iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** VI **Month of publication:** June 2022

DOI: https://doi.org/10.22214/ijraset.2022.45143

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 10 Issue VI June 2022- Available at www.ijraset.com

Centralized Framework to Send Notifications from Multiple Tools and Services on Multiple Messaging Platforms

Nehal N Shet[1], Darshan Narayana Murthy[2], Pranava Bhat[3], Dr. Rajashekara Murthy S.[4]
[1, 4]*R.V. College of Engineering, Bangalore, India*
[2, 3]*Cloudera, Bangalore, India*

Abstract: ***One of the reasons for advancement in the field of technology and IT is that a lot of tech companies constantly innovate and come up with new ideas and products. Apart from the mainstream products and services, organizations also have a lot of tools and services to support the software development life cycle of their product. Different engineering teams would have different sets of tools and services that are used internally within an organization. Every patch of code goes through the phases of the software development life cycle which takes a considerable amount of time. Developers have to proactively monitor the status of the runs, jobs that validates their patch. This can block them and waste a lot of time.***
Notification is one of the means to inform users. Notifying developers at the right time can save their time, they can triage issues if any, or merge the code to production or take appropriate actions based on the status. With the growing number of tasks, runs and jobs being triggered, a notification feature will add value to the service users. But developing notification modules for all the services individually can be inefficient. A better solution would be to have a centralized notification framework that is capable of sending notifications from multiple services. This paper presents a generic notification framework that can be integrated with any kind of tools or services and will be able to send notifications on any messaging platform. The framework also addresses aspects like scalability, fault tolerance, reliability, maintenance and works in real-time.
Keywords: Notification, Scalability, Reliable, Real-Time, Customizable, Flexible, Microservices, REST API, Core, Slack, Email, Queue, DLQ Processor.

I. INTRODUCTION

Present technology is so advanced that people get most of their work done with the help of portable computers, hand held devices. Within a tech organization these technologies and devices play an important role. Many organizations proactively indulge themselves in innovation to come up with new projects and products. And the organizations will also have some internal tools and services. Teams which support infrastructure for development, quality engineering teams, release engineering teams, Cloud admins and many more own the tools and services that support the mainstream product development. When developing a mainstream product, even a small piece of code has to undergo all the phases and it will be certified only if it passes all the phases. But for a piece of code to undergo all these phases takes a considerable amount of time. A scenario wherein a developer develops some patch and expects it to work fine so starts working on other tasks and parallely monitors the previous patch, If the previous patch fails and already a new patch which is dependent on previous patch is created, it will be a waste of time if the developer is unaware about the failure. Sometimes some patches might get stuck at a phase due to some internal issues. Developers have to proactively monitor the status in order to perform appropriate operations. This becomes a blocker and wastes a lot of their time. If somehow developers get to know the status at the right time without proactively monitoring, then they can perform appropriate steps at the right time.
Sending out notifications to the developers at the right time and letting them know the status of the runs, jobs or tasks might be a very useful functionality to have for a tool. In case of failures, they can be triaged at the right time or in case of success the product can be deployed to production as early as possible. Notifications can be sent to communicate critical events like service/instances going down. We can integrate the monitoring framework for getting timely notification for critical events. The current trend in software development is to develop microservices. These microservices will be loosely coupled, scaled and maintained independent of others. In such cases it becomes inefficient to have a notification feature for each microservice. The efficient way is to have a generic Centralized Notification Framework which can be integrated with any tools and services. A plus point would be to have the framework send notifications on multiple messaging platforms.

Some of the previous work related to notification systems includes a python Flask based package named Flask-Notifications which is currently a pre-release pypi package developed by the authors of [1] which is an extension for python flask applications that send out email notifications. This package makes use of redis and celery brokers. Many other efforts were made to develop a notification system specific to some use cases.

The authors of [2] have done a survey on result alert systems. The alert system is designed specifically for an educational institution use case where the students, faculty and staff get alerted about any events, updates on the institution website via SMS and emails. Similar to the previous survey, the authors of [3] have developed a notification system specifically to send out android notifications about the academic information of a student.

