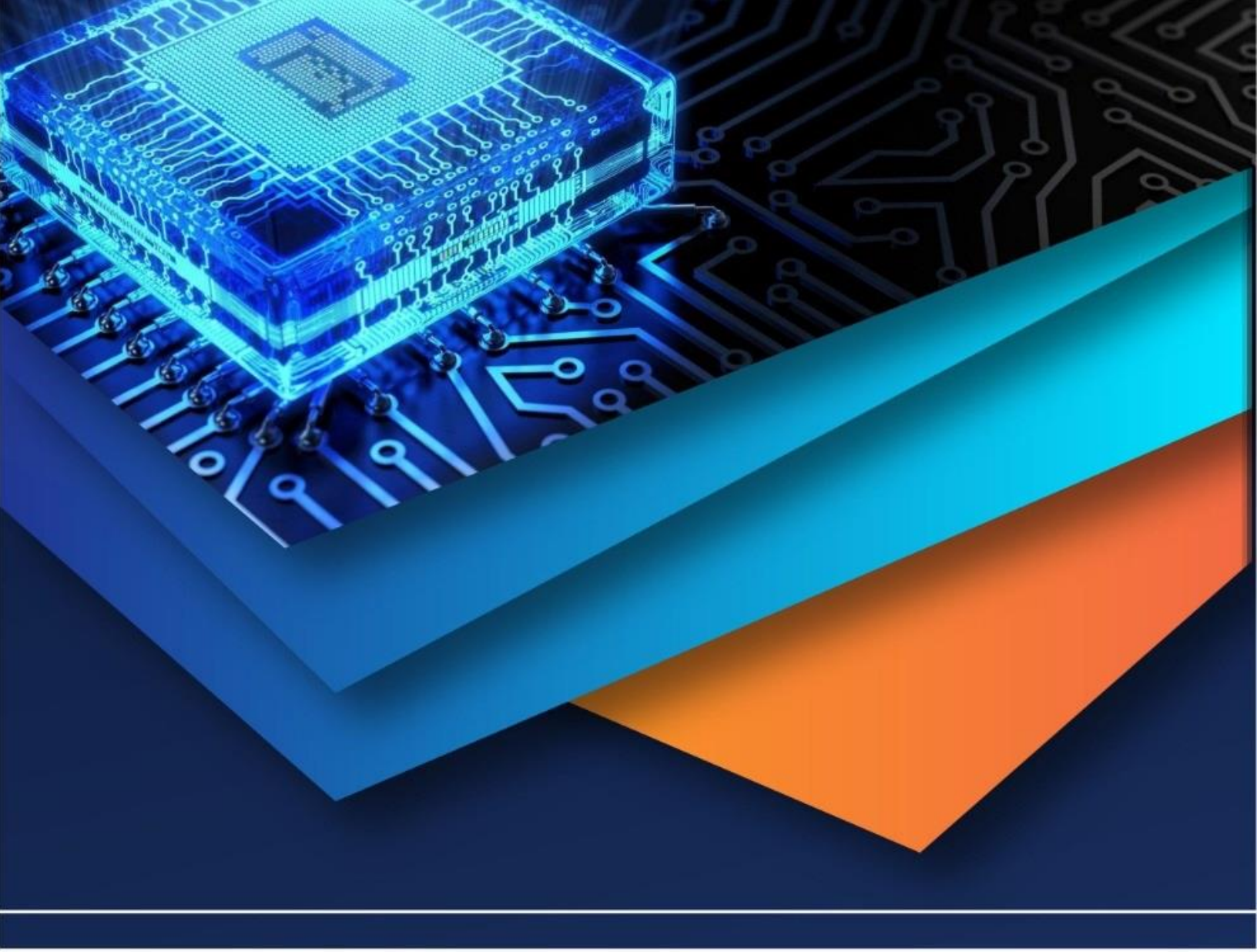
Another enhancement would be to incorporate other communication channels, such as phone calls, chatbots, and in-app messaging among others. Real-time analytics and reporting mechanisms could also be added to give businesses deeper insights about delivery metrics, user engagement, and overall system performance. The prediction of notifications, whereby notifications are sent before an event occurs as opposed to after, should also be considered.

