International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I5P101

Volume 72 Issue 5, 1-10, May 2024
© 2024 Seventh Sense Research Group®

Original Article

E-Delivery Microservices Based Multi-Channel Communications Delivery Framework

Amol Gote

Solutions Architect, New Jersey, USA.

Corresponding Author : aamolgotewrites@gmail.com

Received: 01 March 2024 Revised: 05 April 2024 Accepted: 21 April 2024 Published: 10 May 2024

Abstract *- Effective client communication is paramount for building strong customer relationships and driving business success in the rapidly evolving startup landscape. However, managing and delivering communications across multiple channels can be complex and resource-intensive. This paper presents a comprehensive microservices-based E-Delivery framework to streamline and optimize client communication processes. The E-Delivery framework comprises three core logical components: Managing Templates, E-Delivery Service, and Core Notification Service. This modular architecture facilitates scalability and clear separation of concerns, allowing each component to scale independently based on communication demands. The Framework leverages the power of technologies such as Apache Kafka, AWS Simple Notification Service (SNS), and AWS Simple Email Service (SES) to ensure efficient and reliable message delivery. A key novelty of the E-Delivery framework lies in its template-driven approach, enabling consistent and personalized communication across channels while adhering to brand guidelines. The paper explores the real-world implementation of the Framework at iCreditWorks, a fintech company operating in a multilingual environment, showcasing its effectiveness in delivering diverse communications to clients. The E-Delivery framework offers numerous advantages, including scalability through event-driven architecture, consistency through templates, separation of concerns, flexibility with Kafka streams, and abstraction of third-party service dependencies. By addressing the complexities of client communication, this Framework empowers startups to enhance customer engagement, build stronger relationships, and ultimately drive business growth.*

Keywords *- Client Communications, Microservices, Communication Framework, Template Driven Messaging, Apache Kafka.*

1. Introduction

In the dynamic and competitive FinTech industry, organizations face numerous challenges in acquiring and retaining customers. Effective client communication addresses these challenges, fosters strong relationships, and drives business growth. However, managing and delivering communications across multiple channels can be daunting, often requiring significant resources and expertise.

This research identifies a critical gap in existing industry solutions, specifically, the absence of an integrated, microservices-based framework that supports scalable, resilient services responsible for sending client communications tailored for FinTech startups.

Most existing implementations focus primarily on documentation for specific cloud services delivering communications to specific channels and do not provide a framework or design approach that supports multi-channel communication (SMS, email, push notifications, and letters) for FinTech clients.

This research paper presents a comprehensive solution – the E-Delivery microservices-based framework – designed to simplify and streamline the process of delivering client communications for organizations. The Framework leverages a modular architecture, incorporating three core logical components: Managing Templates, E-Delivery Service, and Core Notification Service.

By delving into the intricacies of this Framework, this paper aims to provide organizations with a structured approach to efficiently handle client communications, encompassing various channels such as SMS, email, push notifications, and even physical letters. The Framework's logical separation of components enables scalability and promotes clear separation of concerns, ensuring each component can evolve independently to meet growing communication demands.

This research paper explores the Framework's architecture, implementation, and real-world use case at iCreditWorks, a fintech company. It seeks to equip organizations with a powerful tool to enhance customer engagement, build stronger relationships, and ultimately drive business success in the competitive FinTech landscape.

By delving into the intricacies of this Framework, this paper aims to equip organizations with a structured approach to efficiently handle client communications, enhancing customer engagement and driving business success in the competitive FinTech landscape.

2. Literature Review

Effective client communication has long been recognized as critical in building strong customer relationships and driving business success, especially in the startup ecosystem. Client communication in the financial industry has transitioned from traditional face-to-face interactions, paper mail, and phone calls to digital platforms, including emails and instant messaging. Evolution has been driven by the need for faster, more efficient, and cost-effective communication solutions that align with changing consumer behaviors and technological advancements. As technology has advanced, so has the complexity of managing these communications, especially with the rise of global and multilingual customer bases and multiple channels. However, managing and delivering communications across multiple channels can be complex and resource-intensive, necessitating innovative solutions.

In recent years, microservices architecture has gained significant traction as a means of developing scalable and maintainable applications (Fowler & Lewis, 2014; Newman, 2015). By breaking down monolithic applications into smaller, independent services, microservices facilitate easier management, deployment, and scalability of individual components (Wolff, 2016). This approach aligns well with the need for scalable and flexible communication systems in the rapidly evolving startup landscape.

Several studies have explored the application of microservices in various domains, including e-commerce, healthcare, and finance. However, research on leveraging microservices for sending client communications is relatively limited.

Event-driven architectures, often facilitated by technologies like Apache Kafka, have been widely adopted in microservices-based systems to enable asynchronous communication and decoupling of services (Kleppmann, 2017; Narkhede et al., 2017). This approach can be particularly beneficial in communication systems, where message delivery may involve multiple steps and third-party integrations. The E-Delivery framework incorporates an event-driven architecture that supports the asynchronous delivery of communications across multiple channels and ensures that these communications are fault-tolerant and reliably ordered.

Template-driven messaging has been explored in various contexts, such as email marketing and customer relationship management (CRM) systems. By leveraging templates, organizations can maintain consistency and personalization in their communications while adhering to brand guidelines and ensuring compliance. Framework’s use of dynamic templates across multiple communication channels, including physical letters and not just digital messages, sets it apart from existing studies. It allows for high personalization and brand consistency, essential for customer engagement and satisfaction in FinTech services.

While existing literature provides insights into microservices architecture, event-driven systems, and template-driven messaging, a comprehensive framework that combines these elements to address the specific challenges of client communication in the startup ecosystem is needed. The E-Delivery Microservices-Based Framework aims to fill this gap, offering a structured approach tailored to the needs of startups operating in diverse industries and serving clients across multiple delivery channels like SMS, email, and push notifications.

3. Methodology

This research focuses on the E-Delivery Microservices-Based Framework, a comprehensive solution designed to streamline client communication processes for startups. At its core, the E-Delivery framework comprises three fundamental components, each playing a critical role in the communication process:

Managing Templates: This component involves creating, maintaining, and managing communication templates that combine static text with dynamic variables for personalization.

E-Delivery Service: The E-Delivery Service acts as a communication hub, exposing endpoints for other business domain services to initiate client communication.

Core Notification Service: The Core Notification Service integrates with third-party cloud services to ensure the delivery of communications across various channels, including SMS, email, and push notifications.

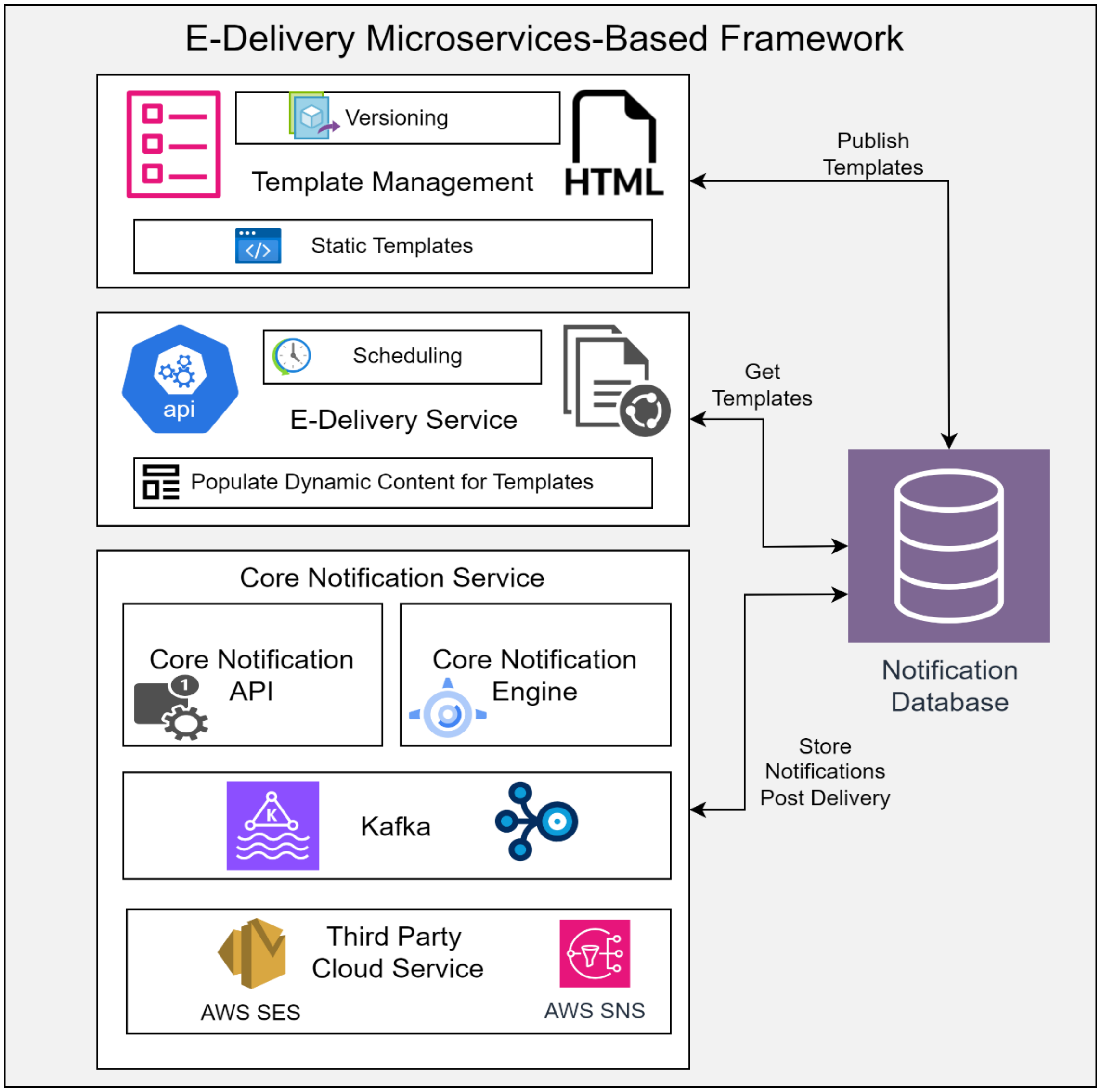


Fig. 1 E-Delivery logical components

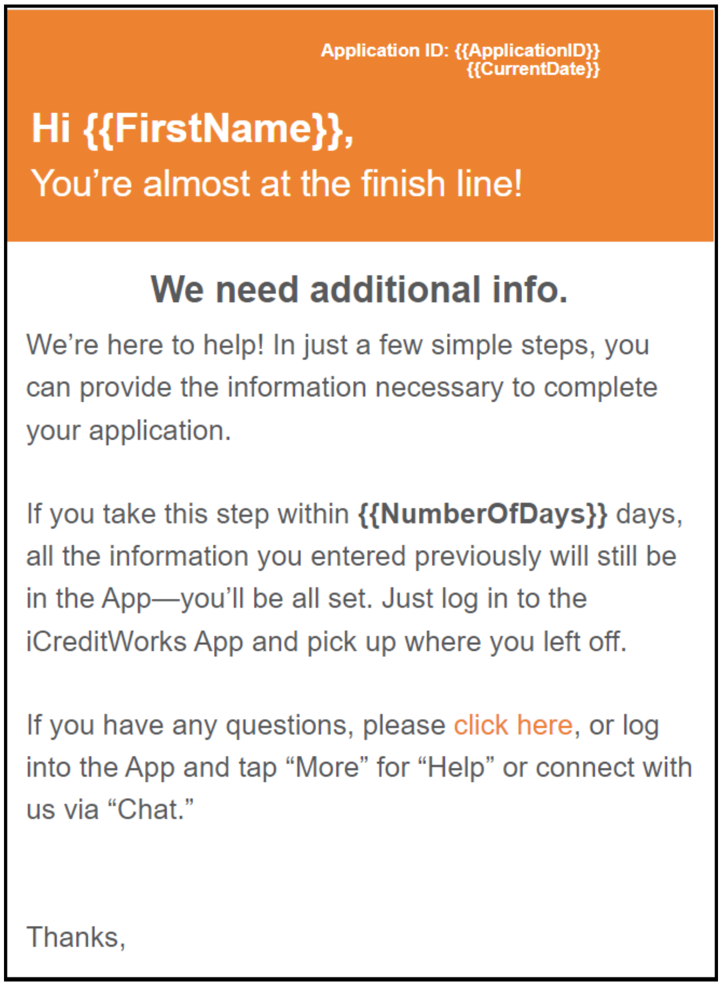
3.1. Managing Templates

Effective communication relies on well-designed message templates. The concept of 'Comm' or Communication is introduced, with each being uniquely identified by a code such as ICC-APP01. Within a Comm, multiple communication channels are catered to, including PUSH, SMS, EMAIL, and Letter, each requiring its dedicated template. These templates can support various languages (locales).

These templates are essential for our messaging strategy, combining static text with dynamic variables for personalization. Plain text templates are used for SMS and Push, and HTML templates are used for email and letter communications. The dedicated content team creates, maintains, and publishes these templates, ensuring they convey information while aligning with the brand identity. The content team generates and maintains high-quality content that aligns with the organization's branding and communication strategies. This includes web content, marketing collateral, digital and print brochures, email campaigns, social media posts, etc.

These templates have a specific structure comprising static text content and dynamic content template variables, identified by two opening braces and two closing braces, e.g., *{{CustomerFirstName}}, {{CustomerLastName}}*. They undergo a life cycle with stages like Draft, Approved, and Published. Additionally, they are versioned, with each publication incrementing the version. Templates are initially developed, tested, and approved in lower development and testing environments and then moved to the production environment.

Below is the sample communication template:


Application ID: {{ApplicationID}}
{{CurrentDate}}

Hi {{FirstName}},
You're almost at the finish line!

We need additional info.

We're here to help! In just a few simple steps, you can provide the information necessary to complete your application.

If you take this step within **{{NumberOfDays}}** days, all the information you entered previously will still be in the App—you'll be all set. Just log in to the iCreditWorks App and pick up where you left off.

If you have any questions, please click here, or log into the App and tap "More" for "Help" or connect with us via "Chat."

Thanks,

Fig. 2 Communication template

3.2. E-Delivery Service

The second core component, the E-Delivery Service, plays a pivotal role in the Framework, serving as the communication hub for various business domain services. It exposes an endpoint, allowing other services to initiate client communication. Send communications endpoint:

POST {{baseUrl}}/api/edelivery/v1/send

Request body:

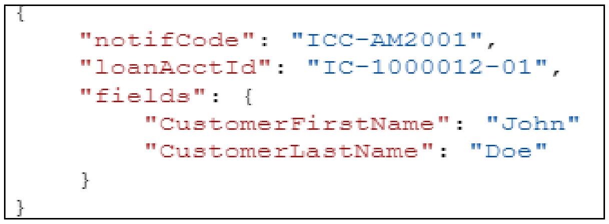

```
{
    "notifCode": "ICC-AM2001",
    "loanAcctId": "IC-1000012-01",
    "fields": {
        "CustomerFirstName": "John"
        "CustomerLastName": "Doe"
    }
}
```

Fig. 3 Request body Send API

notifCode: This serves as an identifier, signaling the type of communication the invoking business domain service intends to send. It helps route the request to the appropriate communication template.

loanAcctId: This unique loan account identifier links the communication to the specific customer's loan account. This identifier can differ for different lines of business.

fields: A map of key-value pairs representing dynamic content to be injected into the communication template. Keys correspond to dynamic variables within the template, such as {{CustomerFirstName}} and {{CustomerLastName}}.

