SystemDesign Follow
Nov 8, 2021 · 10 min read ★ · Listen

System Design Interview: Notification Service

PREV | **HOME** | **NEXT**

Although it does not exist independently, a notification system is an important part of many other systems. Whether you're designing an e-commerce platform or a recruiting system, such as indeed.com, sending notifications is a requirement of many modern applications.

> *Get a leg up on your competition with the* ***Grokking the Advanced System Design***

When subscribed to an e-commerce outlet, customers may be notified of the availability of a product, order placement, or order shipment. When subscribed to a recruiting platform, users may be notified of the latest jobs appearing for the job criteria that they have set. The notifications that pop up on your mobile phone for a new post on your Facebook newsfeed also come to you via the notification service. Many applications use a notification system to send OTPs via email or SMS to authenticate users.

Let's learn how to design a notification service using messaging queues.

System Requirements

Here is a set of functional and non-functional requirements for the system:

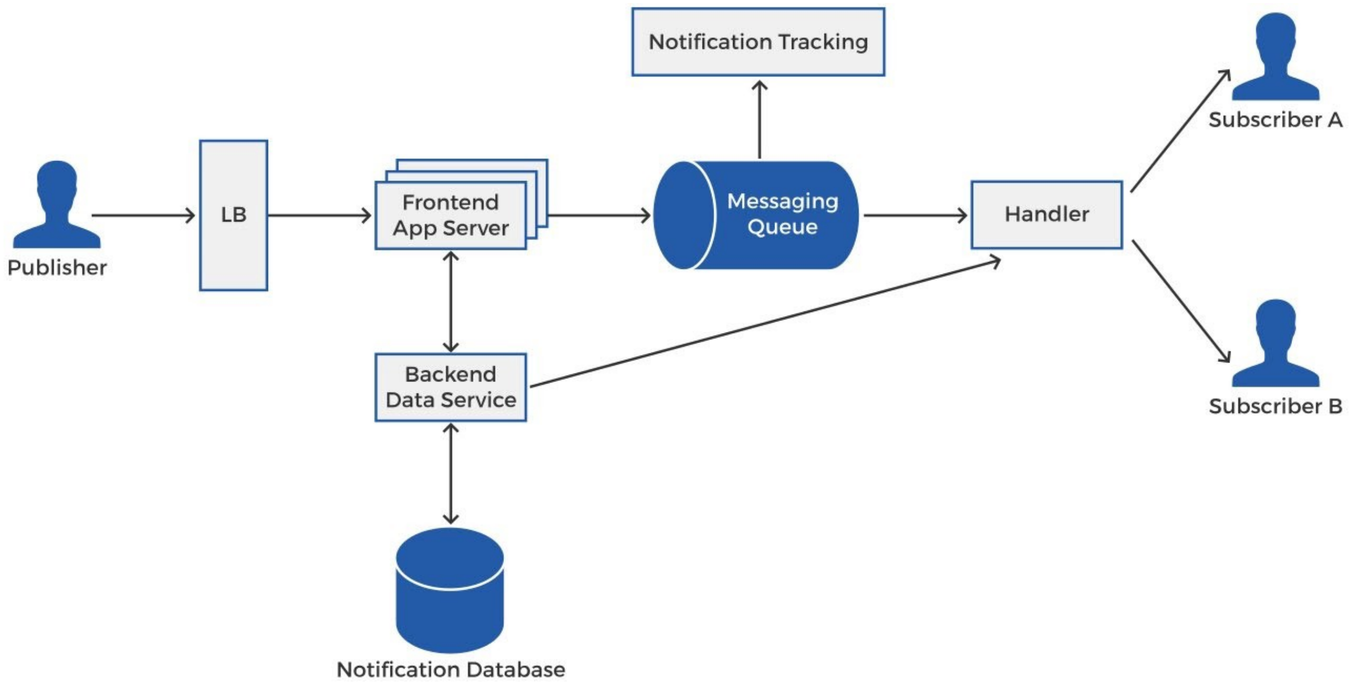
Functional Requirements

- The system should be able to send notifications to the subscribed consumers.
- The system should be able to prioritize notifications. OTPs are high priority messages while news feed updates may be lower priority.
- The system should be able to support Email, SMS and push notifications on mobile and web browser.
- It should be an isolated system that is easy to integrate into an existing system, such as Facebook or LinkedIn.

Non Functional Requirements

- The system should be highly available.
- Latency should be low. OTP messages are time-critical.
- It should be scalable, to handle a growing number of subscriptions.
- The system should be linearly scalable to provide services to different service providers.

High Level Architecture



Publisher

The client for our notification service will be a service provider who wants to send notifications to users that have subscribed to their service. Let's call them the publisher since they publish messages. The publishers will send requests to the notification service to send notification messages for them.

Load Balancer

The client's request to the notification service will first hit the load balancer. The load balancer helps distribute load evenly across the notification system servers.

Frontend App Server

For the high-level design, we're taking the entire notification system as a monolithic system and enclosing it inside the block named "Frontend App Server" in the diagram above. The app server uses information from the backend data service to generate messages. In reality, as we'll discuss later in the blog, it's not as simple. The app server is made of several individual components that collectively manage notifications.

Notification Database

The app server sits on top of a database — let's call it the Notification Database. It will store topics and subscriptions information in the form of tables, for which RDBMS is a good choice.

Check out the course **Coderust: Hacking the Coding Interview** for Facebook and Google coding interviews.

Backend Data Service

We use a separate microservice to fetch data from the Notification Database and deliver it to the app server. While the database could also be designed to directly interact with the app server, having a backend data service allows the caching of notification information so that the front end app server can retrieve it quicker. Only when the information, such as the "topics" or "subscribers" aren't available in the cache, the backend data service will contact the database.

Messaging Queue

The app server will push the messages to the messaging queue for temporary storage. We have used Kafka to store messages, but other queuing services, such as Amazon Kinesis can also be used. The queue holds the messages until the 'handler' picks it up.

Handler

The handler picks up the messages and sends them to the subscribers.

Subscriber

Subscribers are clients of the publisher that have expressed interest in certain topics, either explicitly or through their actions. Their contact information, including Email ID, phone number are stored in the database against the subscribed topics. The subscribers can cancel the subscription ("unsubscribe") from topics if they want to stop receiving

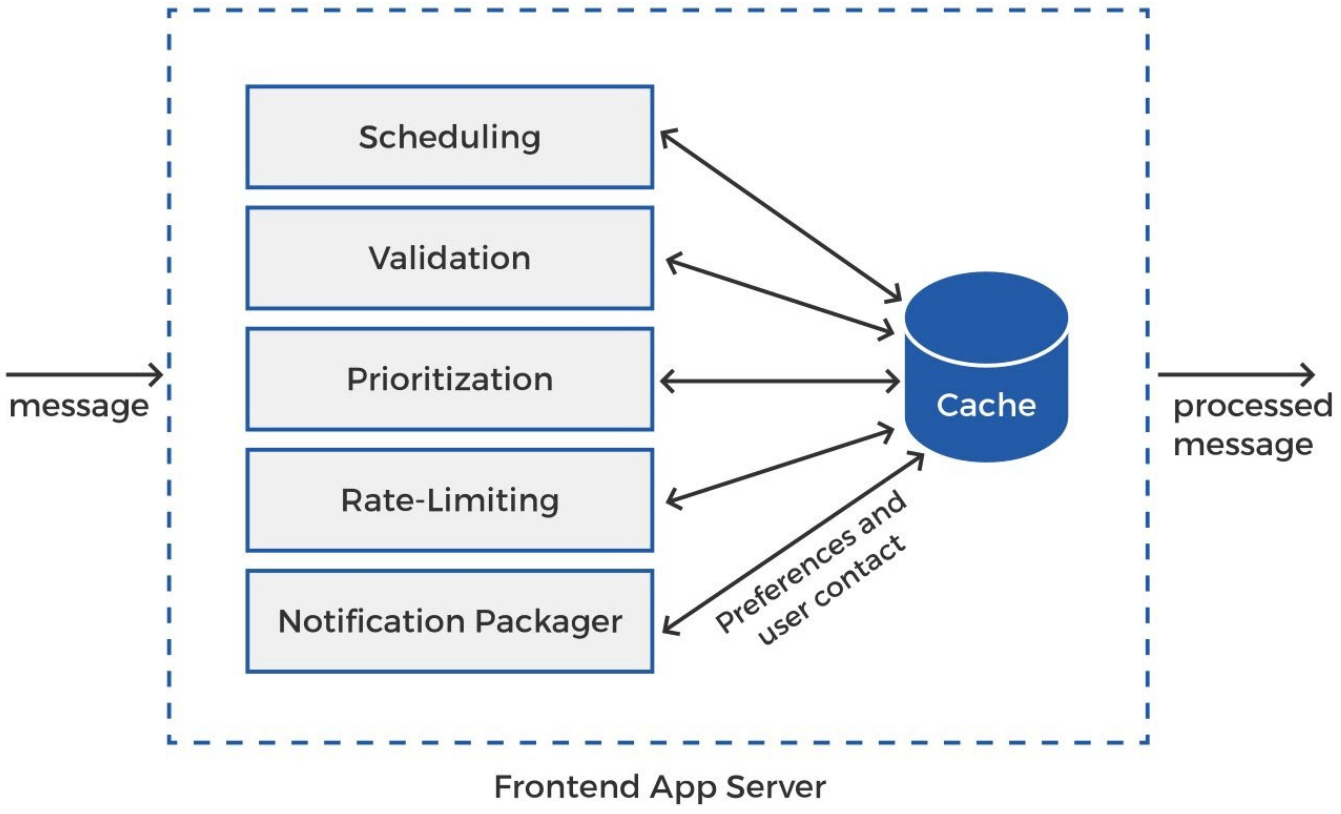
In the diagram, you can see that a service called "Notification Tracking" continuously reads events from the messaging queue. It maintains notifications metadata for system analytics. This service is discussed in detail in the next section.

Low Level Architecture

Now let's zoom in on some of the major components of the high level design to understand how they work.

If you are preparing for tech interviews at FANGs, you may want to check out the course ***Grokking the System Design Interview*** *by Educative.*

Frontend App Server



The load balancer sends the request of the publisher to the app server. We mentioned number of calls made to the backend data service, as shown in the diagram. Google Guava is an excellent choice for maintaining a local cache.

Some of the services that the frontend app server caters include:

Scheduling

The scheduling service offers the APIs for the clients (publisher) to schedule the notifications they want to send. Does the notification need to be sent immediately, or once a week, or once a month? The client can select the frequency or the time for sending messages according to the requirements.

Validation

Messages will be validated via the validation service of the frontend app server. This service validates the messages to make sure it abides by the rules and formats. For example, the email should not be invalid.

Prioritization

The prioritization service prioritizes messages based on which it will be sent to high, medium or low priority queues. OTP messages are high priority messages since they need to be received right away. Certain bulk messages, such as promotional messages are low priority messages, and can be sent to the low priority queue. They are sent to the subscribers during off-peak hours.

Check out the course **Coderust: Hacking the Coding Interview** for Facebook and Google coding interviews.

Rate-Limiter

The rate limiter checks the messages for two things:

- If the publisher is allowed to send that many requests.

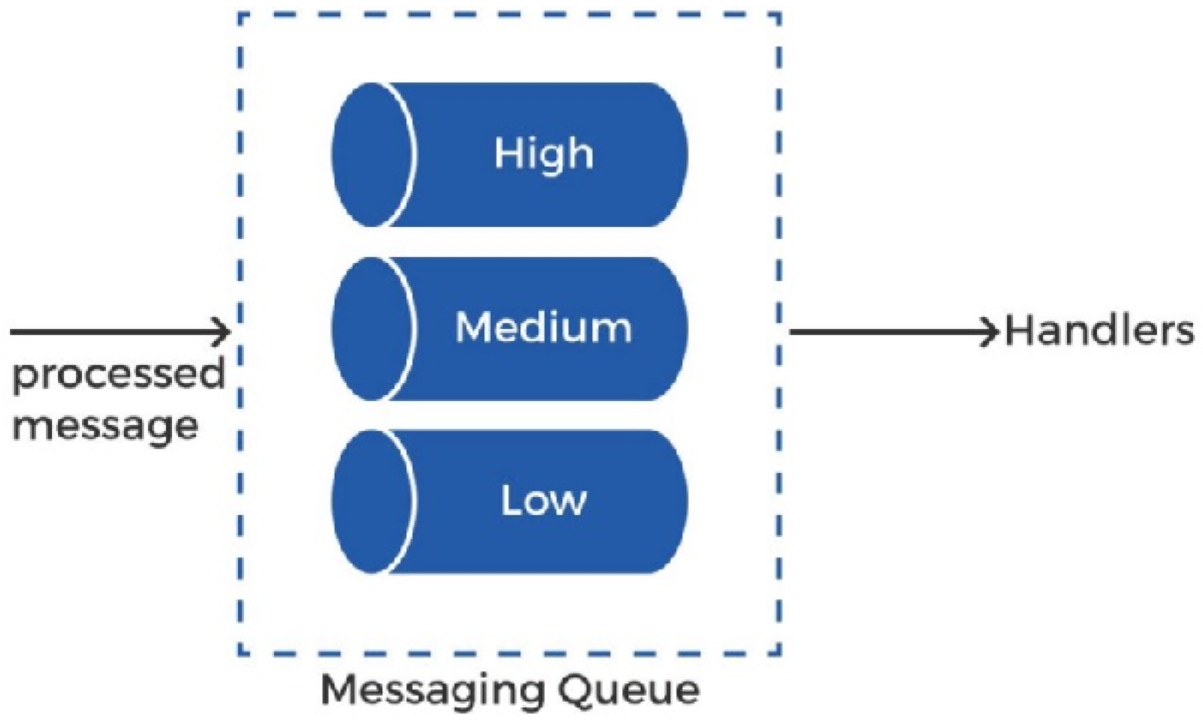
The rate-limiting information is stored in the cache and is checked and updated against the rate-limiting data that comes with the message. If the rate limit isn't exceeded, the request is forwarded. If the rate limit has exceeded, the request is throttled (dropped).

Notification Packager

After all the checks on the message, including validation, prioritization and rate limiting, the final component that the message will go through before leaving the frontend app server is the Notification Packager. This component packages the notifications with the necessary metadata to send it to the subscribers.

If the subscriber has unsubscribed from a topic or has given a preferred channel for promotional messages, the information will be in the cached preference DB that the frontend app server maintains. The subscriber's contact information is also pulled from the cache and appended to the message as the destination address. Now the message is ready to leave the frontend app server.

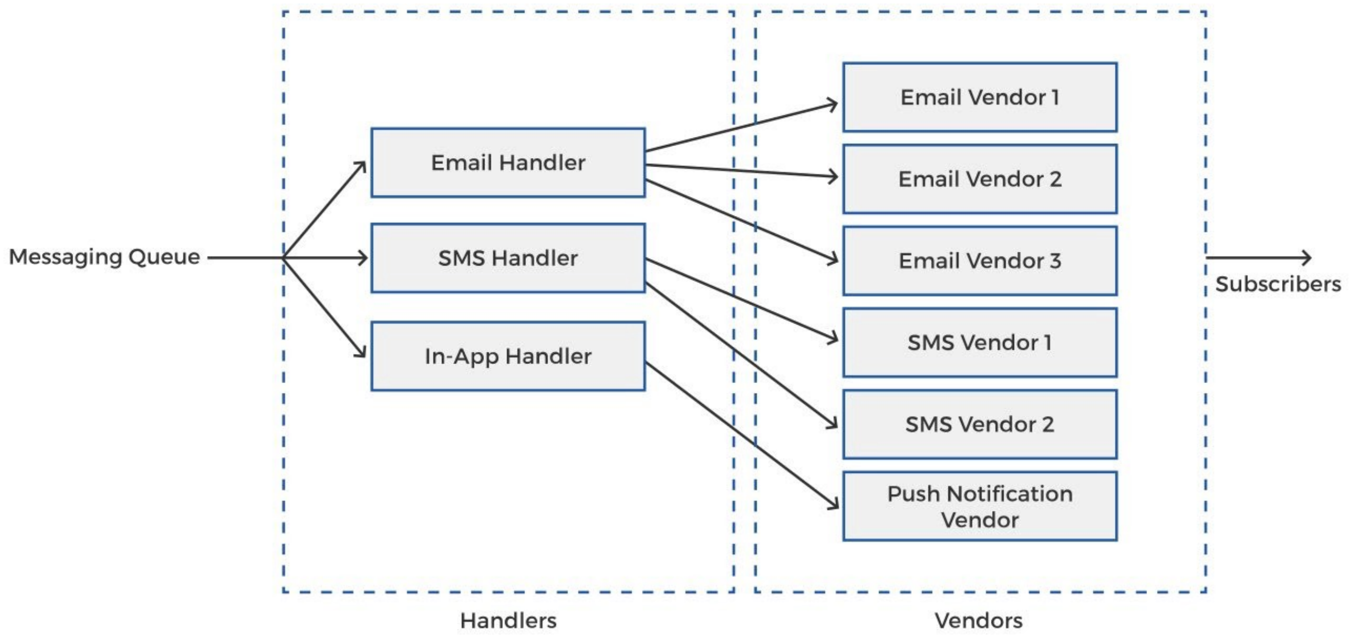
Messaging Queue



The processed message from the frontend app server is pushed to the messaging queue, let's say Kafka, as an event. We use Kafka since it is highly available, scalable and guarantees at-least-once delivery by default. This means that if a subscriber is unavailable, the delivery will be re-attempted unless the notification reaches them at least once. If your application calls for at-most-once delivery, you can configure the settings to that by disabling retries.

The event that is sent to Kafka by the frontend app server will contain the message, destination address and contact channel. Three queues are maintained on the basis of priority: high, medium, and low. The consumers of the Kafka topics, which in this case are the handlers connected to Kafka, as shown in the diagram, will pick the events from the queues based on their priority. The events from the high priority queue are picked first, then the medium priority and then the low priority.

Handler



Various handlers are connected to the messaging queue, as seen in the diagram above. These handlers consume messages from the Kafka queues based on the message type and message priority.

send out in-app notifications. The in-app notification handler will further contact a push notification vendor, such as Firebase to send out push notifications to the users.

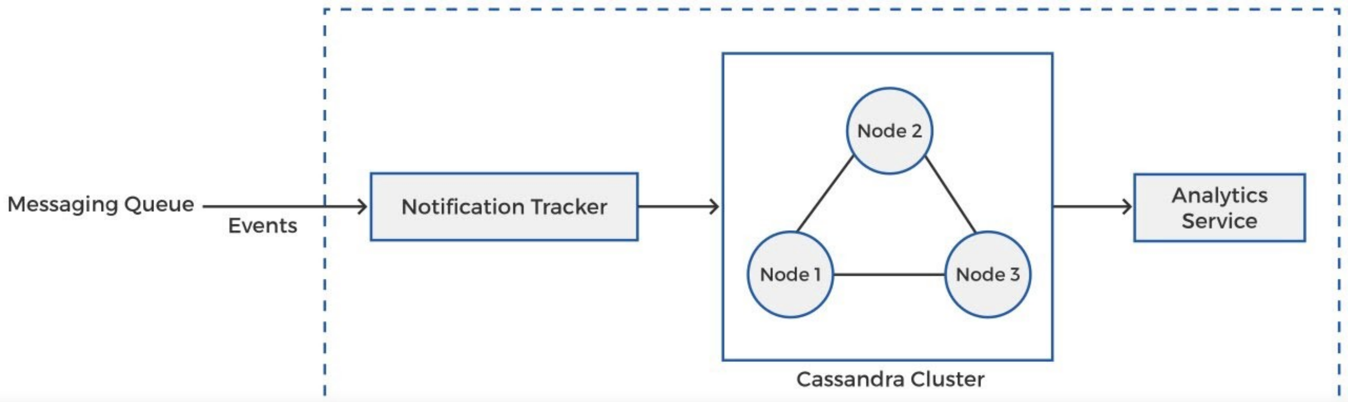
Check out the course **Coderust: Hacking the Coding Interview** for Facebook and Google coding interviews.

Vendor

Each handler will be in contact with different vendors. The SMS handler, for instance, will interact with a number of SMS vendors to send out notifications in different regions. There may even be different SMS vendors available in the same region. The SMS handler will decide which vendor to send the notification to based on the metadata information with the message.

Similarly, the Email handler will be in communication with the Email vendors. Each time it gets a request, it will forward it to the suitable Email vendor to send out an Email to the subscriber. The in-app handler will contact Firebase for sending in-app notifications to Android users. For iOS users, the in-app handler will send the notification over the Apple Notification Service. The diagram above shows how vendors interact with the handlers to send out notifications.

Notification Tracking



As you can see from the diagram above, Notification tracking is composed of three elements: Notification Tracker, Cassandra Cluster and Analytics Service. Together they will track and analyze the notifications from your system.

Notification Tracker

The notification tracker continuously reads events from the messaging queue. In other words, it picks up notifications sent over the messaging queue. With the message, it also identifies and reads the delivery status, delivery time, message type, and other metadata.

*If you are preparing for tech interviews at FANGs, you may want to check out the course **Grokking the System Design Interview** by Educative.*

Cassandra Cluster

The Cassandra cluster is a distributed database for storing notifications information for analytical purposes. This is a write-heavy database to store all the notifications sent over the messaging queue permanently. Since it duplicates messages across multiple nodes, messages are not lost even if a node crashes. For this reason the database is reliable and highly available.

Analytics Service

The analytics service consumes the sent notifications data from the Cassandra cluster to perform various analytics on the system. Analytics may be important for a number of different purposes, for example if you want to find out the number of messages sent by your notification service in an hour or in a second, or the average size of notifications, or identify the most used communication channel and so on.

How To Send Bulk Notifications

There may be use cases for your notification system where the clients want to send bulk notifications to a specific group of users. To allow bulk notifications, we can have a separate bulk notification interface for the clients to interact with. The clients can add their criteria for picking users they want to send the notifications to.

From the bulk notification UI, the bulk notification service consumes the notification message and user criteria to select the receiving users. The message is then passed over the same architecture as the one for a single notification discussed above before reaching the subscriber.

Check out the course **Coderust: Hacking the Coding Interview** for Facebook and Google coding interviews.

Top YouTube Videos On Designing A Notification Service

- System Design Interview — Notification Service by System Design Interview
- Notification Service System Design Interview Question to handle Billions of users & Notifications by codeKarle
- Notification System Architecture by Ben Awad
- Scaling Notifications | How Instagram manages notifications at scale? | System Design by Arpit Bhayani
- Facebook System Design Interview Question | Design a Notification System by WorkWithGoogler

Conclusion

This is how a notification service is designed. With scalability in mind, each component of the system can be horizontally scaled. At the same time, when each component is distributed across multiple data centers, we can also ensure high availability for the

Life Lessons Self

174 claps

If you enjoyed the article, kindly support us and consider following. Thank you :)