User Notification Service /Module

Overview:

This is the generic module which can be configured/integrated with any enterprise level product which want to send notifications to users. This module should be generic enough so that it can be used with any product irrespective to what programming language has been used by them.

When this module is integrated with other services and components, overall diagram and interaction will look like as below.

Diagram

Description automatically generated

Requirements:

1. **Functional Requirement:**

Any service (service-1 to service-n) should be able to send data to “Notification” service if they have generated some event after processing of any business data (say financial transaction or any other kind of business processing). Notification module is able to send notifications (email, sms, push notification to user) based on the event type. Notification for an event can be send to user through one or multiple channels (email, sms, push notification).

1. **Non-Functional Requirements (NFR):**

Below are NFR requirements which I will consider based on requirement given:

* **Performance:** Any notification sent to notification **module should be processed within 1 minutes of time**.
* **Scalability:** Based on the load from external services (pushing events to notification service/module), notification module **should auto scale** its internal components.
* **Portability/Compatibility:** This service should work with any service irrespective of their underlying technology of implementation. **It should be technology agnostic to other services.** This service should work as a service can be integrated with any other microservices.
* **Reliability and availability:** This service **should always be available for 24\*7\*365**. If there is any kind of error, it should be able to recover from its failures. **There should be no data loss** while processing events
* **Security:** Only the authorized clients will be able to send events to this service for processing. Unauthorized to its API will throw error to client.

Flow Diagram:

Below is flow diagram for event creation and then notification processing and sending using notification module.

Diagram

Description automatically generated

**Note:** Here in this flow diagram, “Error/Log Tracking” system is a system where we can keep track for error occurred for events which also has data of services and users. Users/services can see this data on their dashboard or some reporting tools.

**Architecture:**

As per requirement, we need to have a decoupled module which is highly available and resilient system.

Diagram

Description automatically generated

**Different components of Notifications of module and the technology used and the reasonings:**

* **Event Producer:** This is component which will push event data (sent by client) to message queues.
* **REST APIs:** We will expose REST API which can be used by client to push event data to Notification module. This REST API will be part of “Event Producer” component.
* **Scheduled/Routine Process:** This is routine process which will continuously look for any data in local file system.
* **Local file system (inside Event Producer):** If secondary Kafka cluster is down, event data will be written to local file system.
* **Primary Kafka Cluster:** Event data will be written to Kafka queues of primary cluster by default.
* **Secondary Kafka cluster:** Event data will be written to it if primary Kafka cluster is down or during migrations/upgrade related activities.
* **Event Consumer:** It will consume the event data pushed to Kafka queues. It will process event data and based on distribution medium for event, it will send notifications. While processing event data, it will interact to external services (for email service, sms service and/or FCM(Firebase Cloud Messaging)).
* **External Service Providers:** Based on event type, event consumer will use external service providers to send notification to user using proper channel.

**Data Points for selection of Technology:**

1. We wanted to have **event-based processing** for that we have to use some messaging queue. For this we can use either **AWS SNS**(Simple Notification Service) if our infra is in AWS. But that is managed service from AWS.

As mentioned in requirement, there will be high user engagement so obviously number of events generated after business processing will be very high. And Kafka can handle **high volumes of request**. So I have **chosen Kafka** for messaging queue.

There are others also like **RabbitMQ, ActiveMQ**. Detailed analysis can be done if there are some other requirements and decision can be taken.

1. Because we want **high** **configurability** of notifications, we need to use templates which takes dynamic parameters and static content can be configured.

External Service Providers:

|  |  |  |
| --- | --- | --- |
| Distribution Medium | External Software/Service | Reason for choosing |
| Email | Twillio SendGrid | * Templates can be configured/managed without any dependency on business logic and takes dynamic parameters. * If-else, loop Logic can be implemented in templates * Dynamic values can be passed to templates * More popular email service providers. * Static image configuration support in template. * Versioning support for templates |
| SMS | Twillio SMS | * More popular and provide all features required for integrating SMS. |
| Push Notification | Firebase Cloud Messaging (FCM) | * Provide support for notifications on Web applications (all browsers), Android, IOS Apps. * Technology agnostic. It supports multiple technologies |

1. I have chosen these technologies so that dynamic params can be passed to these templates at run time and templates can be measured without affecting the business logic. If we have to add new template for any use case or want to modify existing template, admin users can do it without affecting business logic.
2. Client services can also send information related **to template, template versions and their distribution mediums** while posting event to notification modules by calling REST API.

**REST API Detail to push event to Notification Module:**

REST API details are as below which need to be followed by any client service which intend to call it.

**URI:** /api/event

**Method:** Post

**Headers:**

|  |  |  |
| --- | --- | --- |
| Key | Value | Comment |
| APP\_KEY | <Client specific service APP key> | For security reason, every client (caller to it) will have to get APP\_KEY and APP\_SECRET |
| APP\_SECRET | <Client specific service APP secret> | For security reason, every client (caller to it) will have to get APP\_KEY and APP\_SECRET |
|  |  |  |

**Request Payload:**

{

"notificationMetaData":[

{

"notificationType":"1",

"templateId":"et-1",

"dynamicParams":{

"key1":"value1",

"key2":"value2"

},

"receivers":[

"a1@xyz.com",

"a2@xyz.com"

],

"from":"x1@xyz.com"

},

{

"notificationType":"2",

"templateId":"st-1",

"dynamicParams":{

"key1":"value1",

"key2":"value2"

},

"receivers":[

"9811111111",

"9811111112"

],

"from":"MIC-IND"

},

{

"notificationType":"3",

"templateId":"pt-1",

"dynamicParams":{

"key1":"value1",

"key2":"value2"

},

"receivers":[

"deviceXXXXX1",

"deviceXXXXX2"

]

}

],

"messageContent":"",

"subject":"Welcome to APP"

}

**Response:**

{

"status":"SUCCESS",

"eventId":"evntid-1"

}

**More Infor about data in payload of REST API:**

1. We will have some metadata about what different notification are distribution types/channels as below:

**Supported Notification Types:**

[

{

"id":"1",

"name":"EMAIL"

},

{

"id":"2",

"name":"SMS"

},

{

"id":"3",

"name":"PUSH\_NOTIFICATION"

}

]

1. “dynamicParams” will contain the list of key-value pair which need to be replaced during processing of notification event. These values will be replaced in in template or messageContent.
2. “templateId” is optional and if it is not found, “messageContent” will be used. And dynamic parameters will be replaced in this content.