This feature is helpful for the students, teachers and staff of the educational institution. Though this notification system is built for a specific institution, they claim the system to be extendable for a wide range of users. A similar use case with a different approach has been addressed by the authors of [4], this notification system is developed specifically to be used by the lecturers and students at Universiti Teknologi Petronas.

The authors of [5] developed an electronic notification system to notify influenza as a part of national surveillance in New Zealand.

The authors of [6] developed a notification framework that can be used by any microservices and send out notifications. The framework was a spring boot framework developed in Java. But the notifications were pushed to the browser to a custom frontend web application.

Authors of [7] presented a notification system that can highly handle scale and send out notifications on various devices like smartphones, personal computers, android tv, smart watches and many more.

Building a notification system that can be highly reliable and scalable and work in real time is not an easy task. The requirements keep on changing as per the requirements and as the process of development progresses. Some aspects like the presentation of the message of the notification, avoiding spams, and understanding the requirements of the users should be taken care of. The article [8] covers all these aspects and describes the means of designing a scalable notification framework. Various articles and resources provide various technologies and methodologies to make use of for developing notification frameworks, one more example includes the notification system design by oracle[9].

With most of the organizations moving towards a microservice architecture style of software development, the notification framework proposed in this paper will be based on a similar approach, all the sub components will be loosely coupled and independent of the other components. This framework will make use of RabbitMQ message broker as an asynchronous message queue and some of the third party APIs like Slack API in case of slack, smtplib in case of emails, Twilio for sending SMS and other APIs as per the user requirement. The notification framework proposed in this paper is containerized and deployed on the kubernetes cluster. The results obtained, efficiency of the framework, scalability and other aspects will be discussed with context to docker and kubernetes.

II. NEED FOR A CENTRALIZED SYSTEM

Nowadays most of the software's is developed using microservice architecture. A large project will have many smaller components that are loosely coupled with each other. And if the tasks and processes take a considerable amount of time without the users being aware about the status, then due to this gap there can be many blockers.

If the service owners include a notification module as a part of their services individually, then there will be a lot of duplicate code and maintaining them will be an issue. Also if the organization decides to use a new messaging platform for communication, then all the notification modules should be updated accordingly which can cause a lot of overhead. On the other hand, having an independent generic notification framework that can be made use by any kind of tools and services would be a better and efficient option.

Centralized notification system is different from Centralized notification framework. A centralized notification system can be made use by many tools and services but it will not have an option to extend to other messaging platforms, whereas a centralized notification framework provides the flexibility to extend it to send notification over any messaging platforms.

A Centralized notification framework provides an easy interface for all service owners to integrate their service with it and also be generic to get extended to use any messaging platforms will solve a lot of overhead. It will also be a great plus point to have a dead letter queue processor to handle failures while sending the notifications. Having a monitoring feature in this framework will also help to analyze the metrics on notifications being sent.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VI June 2022- Available at www.ijraset.com

III. DESIGN

The following section explains the design of the proposed notification framework. The framework has three major pillars, that is the REST API, Core and the Notifiers. The external data stores used are MongoDB to store user's