Here is the sequence diagram for things performed as part of the e-delivery.

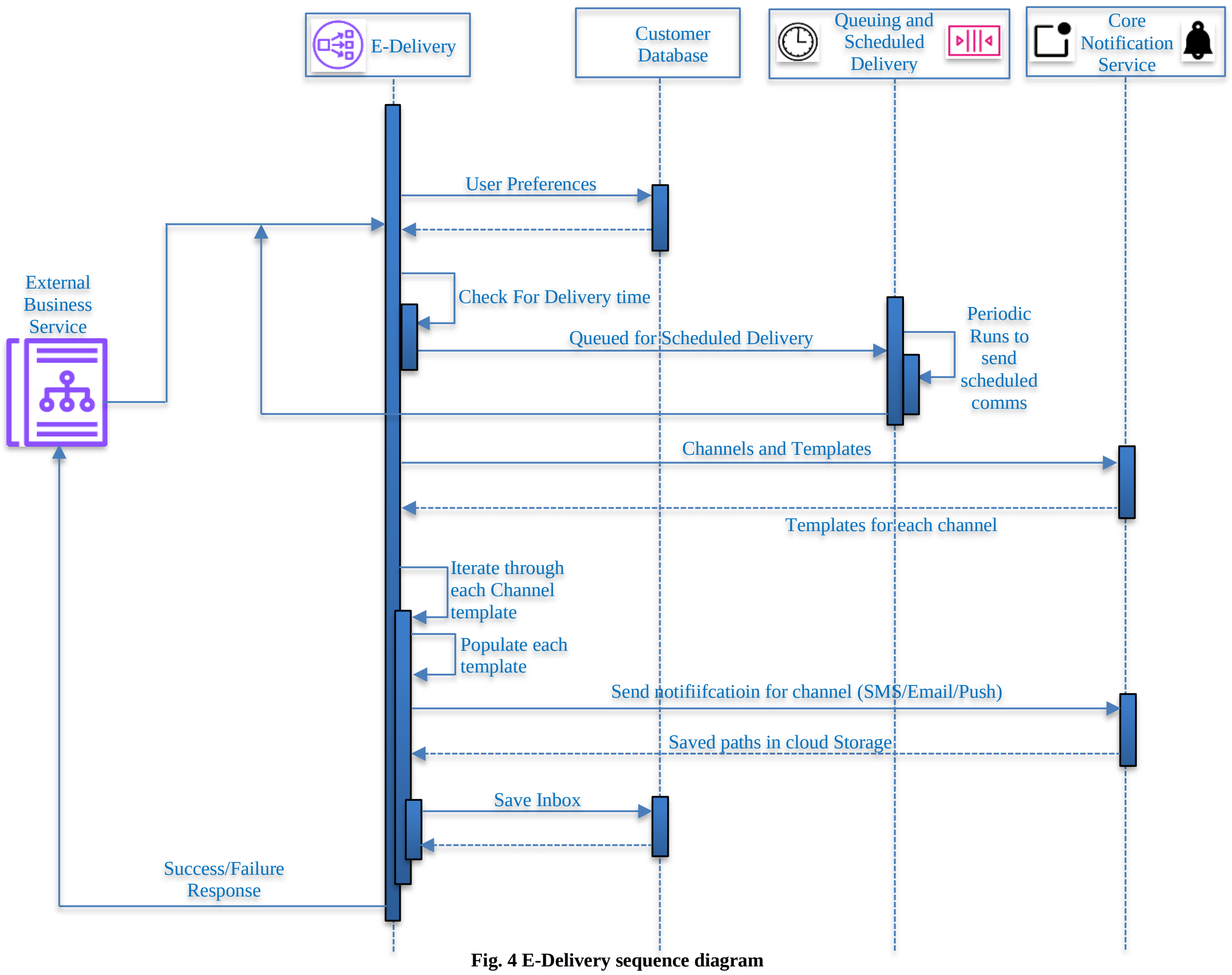

E-Delivery
Customer Database
Queuing and Scheduled Delivery
Core Notification Service
External Business Service
User Preferences
Check For Delivery time
Queued for Scheduled Delivery
Periodic Runs to send scheduled comms
Channels and Templates
Templates for each channel
Iterate through each Channel template
Populate each template
Send notifiifcatioin for channel (SMS/Email/Push)
Saved paths in cloud Storage
Save Inbox
Success/Failure Response

Fig. 4 E-Delivery sequence diagram

Each communication has specific attributes, including a designated delivery time window. If the current time falls outside the window, the client communication is queued for later delivery.

User preferences for communications and locale settings are retrieved based on a key identifier, such as the loan account identifier. User preferences include choices regarding the delivery of SMS and Email notifications, allowing users to decide whether they want both delivered or only one.

The service fetches all channels (PUSH/SMS/EMAIL/LETTER/INBOX) for the given communication code and loads individual templates into memory for each channel.

The delivery of the communications process begins with push notifications and continues with SMS, email, letter, and inbox messages.

For each channel, the service constructs a message template that needs to be sent to the end user.