Finally, adding security measures to ensure that sensitive information remains confidential is very important. Some of these measures may include end-to-end encryption, access control, and data protection policies.

The design of the architecture may further be enhanced by implementing the concepts of multi-region deployment and geo-distributed queues for decreasing latency and increasing availability from various geographic locations. In addition, the use of cost-effective and high-performance solutions through scalable solutions and load balancing will help improve the system. In conclusion, these changes would allow the system to becomean intelligent, secure, and highly scalable notification system tomeet industry demands.

REFERENCES

[1] ChiangLiangKokJovanBowenHeng,YitYanKoh,&TeeHuiTeo.(2025). Energy,Cost,andResource-EfficientIoTHazardDetectionSystemwithAdaptiveMonitoring.
[2] MarilenaIanculescu&Victor-ȘtefanConstantin.(2025). EnhancingConnectedHealthEcosystemsThroughIoT-EnabledMonitoringTechnologies.
[3] Sensors(MDPI)
[4] A.R.Al-Ali,SalwaBeheiry,AhmadAlnabulsi,ShahedObaid,NoorMansoor,NadaOdeh,&AlaaeldinMostafa.(2024). AnIoT-BasedRoadBridgeHealth Monitoring andWarning System.Sensors(MDPI).
[5] A.R.Al-Ali,SalwaBeheiry,AhmadAlnabulsi,ShahedObaid,NoorMansoor,NadaOdeh,&AlaaeldinMostafa.(2024). AnIoT-BasedRoadBridgeHealthMonitoringand Warning System.Sensors(MDPI).
[6] A.R.Al-Ali,SalwaBeheiry,AhmadAlnabulsi,ShahedObaid,NoorMansoor,NadaOdeh,&AlaaeldinMostafa.(2024). AnIoT-BasedRoadBridgeHealthMonitoringand WarningSystem.Sensors(MDPI).
[7] T.Williams,H.Martinez,&J.Brown.(2024). Cloud-NativeArchitecturePatternsforEvent-DrivenMicroserviceswithAWSServices.IEEEMicro.
[8] Kumar,Verma,&N.Singh.(2023). EfficientMessageQueuePrioritizationinKafkaforCriticalSystems.TheResearchJournal(TRJ)
[9] L.Johnson,C.Davis,&E.Miller.(2024). DesignandSchedulingofanAI-BasedQueueingSystem.arXivPreprint
[10] S.Kim,J.Park,&M.Cho.(2024). PerformanceBenchmarkingofCloudServerlessFunctions(AWSLambdavsFargate).IJISAE—InternationalJournal ofIntelligentSystemsand ApplicationsinEngineering
[11] D.Nakamura,K.Tanaka,&M.Suzuki.(2024). DistributedMessageQueueOptimizationforLow-LatencyNotificationDeliveryinCloudInfrastructure .IEEET ransactionso nParallel andDistributedSystems
[12] V.Desai,A.Iyer,&R.Menon.(2024). AdaptiveFailoverStrategiesandChannelRoutinginMulti-ProviderCloudNotification Systems.Journal ofCloudComputing(Springer)
[13] A.Chavan.(2024). Fault-TolerantEvent-DrivenSystems:TechniquesandBestPractices.JournalofEngineeringandAppliedSciencesTechnology
[14] ARaghavAgarwal.(2025). Event-DrivenArchitecturesforCloudStorageSystems.
[15] JETIR—JournalofEmergingTechnologiesandInnovativeResearch
[16] A.Chavan.(2024). Fault-TolerantEvent-DrivenSystems:TechniquesandBestPractices.Sharmaetal.(2025). Event-DrivenArchitecturesforMicroservices: AFrameworkforMigratingfrom MonolithicSystems.IJSAT—InternationalJournalofScientificandAdvancedTechnology



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY