Building a user-oriented notification system at CERN

Carina Antunes[1,*], *Jose* Semedo[1], *Andreas* Wagner[1], *Emmanuel* Ormancey[1], *Caetan* Carpente[1], and *Igor* Jakovljevic[1]

[1]CERN

Abstract. CERN, as many large organizations, heavily relies on a variety of communication means for different use-cases and teams. The large amount of notifications received every day leads to adverse consequences. For the users receiving them, it is challenging to control and keep track of where, how and when some information was received. At the same time for those sending and maintaining the tools that deliver notifications, it is difficult to choose which targets to adopt (email, messaging system and platforms, electronic devices, etc.).
The CERN Notifications system aims at consolidating communication by providing a central place where notifications are created, maintained and distributed. CERN Notifications allows not only optimising the flow for the multiple people and teams which are responsible for sending, but also empowers the target users by respecting their preferences: how, where and when they receive their notifications.
This paper describes the design and architecture of the CERN Notifications system and its components, how it was designed with a flexible and highly modular architecture which allows adding further device targets with little effort. Furthermore, it presents implementation details and the decisions behind those. And last but not least it describes the features that empower users to choose how to consume information sent to them.

1 Introduction

Notification models have for long been a topic without consensus. While there's many users content with the status quo, we argue that models where the subscriptions are managed by the information producers (exemplified in Figure 1) present many challenges and pitfall, and therefore propose an alternative to it. In this pattern, users have little control over their configurations (when, where and how users prefer to receive notifications) and no easy way to opt-out. Additionally managers may struggle to make choices which please all users and take into account their preferences. This setup provides as well an added barrier to those without technical knowledge, which have less access to producing notifications due to the lack of knowledge on how to automate and manage information at scale. Ultimately, across all those responsible for sending information, a lot of effort is spent on maintaining similar scripts and tools.

We believe that many scripts and tools default to email and mailing lists in an effort to simplify, as a result of the little flexibility to integrate more devices as targets of delivery,

[*]e-mail: carina.antunes@cern.ch

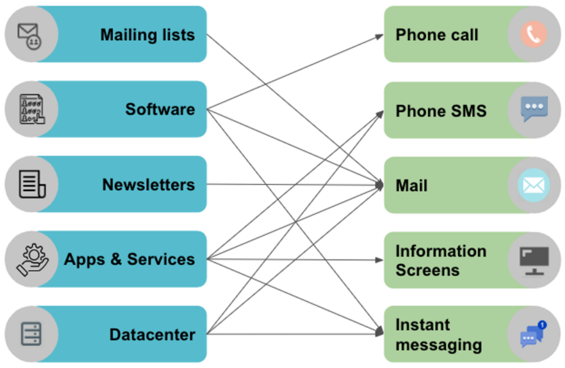


Figure 1. Motivation: Notifications systems flow overview where all subscriptions are managed by services.

and lack of any alternative which complies with individual user preferences. At CERN, on one end of the spectrum we have communication teams writing individual emails to users on a daily basis, which may be small targets, or in the order of thousands. On the other end, there are many automated tools and scripts which generate thousands of notifications daily, mostly in the form of emails. As an example, the study in [1] confirms the trend we've been observing regarding high email usage, which reports office workers received in average around 300 emails daily in 2023 with a 4% yearly increase foreseen for the next years. We believe if we consider service managers and administrators which receive the output of monitoring tools, this number could potentially be on the larger end of the sample.

A significant side effect of the high volume of automated emails is that the higher the volume of information the more challenging it can become for those consuming it, to manage, filter and retain it. Besides the expected challenges to filter what's relevant information and to identify what should be prioritized, there's also risks to the employee well being overtime. As proposed in paper [2] (which builds on information overload, stress and burnout) communications in general, and e-mail communications in particular, should be conceived either as job demands or as job resources, both of which are related to burnout. To mitigate these challenges, as an individual user, one can set up custom mailbox filter rules in order to attempt improving the usability of the mailbox[3]. However this is a limited solution with rules often based on text, such as keywords, subject or sender (which commonly is just a no-reply email). As the rules list grows it becomes increasingly challenging to maintain, and may result in important information being missed due to a faulty setup. Additionally the inconsistent visuals from the multiple individual sources of notifications may generate a lack of trust in the end users consuming them.

For organizations which follow the model described above there is also an associated cost in maintenance, caused by the replication of tools and scripts sending notifications.

We focus on what we've identified as the major pitfalls of the current status quo of notifications systems, and present an alternative. The notification system presented in this solution proposes a shift in the paradigm of handling notifications, moving from a setup where the subscriptions are managed by the services, to a solution which empowers users to make those choices. It allows saving efforts and costs by avoiding multiple parallel implementations of communication systems and their maintenance and details such as retry and failure mechanism, version updates, etc. The system was designed to allow those who send information to focus on the content and relevance of the communication without the requirement to learn about technical details of the many frameworks available to distribute information.

2 Towards a user centered notification system

One of the main goals while designing a solution which empowers users, is to shift the choices of when, how and where to receive communications from the producers to the consumers of information.

Additionally, the system is designed to enhance and complement the existing communication methods, in order to be integrated as a new layer between existing message systems - a hub for the notifications. It aims to provide a simple and common entry-point both for the consumers and producers. Furthermore it aspires to improve overall efficiency in notification handling by absorbing a lot of the effort in maintenance of the many tools and scripts producing notifications in parallel. It avoids multiple implementations of notification handling, and enables efficient integration development. Lastly, it is designed to be scalable and modular, in order to facilitate maintenance in the long term and to allow adapting to changes in needs, e.g. by providing flexibility to add and remove devices at any time. Ultimately the system is designed as a cohesive communication hub, as summarized in Figure 2. The system allows subscriptions to be managed by end users through a central platform, where it is easy to opt-out and organize preferences.

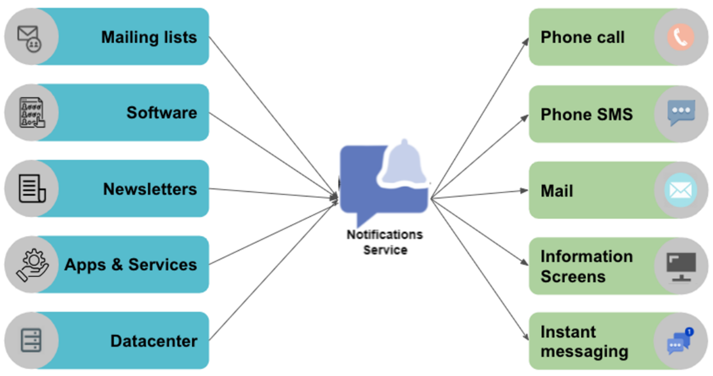


Figure 2. Overview of the notification system, as a new layer between existing message systems, a hub where subscriptions are managed by users.

2.1 Requirements and design

A key functionality designed into the system is to empower services and managers for easy and standard access to users, with no technical knowledge required, but still providing flexibility for programmatic access support for expert users. This is achieved by means of providing both a web interface for non-technical users, and an Application Programming Interface (API) for advanced users.

Many requirements have been bound into the design of the system. In order to reduce mailbox stress, the system allows configuration of alternative target devices (SMS, browser push notifications, etc.), muting irrelevant notifications entirely, or aggregating notifications under a digest format. By implementing a shared and recognizable notification layout, message trust is improved - making use of this central hub, distribution is done in a standardized and recognizable message layout, which in turn addresses the issue of heterogeneity among communications, that undermines trust and aid users recognize possible malicious outliers (phishing). Furthermore, the system improves confidence and reduces human errors, since email filter configurations are not based on text, but instead rules are directly set in the system, where users can target the producers of notifications directly (by the concept of channels). It provides concepts of critical/emergency notifications and mandatory communication channels, features that improve trust in senders, as a consequence of being built-in features and verifiable within the system. Notifications are non-deletable and non-editable, to achieve trust and accountability in senders.

The hub is consolidated under the form of a web page, a central place to manage the system, both for senders and receivers of notifications. Users can create channels, and manage their subscriptions. They can consult past notifications, and set up preferences.

Preferences are a key concept of the model - they empower users by allowing them to select how, where and when they want to receive their notifications. Users can set up preferences globally as default rules, as well as specific preferences by channel. As an example, a user can configure a default preference specifying all notifications should be received as daily summary, at the end of the day, via email, by creating a global preference stating this. Additionally, for a channel called "LHCb Monitoring", which the user deems important, they can configure that all notifications are to be received instantly on all available devices. This is possible with Channel-specific preferences, in this example a preference which only applies for channel "LHCb Monitoring". This level of flexibility lessens information loss - in allowing non time-sensitive communications, which frequently are overlooked and discarded, to be consulted or aggregated into summaries at a later date.

Users can also mute channels deemed irrelevant. We refer to the action of muting a channel as a "mute". Even further, they can configure global temporary mutes. E.g, when going on holidays, a global mute can be created for the duration of the absence, clearing the communication devices for urgent, relevant and user-to-user communication, which would otherwise be busy with notifications one is not planning to act upon.

The system supports both concepts of individual users and groups, for channel members and for targets of notifications. Subscriptions are mapped as channel-user associations, but as well as channel-group. Individual users can be added to the channel members by the owners and administrators of a channel, and as well as self-subscription, if the channel configuration allows it. For the groups, the system is designed to connect with the CERN Identity Services[4], which allows reusing the already existing concept of groups. The system will fetch the group memberships and send the notification to all the users mapped to that group.

2.1.1 Model and concepts

A brief overview and definition of the most relevant model entities.

Users represent a single identity. Users may be both creators and consumers of notifications. Users may be owners and have administrator privileges of channels.

Groups represent a group of users. Groups are used to set up permissions. Their memberships are extracted and managed in a 3rd party service, the Identity Services.

Channels are a unit to organize and group notifications, and their properties such as visibility and privacy.

Memberships associate users and groups to channels, in order to receive notifications. Other associations between channels are **un-subscriptions** (to stop receiving notification) and **mutes** (both permanent and temporary).

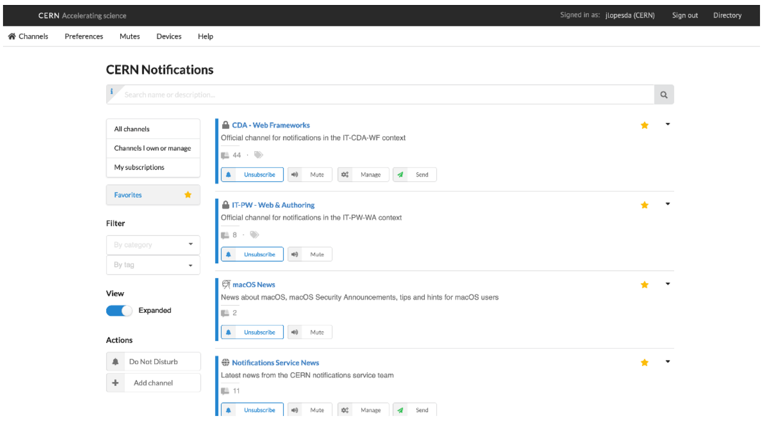
Preferences store configurations of when, where and how to receive notification. Preferences hold information regarding to which notifications they apply - which priority (Low, Normal, Important) and during what hours they apply - and what is the outcome of the preference - what frequency (Live, Daily, Weekly, Monthly) and to which device they'll be delivered.

Devices model the target of the notification, such as email, browser push notification, SMS, instant chat message.

Notifications model the actual notification content. They must be associated with a single channel, and multiple users and groups optionally. The system currently supports both text and HTML for content types. Notifications are non-deletable and non-editable.

2.2 Interface

The Web interface serves as a central place for simple and standard access to all users, both with technical and non-technical expertise. For consumers of notifications the interface allows users to list and configure their memberships: subscribe, unsubscribe and mute channels. Additionally it allows them to see past notifications of specific channels, and configure their preferences and devices. For producers, users can create channels and configure them, with Web pages to manage channel members and visibility. Channel administrators can send notifications via the Web interface with a full featured WYSIWYG HTML editor.



CDA - Web Frameworks
Official channel for notifications in the IT-CDA-WF context
44

IT-PW - Web & Authoring
Official channel for notifications in the IT-PW-WA context
8

macOS News
News about macOS, macOS Security Announcements, tips and hints for macOS users
2

Notifications Service News
Latest news from the CERN notifications service team
11

Figure 3. The Web Interface's landing page, showcasing the list of channels a user has access. The central element of the page is a list of channels. For each channel several shortcut actions are available: muting, (un)subscribing, managing the channel and sending notifications. Users may favorite their channels in this page as well.
At the top bar, options for channels can be seen, as well as option for preferences, mutes and devices. A Help page is accessible as well.
At the left column, many options for filtering channels are available: all channels, channels with administrative privileges, subscribed channels and favorites. Additionally filters for categories and tags for discoverability and exploration. An option for expanded or reduced view, for advanced users convenience. And lastly options to mute all channels and create new channels are available.

2.3 Architecture

A summarized view of the architecture can be seen in Figure 4. The user facing components include the already mentioned - a Web Interface and the REST API - as well as an Email gateway for convenient integration with legacy systems using emails, and a Swagger UI integration under the form of a Web page. The Swagger integration is exposed in a QA environment for users to be able to get familiar with the API and explore it, without the risk of accidental spam.

The heavy lifting of sending notifications is asynchronous and backed by message queues. A modular architecture has been adopted, which allows to reduce the complexity by splitting the code by functionality. For the asynchronous part two main concepts exist: routing and the consumers. The first processes a notification, expanding a notification members and their preferences, and routing it into the consumers. The consumers are the part of the system responsible to actually deliver the notification into the end devices. There is a consumer per device type, i.e., one consumer responsible for pushing via email, another for delivering via SMS and so on.

Other mainstream components have been adopted such as a component to handle infrastructure as code, others for monitoring, auditing and archiving. The infrastructure as code component exists under the form of a Git repository, which allows deploying with the stateof-the-art continuous integration tools[5]. It allows versioning, recovery, automated releases and rollback flows.

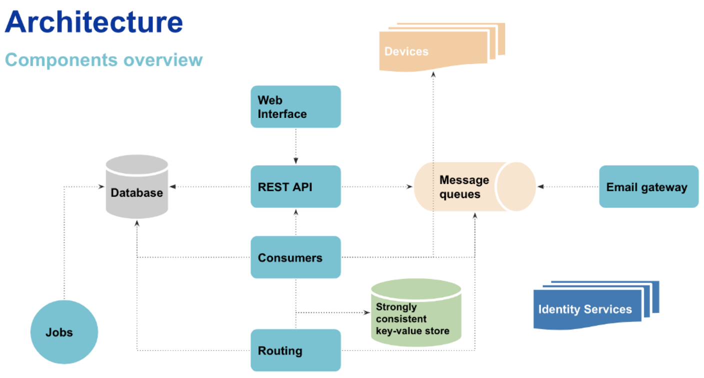


Figure 4. Notification system model: main components overview

All stack has been chosen to comply with Open Source by design, in order to avoid vendor lock-in. It is just as well, heavily influenced by the current landscape of stack offered at CERN via central services, following guidelines to reuse as much as possible in order to optimize resources, and promote maintenance in the long run. This results in a vast stack of widely adopted technologies, frameworks and languages such as ActiveMQ[6] for the message queues, PostgreSQL for databases and OKD[7] for hosting.

2.4 Member Expansion and Preference Routing

A single notification object in the system is mapped to many end targets. The system, specifically the Routing component (see Figure 5), holds the logic to fetch all associated users, their preferences and their devices. This component multiplies a notification object associated with a channel (and optionally directly with a set of target users and groups) into many individual notification messages associated with a single end user/device.

Processing starts by expanding all of the users associated with a notification: for each notification, it fetches the list of users and groups members for the associated channel (or the targets directly). For each group (in the channel members list) it will reach the Identity Services and fetch its list of users members. Processing continues by merging all the individual users, removing duplicates, mutes and unsubscribes. Finally for each individual user, their list of preferences which applies is fetched (time, format and device a user prefers to receive said notification). The final individual notifications are passed onto the consumers via the message queues.

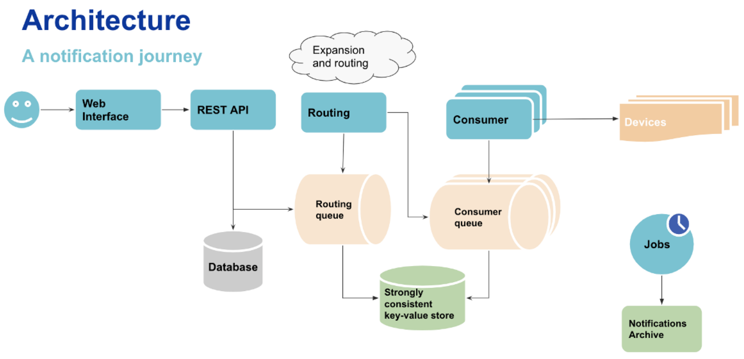


Figure 5. Notification flow overview through the system components from creation to delivery

2.5 Delivery Strategy, Retries and Failure Handling

Within the system, the notification journey can start in the web interface, in the email gateway or the REST API (and Swagger[8]). It is then stored in the Database and put into the Routing queue - at that moment, for the producer the process is concluded and they'll receive a success response.

Asynchronously, the processing of the notification starts at that moment. The Routing component reads the message holding the notification information from the routing queue, expands it into multiple notifications by end target (both user and device information) and generates as many messages into the corresponding consumer queues. A queue and consumer exist by device type, which allows to scale all components individually and horizontally. Finally the consumer components deliver the notification to each device, exactly once.

All queues have configuration for retries following an exponential backoff strategy, which allows a growing time window for recovery and solution of any issues in the system or codebase. A persistent failure lasting days is not expected, since the system includes monitoring and alerting. Nonetheless Deadletter queue[9] have been configured as a last resort safety policy. Those will generate alerts for administrators of the system to manually act upon.

Configuration for retries in queues and consumers guarantees at-least-once delivery[10], but could lead to duplicated deliveries. To achieve exactly-once delivery, and for queue consumers to be idempotent, a strongly consistent[11] key-value store has been added. This component stores the state of delivery to a device of a notification (Notification ID/Device ID).

A retry after deliver to device is not expected in the Consumers. Regardless, checking this state is sufficient to avoid duplicated delivery. However in the Routing component, this store is fundamental to avoid large numbers of duplicates in case of failure. Due to its implementation, failures can occur in the middle of processing, where part of the individual notifications could have already been generated and passed to the consumer message queues. As an example, network instability can occur in the middle of processing, causing the message to be returned to the queue for later retry, at a stage where part of the individual messages have been sent to the consumer queues. On retry, processing starts again from the beginning. To ensure idempotency, and avoid duplicates being sent, the introduction of this store allows skipping all notifications already delivered.
For this store, etcd[12] which is distributed, has been used, as it is fast and simple, and the data needs only a short life span.

3 Evolution

The system described is a new solution that aims to become a powerful hub which empowers users. At time of publication it is being evaluated for its potential to help user communities. Simultaneously it is available at CERN (https://cern.ch/notifications) and has been steadily growing adoption and performing well. Nonetheless there is still much room for improvements. Additional features have been requested that could provide growing usability and integration of wider use-cases, such as support for recurrent notifications, creation and usage of templates for notifications, statistics for notifications and channels, a built-in media store, customization of channels and notifications look-and-feel for limited and relevant official channels and a Indico integration (Indico is a open-source event management system, popular in the HEP community).

CERN Notifications is an Open Source solution. In order for it to be flexible enough to be integrated within other organizations, refactoring to a Plug-in architecture[13] is required, providing extension points, interfaces and configurable modules. Devices could be added and passed as extensible plugins, allowing alternative implementations and facilitating packaging for easy deployment.

Finally, consolidation of the deployment methods, performance tuning and optimization of notifications deliveries per device type are being performed. A specific focus is set on setting limits, monitoring probes, improving horizontal auto-scaling, tracing and testing. Service Level Agreements have yet to be defined, as well as to identify the limits the system can handle. Work to optimize all the codebase is ongoing, focusing on concepts such as pagination, caching, memoization, multi-threading, configuration and memory/cpu optimization.

4 Source Code

The Notifications system is Open Source and its source code is available at https://github.com/hermodnotifications/.

References

[1] *The Radicati Group, Inc. Email Statistics Report 2023-2027*. 2023. URL: https://www.radicati.com/wp/wp-content/uploads/2023/04/Email-Statistics-Report-2023-2027-Executive-Summary.pdf.

[2] Claudia P. Estévez-Mujica and Eric Quintane. “E-mail communication patterns and job burnout”. In: (2018). DOI: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0193966.

[3] *Email Filtering*. 2023. URL: https://en.wikipedia.org/wiki/Email_filtering.
[4] A. Ahmad et al. "The new (and improved!) cern single-sign-on". In: (2021). DOI: https://doi.org/10.1051/epjconf/202125102015.
[5] *How to create a CI/CD pipeline with Auto Deploy to Kubernetes using GitLab and Helm*. 2023. URL: https://about.gitlab.com/blog/2017/09/21/how-to-create-a-ci-cd-pipeline-with-auto-deploy-to-kubernetes-using-gitlab.
[6] *ActiveMQ*. 2023. URL: https://activemq.apache.org.
[7] *OKD*. 2023. URL: https://www.okd.io.
[8] *Swagger*. 2023. URL: https://swagger.io.
[9] *Message redelivery and dlq handling*. 2023. URL: https://activemq.apache.org/message-redelivery-and-dlq-handling.
[10] C. Fehling et al. *Cloud Computing Patterns*. Springer Vienna, 2014.
[11] S. Newman. *Building Microservices*. O'Reilly Media, 2015.
[12] *etcd*. 2023. URL: https://etcd.io.
[13] F. Schweiggert J. Mayer I. Melzer. "Lightweight Plug-In-Based Application Development". In: (2002). DOI: https://doi.org/10.1007/3-540-36557-5_9.