A. Rest API

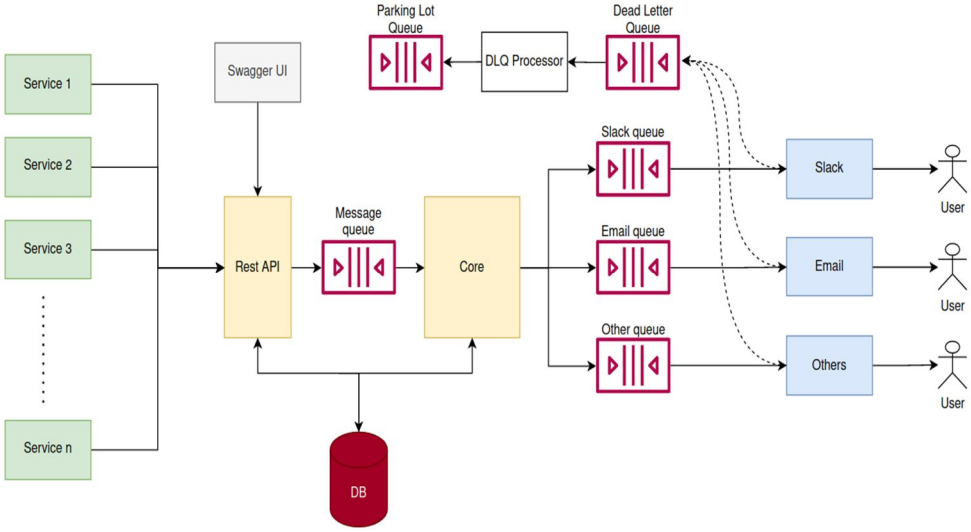


Fig 1. High level design diagram

```
{
    serviceName: str,
    title: str,
    message: str,
    recipients: List[str]
}
```

Fig 2. Simple schema defined for message

Notification preferences and RabbitMQ queues to act as a message broker and to avoid the loss of any message. The REST API will provide endpoints and an easy interface for the tools and services to get integrated with the framework. Even the users will be able to set their preferences using these endpoints. The REST shall contain a POST endpoint and have a minimal schema defined for it. The schema shall be very simple as present in Fig 2. and is designed by keeping in mind to provide an easy interface. This schema can be modified based on requirements. Tools and services should have the logic to populate this payload. Just by populating this minimal payload and making a POST call, any service shall be able to send out notifications. The service which wants to send out notifications shall define a couple of functions to construct this message and make a curl or POST call to this endpoint. The remaining endpoints will allow users to set their preferences. Usually any notification system would ask the users to subscribe to the notification service in order to start receiving notifications, but this framework works the other way. Users need not do anything from their end to start receiving notifications, they will automatically start receiving notifications when the specific notifiers are implemented. These user preference endpoints can be used by users to unsubscribe from any services on any messaging platforms. Users will be able to mute/unmute notifications from any services on any messaging platform. For example a user can mute notifications from service1 on slack and continue to receive notifications from other services, service1 shall still be able to send notifications on other messaging platforms. It would be good to have a frontend for this REST API but Swagger docs or Postman can also be used to invoke the REST endpoints.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VI June 2022- Available at www.ijraset.com

B. Core

Core shall have the main logic to check for user notifications preferences. Services might not be aware about users preference for the notification being sent, so it might pass that user's name in the recipient list. Core shall handle these cases. Core also shall have logic to segregate the recipients for various messaging platforms. Since the aim is to provide an easy interface, the API shall have just one field where the services can populate the user names of all the recipients and core shall intelligently segregate it. For example if the framework supports Slack and Emails, the service shall only provide the usernames of individuals, the entire email address in case of email groups and entire name of channel in case of slack channels. A differentiator can be set to differentiate between various messaging platforms. Like prefixing the channel name with # in case of slack. Specifying the entire email address in case of emails, having a @ prefix in case of discord etc. The core can be designed and modified as per the requirements.

C. Notifiers

Notifiers shall contain the core logic of sending out notifications. Notifier can be designed as an interface which can be implemented by various other notifiers specific to the messaging platforms. For example in the case of slack, the framework owners have to create an app, generate keys and add that app to their workspace. For email, an email account has to be created and made use. Similarly even other messaging platforms provide APIs and tokens to allow sending notifications programmatically. And on the framework side, the specific notifiers logic shall be implemented to make use of those tokens and send the notifications programmatically.

D. Other Components

Apart from these pillars, the framework shall also have a dead letter queue processor. If due to some internal issues in any of the messaging platforms and failures while sending the notifications, the DLQ processor shall have the logic to keep track of these failed notifications and try to resend them after some time.

RabbitMQ queues are used as message brokers for communication between the framework pillars. These queues will make sure that there is no loss of messages due to high traffic. Message queue acts as a link between the API and the Core. And the notifier queues act as a link between Core and Notifiers. All the unsent notifications shall be stored in the dead letter queue and the DLQ processor will handle those notifications. Even if the DLQ processor won't be able to send the notifications after a fixed number of retries, the messages are stored in the parking lot queue. THe framework owners can manually intervene the messages.