Templates feature dynamic templated variables, divided into two types: common variables shared across all templates, such as social media links or contact emails (referred to as global fields), and custom fields sent as part of the request payload. The service replaces templated variables with values from the request payload for custom fields and uses mapped fields from the configuration database for global variables, ensuring the completion of individual templates.

Once the message is entirely constructed, the service calls the notification service to send communication. The notification service's payload includes an attribute indicating the type of communication (Email/SMS/Push).

Inbox messages are private messages visible in mobile applications. For the Inbox channel, the message generated from the filled template is saved in AWS S3 (Amazon Web Services - Simple Storage Service), and its path is stored in the database, associated with the business identifier and comm code.

A similar pattern is followed in the Letter channel. The message generated from the filled template is an HTML document converted into a PDF format. These generated letter PDFs are stored in AWS S3 and subsequently printed and shipped.

This structured process ensures that messages are efficiently delivered and customized to user preferences and locales, enhancing the overall client communication experience.

E-Delivery service also exposes an additional endpoint, "send-direct." This "send-direct" endpoint is like the previous one, with the critical difference that it sends the communication immediately and does not queue it. It is essential for consuming business application services to ensure they specify the target channel (SMS/Email) and destination when using this endpoint.

POST {{baseUrl}}/api/edelivery/v1/send-direct

```
{
    "notifCode": "ICC-APP-PEND-CR-01",
    "loanAcctId": "IC-2112139-01",
    "emailAddress": "johndoe@yopmail.com",
    "phone": "9999999999",
    "fields": {
        "queueName": "Pending Credit"
    }
}
```

Fig. 5 Send direct API request

3.3. Core Notification Service

Lastly, the Core Notification Service plays a crucial role in ensuring the delivery of communications across different channels to end users, consisting primarily of two components:

- Core Notification API
- Core Notification Engine

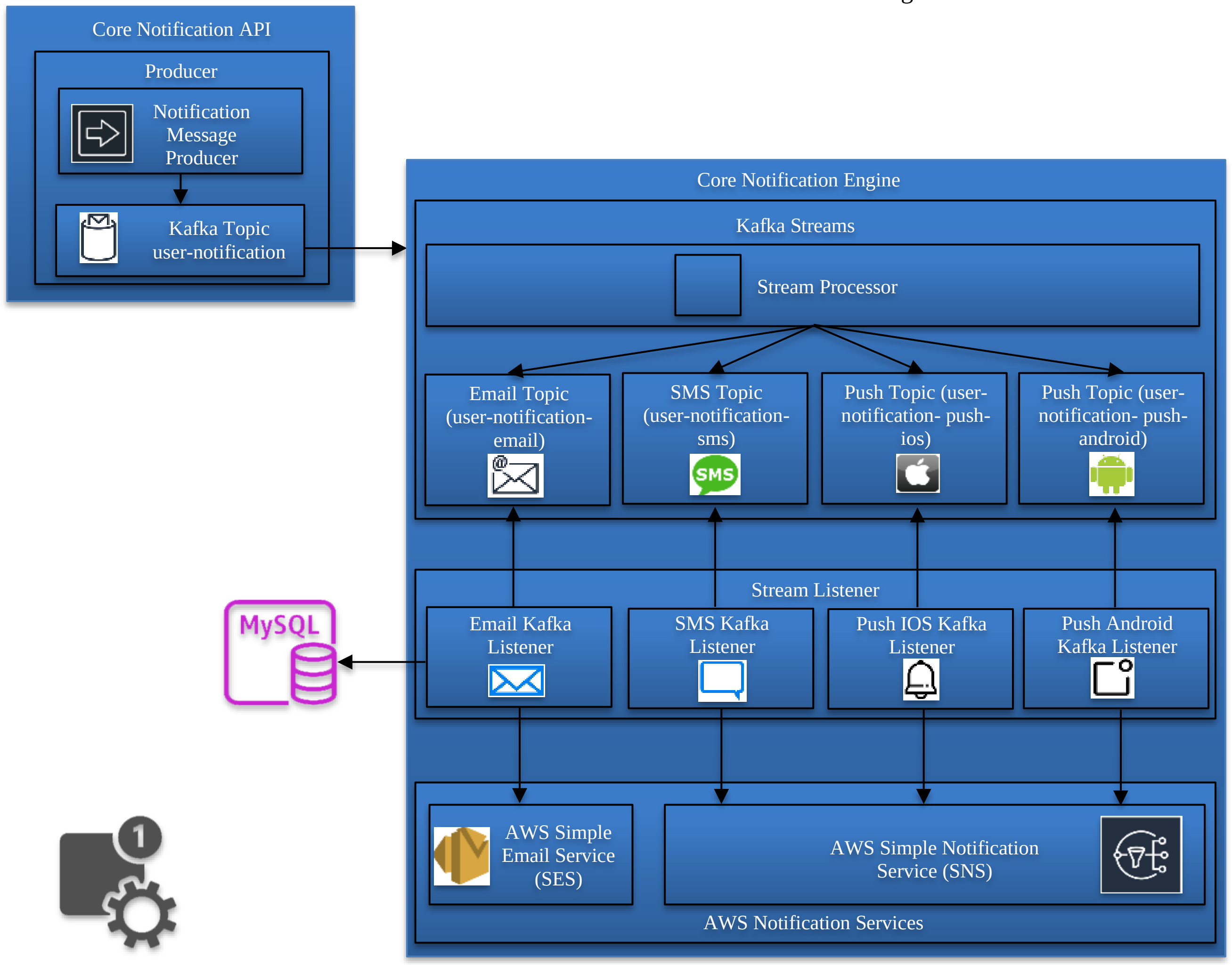
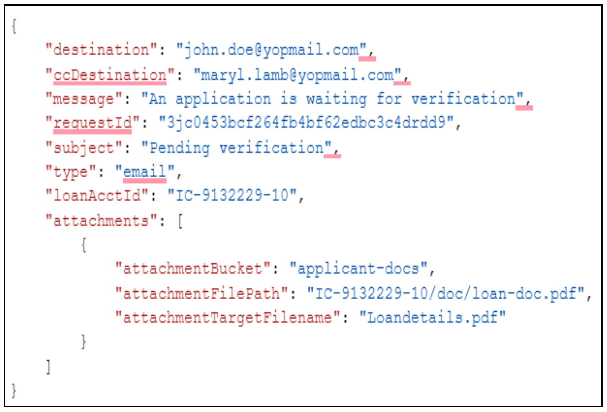

Core Notification API
Producer
Notification Message Producer
Kafka Topic user-notification
Core Notification Engine
Kafka Streams
Stream Processor
Email Topic (user-notification-email)
SMS Topic (user-notification-sms)
Push Topic (user-notification- push-ios)
Push Topic (user-notification- push-android)
SMS
Stream Listener
Email Kafka Listener
SMS Kafka Listener
Push IOS Kafka Listener
Push Android Kafka Listener
MySQL
AWS Simple Email Service (SES)
AWS Simple Notification Service (SNS)
AWS Notification Services
1

Fig. 6 Core notification service

3.3.1. Core Notification API

This component exposes an endpoint responsible for initiating the final delivery of various communications. Sample requests for email and SMS are as follows:

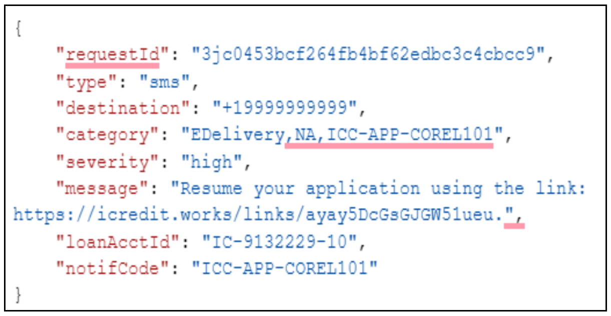
Email Request:



```
{
    "destination": "john.doe@yopmail.com",
    "ccDestination": "maryl.lamb@yopmail.com",
    "message": "An application is waiting for verification",
    "requestId": "3jc0453bcf264fb4bf62edbc3c4drdd9",
    "subject": "Pending verification",
    "type": "email",
    "loanAcctId": "IC-9132229-10",
    "attachments": [
        {
            "attachmentBucket": "applicant-docs",
            "attachmentFilePath": "IC-9132229-10/doc/loan-doc.pdf",
            "attachmentTargetFilename": "Loandetails.pdf"
        }
    ]
}
```

Fig. 7 Email request

SMS Request:



```
{
    "requestId": "3jc0453bcf264fb4bf62edbc3c4cbcc9",
    "type": "sms",
    "destination": "+19999999999",
    "category": "EDelivery,NA,ICC-APP-COREL101",
    "severity": "high",
    "message": "Resume your application using the link:
https://icredit.works/links/ayay5DcGsGJGW51ueu.",
    "loanAcctId": "IC-9132229-10",
    "notifCode": "ICC-APP-COREL101"
}
```

Fig. 8 SMS request

The API performs validation checks on the received request payload and then queues the request with a Kafka message in a topic called "user-notification".

Attachments in the email request are not received as binary files but as AWS S3 document paths. These files are retrieved from AWS S3 and sent as attachments, with the filename specified as "attachmentTargetFilename."

The "message" attribute contains the communication body, which is a lengthy HTML body in the case of email. Since the API pushes the message to the Kafka topic, its response time is less than 100 milliseconds.

3.3.2. Core Notification Engine

This is where the messages delivered to different channels fan out and get delivered to the end user through the appropriate channel. It comprises of

- Stream Processor
- Stream listeners to each channel
- Channel-specific (SMS/EMAIL/PUSH) third-party cloud services.

Stream Processor

This component subscribes to the "user-notification" topic and receives messages, fanning them to different topics dedicated to each communication channel.

Stream Listeners to each Channel

For all the topics mentioned above in the stream processor, there are dedicated stream listeners; these listeners are:

1. Email Stream Listener: This listener receives messages from the "user-notification-email" topic and uses AWS SES (Amazon Web Services - Simple Email Service) to send emails to end users.
2. SMS Stream Listener: This listener receives messages from the "user-notification-sms" topic and sends text messages using AWS SNS (Amazon Web Services—Simple Notification Service).
3. Push Notification for iOS Devices (APN Listener): This listener receives messages from the "user-notification-push-apn" topic and uses AWS SNS to send push notifications to Apple devices.
4. Push Notification for Android Devices: This listener receives messages from the "user-notification-push-android" topic and uses AWS SNS to send push notifications to Android devices.

Channel-specific (SMS/EMAIL/PUSH) third-party cloud services.

The Framework leverages AWS SDK APIs to send communications for all the AWS cloud services, such as SNS and SES.

- AWS SNS will send push notifications to Android and iOS devices.
- For SMS, use AWS SNS.
- AWS SES to send emails.

Once messages are successfully delivered using third-party cloud services (in this case, AWS), all the above listeners store the request ID and all related message attributes in the database for an audit trail.

This ensures that a comprehensive record of all communication activities is maintained for tracking and verification purposes.

Once a notification is delivered, an AWS Lambda-based implementation updates the delivery status and handles bounced messages. However, that is a separate topic or article, so it is out of the scope of this paper.

4. Real-World Use Case: iCreditWorks

iCreditWorks, a fintech company, uses the E-Delivery framework in its day-to-day operations.

4.1. Efficiency Across Languages and Channels

iCreditWorks operates in a multilingual environment, serving clients in both English and Spanish. With over 350 communication codes designed to meet specific business needs, the company has seamlessly integrated communication across five channels: PUSH, SMS, Email, Inbox, and Letter.

4.2. Delivery Metrics

iCreditWorks has successfully delivered a substantial number of communications.

1. Notifications will be delivered till Feb 2024. It includes SMS, Email, and push notifications. Physical letters have been excluded as they are sent only if all digital means of communication fail. The delivery rate has been 99.99%.

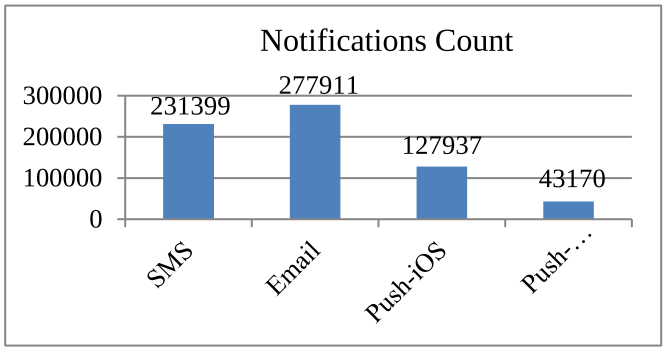


Fig. 9 Notifications delivered to Date.

Notification Channel	Count #
SMS	231399
Email	277911
Push Notification - iOS	127937
Push Notification - Android	43170

2. 30 Days API Hits – For Jan to Feb 2024

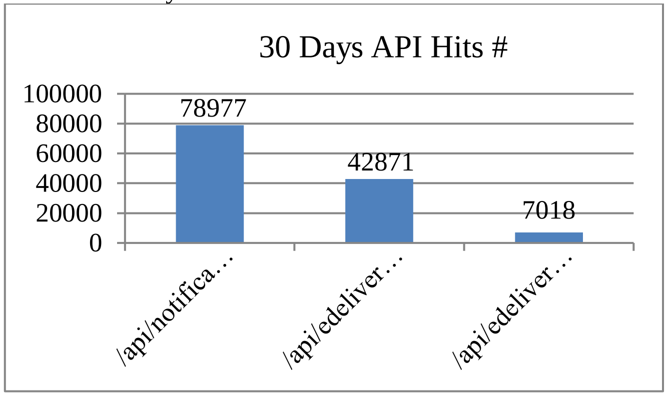


Fig. 10 Thirty days API Hits

E-Delivery API	Hit Count #
/api/notification/v1/	78977
/api/edelivery/v1/send	42871
/api/edelivery/v1/send-direct	7018

So, whether a fintech startup or any business looking to streamline client engagement, the E-Delivery framework provides a reliable and efficient approach. The below section covers its advantages more thoroughly.

4.3. Availability

The E-Delivery framework has achieved a service availability of 99.9%. The Framework's architecture features loosely coupled components, which enhance the system's resilience and fault tolerance. This design allows individual service components to fail and recover without disrupting the entire system.

5. Advantages and Comparative Analysis

5.1. Scalability and Efficiency in the E-Delivery Framework

The E-Delivery framework is designed to deliver communications across multiple channels by leveraging Apache Kafka's robust capabilities alongside a strategically segmented microservice architecture. This combination allows the Framework to outperform traditional monolithic systems and other contemporary microservice-based solutions, primarily in scalability and reliability.

The logical separation of components within the E-Delivery framework allows individual microservices to scale independently, and Kafka plays a pivotal role in enabling this scalability. Kafka's distributed and event-driven architecture enables seamless scaling of message processing. It provides a highly reliable and fault-tolerant platform for handling communication between components. It ensures that as the system's load increases, Kafka can efficiently distribute messages across multiple consumers, allowing each microservice to scale horizontally by adding more processing nodes.

This dynamic scaling capability ensures that the E-Delivery system can effectively handle growing communication volumes, maintaining high performance and responsiveness, all while minimizing downtime and service disruptions.

5.1.1. Performance Metrics

When tested underload, the E-Delivery system demonstrates superior performance and can handle 300 concurrent requests without degradation in response times; this can be scaled up if there is a business need. The response time for the service with 300 concurrent requests was around 500 milliseconds.

5.1.2. Fault Tolerance

Kafka's built-in fault tolerance mechanisms, such as data replication and partitioning, ensure the system maintains high availability and data consistency even in node failures.

5.2. Template Management

Using templates within the E-Delivery framework offers several advantages for startups, including consistency in communication across channels, efficient message generation through reuse, personalized customer engagement, adherence to brand guidelines, version control, testing, centralized management, and creating an audit trail for compliance and verification purposes.

Unlike traditional systems, which often require separate tools and processes for each communication channel, leading to potential inconsistencies and errors, the integrated template system in E-Delivery ensures uniformity and reduces the operational burden.

5.3. Separation of Concerns

Traditional communication systems often combine data management, message delivery, and template management into a monolithic structure.

A key strength of the E-Delivery framework is its well-defined separation of concerns. The E-Delivery service focuses on managing user data associated with specific lines of business for the startup, ensuring a tailored approach to communication. The Core Notification service is dedicated solely to the efficient delivery of communications and remains agnostic to the intricacies of individual business lines. Furthermore, the Template Management component resembles a typical content management system but offers a more structured and organized approach, streamlining the creation and maintenance of communication templates. This clear separation optimizes the efficiency and clarity of each component's responsibilities within the Framework.

5.4. Resilience Through Kafka

Utilizing Kafka streams and topics within the E-Delivery framework provides the flexibility to handle disruptions in third-party services effectively. In case of issues with the underlying third-party services, Kafka messages can be seamlessly redirected to an alternative queue, allowing for message reprocessing once the third-party service becomes healthy again. This built-in resilience ensures that critical client communications are not lost and can be delivered with minimal disruption, contributing to the robustness of the communication system.

Conventional communication systems often rely on a linear approach to message handling, where disruptions in a third-party service can lead to significant communication delays or losses.

5.5. Decoupling from Third-Party Dependencies

The Core Notification service abstracts the dependency on third-party services like AWS SNS and SES. This abstraction allows for selecting equivalent services in alternative cloud environments, such as Azure Communication Services and Azure Notification Hub.

Compared to conventional systems that are often tightly integrated with a single provider, this architecture significantly reduces the risk of vendor lock-in. It allows users to leverage the best capabilities of various cloud services while maintaining a consistent and reliable service delivery model.

6. Limitations

Implementing the E-Delivery microservices-based framework may involve a learning curve. Customization to fit specific business requirements requires time and effort.

Relying on cloud services like AWS SNS and AWS SES can be advantageous. However, it also means the E-Delivery system depends on these services' availability and performance, and downtime can impact operations.

Operating and maintaining the infrastructure, microservices, and third-party services come with associated costs.

7. Future Work

While the E-Delivery Microservices-Based Framework provides a robust solution for managing multi-channel client communications, several avenues for further development can enhance its functionality, scalability, and adaptability.

Integration with Additional Cloud Services: The current Framework primarily utilizes Amazon Web Services (AWS) for its backend services like SNS and SES. Future iterations could explore compatibility with other cloud platforms, such as Microsoft Azure or Google Cloud Platform. This would diversify the Framework's dependency on cloud providers, allow users to leverage different cloud-specific features, and potentially reduce costs.

8. Conclusion

In conclusion, the E-Delivery framework presents a structured framework for efficiently managing and delivering client communications. This Framework offers several key advantages, including logical component separation and scalability, as well as the utilization of technologies like Kafka, AWS SNS, and AWS SES. By streamlining communication processes and ensuring personalized and timely interactions with clients, the E-Delivery framework contributes to building strong customer relationships. The clear separation of responsibilities among its components, including Template Management and the Core Notification service, optimizes efficiency and clarity within the Framework. Moreover, the use of Kafka streams and topics adds a layer of flexibility and resilience to the system, ensuring that critical client communications are consistently delivered, even in the face of disruptions in third-party services.

While the E-Delivery microservices-based framework may come with a learning curve and customization efforts, its benefits far outweigh the initial challenges. Startups can leverage the Framework to enhance communication through different channels and improve client engagement.

In today's dynamic business landscape, where client communication plays a pivotal role in success, the E-Delivery framework offers a solution that empowers one to navigate the complexities of client interactions efficiently and effectively.

References

[1] James Lewis, and Martin Fowler, Microservices, Martinfowler, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html.
[2] Sam Newman, *Building Microservices: Designing Fine-Grained Systems*, O'Reilly Media, pp. 1-280, 2015. [Google Scholar] [Publisher Link]
[3] Eberhard Wolff, *Microservices: Flexible Software Architecture*, Addison-Wesley Professional, pp. 1-332, 2016. [Google Scholar] [Publisher Link]
[4] Martin Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*, O'Reilly Media, pp. 1-616, 2017. [Google Scholar] [Publisher Link]
[5] Neha Narkhede, Gwen Shapira, and Todd Palino, *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*, O'Reilly Media, pp. 1-322, 2017. [Google Scholar] [Publisher Link]
[6] Amazon Simple Notification Service Documentation, Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/sns/
[7] Amazon Simple Email Service Documentation, Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/ses/
[8] Amazon Simple Storage Service Documentation, Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/s3/