MongoDB is used to store users notification preferences which will be used by the core which segregates the recipient list.

IV. ADVANTAGES

This framework provides many advantages over having a centralized notification system or notification modules separately for all the tools and services individually.

1) *Easy Interface:* This framework provides an easy interface for any service to get integrated. Just by making a curl or post call will be able to send notifications from that service to the respective user base.
2) *Extendable:* The framework also provides an easy interface to get extended to send notifications over any messaging platform. The framework owners shall be able to define and customize their own notifier logic as per the messaging platforms.
3) *Customisable:* The notification payload schema, the differentiator for different messaging platforms, logic to send notifications to the users can be easily customisable based on the requirements.
4) *Use Case:* With this framework, multiple services will be able to send notifications over multiple messaging platforms. Usually this is not possible in most cases. Every service will have a different logic to send notifications over different messaging platforms.
5) *Scalability:* This framework is highly scalable. Deploying multiple instances of this framework will handle restrictions from any third party apis if any. For example slack api has some restrictions over the usage of its api, but this framework allows multiple bots to handle those restrictions too.
6) *Real Time:* The notifications sent by this framework are real time. Some processing time might be required while sending the message for the first time based on the implementation.
7) *Reliable:* By having multiple instances of each pillar of this framework, and by using rabbitmq statefulset, there is no loss of message during the entire flow of the message.

V. CONCLUSION

With the increased use case and requirement of a notifying feature, this framework reduces complexity and overhead of having multiple notifier modules. Since a framework like this can integrate with multiple notifiers (slack, email) receivers can get notified on their preferred platform and can quickly rectify things to avoid escalation. But this is not the perfect solution, rather a flexible solution. Based on the requirements the design and implementation can be modified for better usability and use cases. The framework has been deployed in Cloudera and is integrated with multiple services. Currently on an average the framework is sending over 240 notifications per hour.

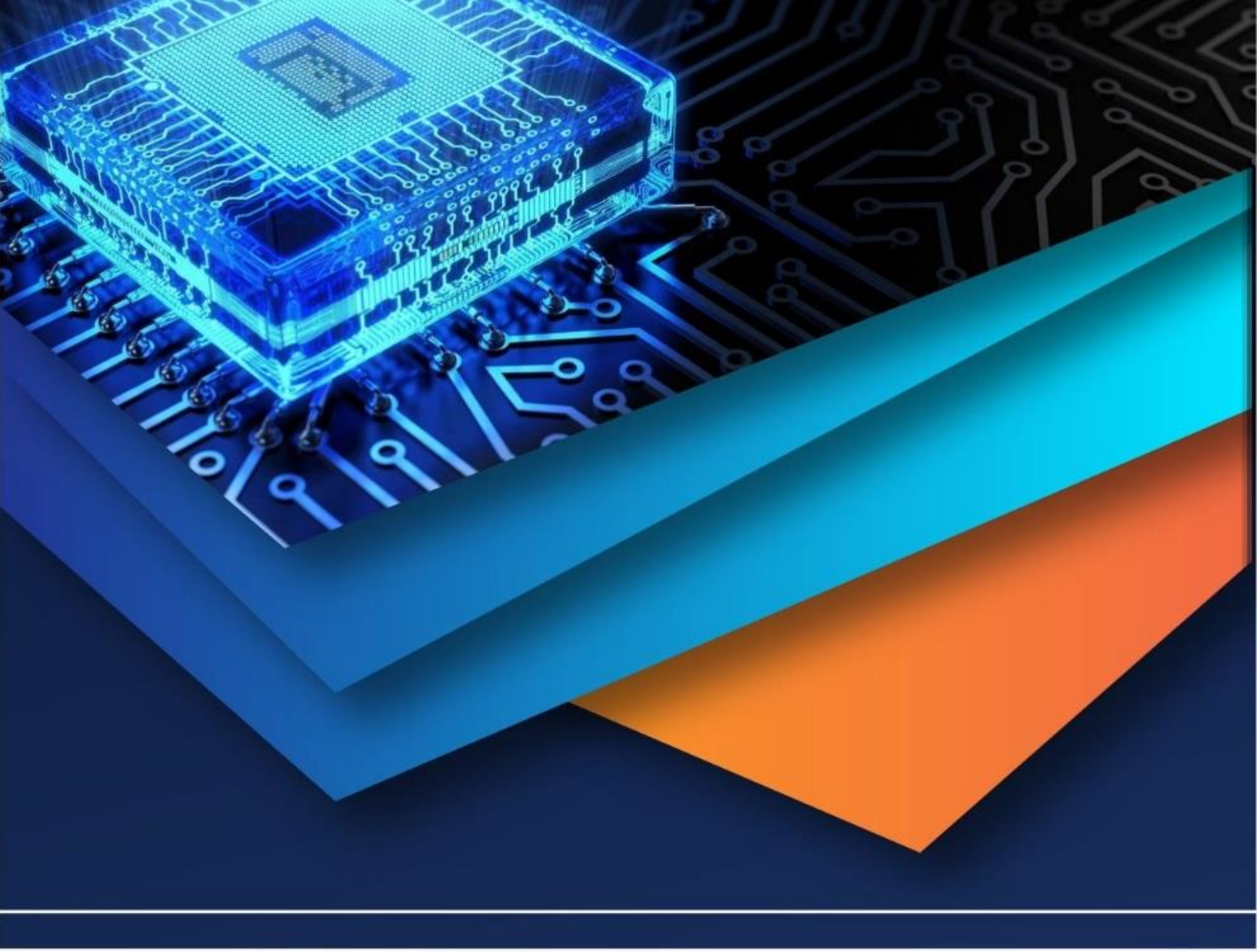
In this paper, we have proposed a simple and high level design of a notification framework that is flexible and customisable and can be integrated with multiple tools and services and send notifications over multiple messaging platforms. This solution is proposed to address aspects like scalability, reliability, real time.

VI. ACKNOWLEDGEMENT

We acknowledge Cloudera for providing the opportunity to work on this project - 'Centralized Framework to Send Notifications from Multiple Tools and Services on Multiple Messaging Platforms' as a part of internship. We thank R.V. College of Engineering, Bangalore and its faculty for the support and guidance in writing the paper.

REFERENCES

[1] Jorge Vicente Cantero, Jiri Kuncar "Building a Real-Time Notification System" Cern OpenLab Summer Student Report 2015
[2] Urja Upadhyay, Disha Jaiswal, Neha Kumari, "International Journal for Research in Applied Science and Engineering Technology, Volume 5, Issue 3, March 2017"
[3] Sagar Gore, Nitesh Sonawane, Sayali Pawar, Mrunal Nerkar, "An Android Based Mobile Framework for Student Alert Notification", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 6, Issue 3, March 2017
[4] Puleng Joyce Jack Mack "Mobile Notification System" Dissertation submitted in partial fulfillment of the requirements for the Bachelor of Technology, Universiti Teknologi PETRONAS Bandar Seri Iskandar 31750 Tronoh Perak Darul Ridzuan
[5] Mehnaz Adnan, Donald Peterkin, Graham Mackereth, "Development of an Electronic Notification System for Influenza-Like Illness Sentinel Surveillance", Stud Health Technol Inform, PMID: 27440281, 2016
[6] Sohom Majumdar, Vatsalya S N "Building a Notification Framework for Microservice-based Application", medium.com, 2021
[7] Dominik Weber, Alireza Sahami Shirazi, Niels Henze "Towards Smart Notifications using Research in the Large" MobileHCI '15: Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct, August 2015
[8] Hana Mohan, "Building a user notification system", online resource, https://www.magicbell.com/blog/building-a-user-notification-system
[9] Oracle, "Understanding the Notifications Framework", online resource, https://docs.oracle.com/cd/E56917_01/cs9pbr4/eng/cs/lscc/concept_UnderstandingTheNotificationsFramework.html#topofpage



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY