

Project Report

on

Message Queuing using RabbitMQ

**By,
Swapnil Katkar
Jithin J Eapen
Fall 2017
(Group No - 3)**

Table of Contents

Table of Contents.....	2
1. Message Oriented Communication	3
1.1. Message-Oriented Transient Communication	3
1.2. Message-Passing Interface.....	3
1.3. Message-Oriented Persistent Communication.....	4
2. Message Queues.....	5
2.1. Advantages of Message Queues	6
3. Publisher-Subscriber model.....	6
3.1. Advantages of Publisher-Subscriber Model	7
4. RabbitMQ.....	7
5. Problem Statement.....	8
6. Project Environment.....	9
7. Project Technical Architecture.....	10
8. Project Screenshots.....	11
8.1. Publisher application.....	11
8.2. Subscriber applications.....	12
9. Conclusions	14
10. References	14

1. Message Oriented Communication

Message-oriented communication is a way of communicating between processes. Messages, which correspond to events in RPC/Client server models, are the basic units of data delivered. Message-oriented communications are classified as either synchronous or asynchronous communication, and transient or persistent communication. In asynchronous communication, both the sender and the receiver are not required to be executing simultaneously. In synchronous communication, the sender process gets blocked while waiting for the receiver to reply to the message. Thus the sender and recipient are loosely-coupled. The number of times messages are stored, determines whether the communication is transient or persistent. Transient communication stores the message only while both partners in the communication are executing. If the next router or receiver is not available, then the message is discarded. In persistent communication, the message is stored until the recipient is active and receives it (Kim et al.).

1.1. Message-Oriented Transient Communication

A message is discarded by a communication server as soon as it cannot be delivered at the next server, or at the receiver. Typically, all transport-level communication services offer only transient storage. A communication server corresponds to a store-and-forward router.

Standards have been developed like Berkeley Sockets in 1970 and XTI, which stands for X/Open Transport Interface, which was formerly called Transport Layer Interface (TLI). Transient Communication is used in cases where messages need only be routed and not stored at a particular location. Typical usages of Transient Communication are in Transport layer level communication offered by a network router.

Communication occurs via endpoints called Sockets. A socket is a communication endpoint to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read. A socket forms an abstraction over the actual communication end point that is used by the local operating system for a specific transport protocol (Tanenbaum et al.).

Primitives like Bind, Listen, Send, Receive etc are used to operate on the sockets. It is usually based on the general-purpose protocols like TCP/IP for performing the communication.

1.2. Message-Passing Interface

MPI is a communication protocol used for programming parallel computers. It supports both point-to-point and collective communication. MPI is a message-passing API, protocol and semantic specifications for how its features must behave in any implementation. MPI's goals are to achieve high performance, scalability, and portability for parallel scientific applications. MPI is the most used messaging model in high-performance computing today. The MPI standard

specifies the syntax and semantics of a core of library routines that can be used to write portable message-passing programs in C, C++ etc.(Gropp et al.).

MPI provides a clearly defined base set of routines that can be efficiently implemented by the parallel hardware vendors. The hardware vendors can build upon these standard low-level routines to create high-level routines to be used in the distributed-memory communication environment of their parallel machines. MPI provides a simple portable interface for the basic user, but allows programmers to implement the high-performance message passing capabilities of advanced machines.

MPI is more abstract than sockets, and is hardware independent. It is better suited for proprietary high-speed interconnection networks and parallel applications. For MPI, proprietary communication libraries were created for high-level communication primitives.

MPI assumes communication takes place within a known group of processes. Each group is assigned an identifier. Each process within a group is also assigned a (local) identifier. A (group/D, process/D) pair therefore uniquely identifies the source or destination of a message, and is used instead of a transport-level address. There may be several, possibly overlapping groups of processes involved in a computation and that are all executing at the same time. (Tanenbaum et al.)

1.3. Message-Oriented Persistent Communication

A message-oriented persistent form of communication offers intermediate-term storage capacity for messages. They are generally implemented in the form of message-queuing systems or a Message-Oriented Middleware (MOM). They do not require either the sender or receiver to be active during message transmission. An important difference to socket-based and MPI based systems is that persistent communication systems are typically targeted to support message transfers that take minutes to complete, instead of seconds or milliseconds. Asynchronous model of communication is more suitable than synchronization model for application integration within resource allocation. Reliability of message delivery, decreased failures of message transmission and scalability of the middleware are important attributes for cross-platform communication in SOA applications systems.

MOM consists of applications which are made of loosely coupled components that communicate on geographically distributed networks. An advantage of this class of applications is asynchronous exchanges, heterogeneity, and scalability. MOM also provides an architecture that transmits messages and events into the widely distributed components of a service and connecting them by logically distributed concept.

2. Message Queues

A message queue allows for an asynchronous communication protocol, in which a process that places a message onto a message queue need not wait for an immediate response to continuing processing. One of the best examples of asynchronous messaging is an Email inbox. When an email is sent the sender can continue processing other things without an immediate response from the receiver. This method of messaging thus decouples the producer from the consumer. The producer and the consumer processes do not need to interact with the message queue at the same time.



Fig 1: A simple message queue

A message is stored and retrieved in a queue in a LIFO fashion. The sender of the message enqueues the message and a receiver removes or dequeues the message. Messages placed onto the queue are stored until the consumer retrieves them.

In a typical large scale message queuing system, each receiver application has its own queue. When the application wishes to send a message it stores it into the local Send queue, which is then placed into the global queue and then is stored at the local Receive queue of the receiver application.

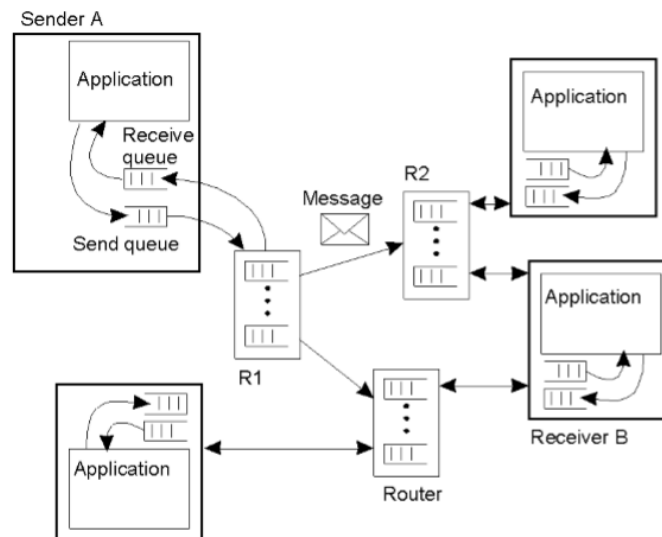


Fig 2: A Message Queuing System

2.1. Advantages of Message Queues

Message Queuing Systems offer a number of advantages as compared to other forms of communications like Remote Procedure Calls (RPCs) and Client Server Models.

1. As the messages are stored intermediately in a queue, the client need not block itself until receiving a response from the sender application. It can access the message when it is made available, from the queue directly.
2. The sender and receiver applications need not be active simultaneously. The messages are not lost and can be retrieved whenever the receiver application becomes active.
3. The user applications need not handle the routing of messages as it is automatically handled by the queue. Queue manager can decide which receiver applications the message needs to be delivered to.
4. Message Brokers handle transformation of messages so that applications can send/receive messages in their native formats. For example, a sender application may compose a message in csv format, but a receiver application can only read messages in json format. The message broker within the MQ system handles the conversion of such formats.

3. Publisher-Subscriber model

The publish/subscribe paradigm, as used in Message Oriented Middleware systems, is employed to let applications or users subscribe to topics of interest. Thus it does not require a simultaneous end-to-end connectivity. Publish/subscribe MOM (PSMOM) builds on asynchronous communication and enables data-centric communications in which consumers subscribe to the data they need and publishers make this data available (Hara et al.). Typically, messages are published to a broker on well-known topics by publisher clients, and the subscription on topics is made by subscriber clients. The broker also gives guarantees of delivery, so a message is not deleted until all subscribers to a topic have received the message from the broker, relieving the sender from the burden of ensuring delivery.

It is more of a data-centric approach than a system-centric approach as followed by a client-server system. The typical components include Publisher applications, some Subscriber applications, and a Notification system. Publishers disseminate messages in form of events and the subscribers get notified when the event is published. The notification management system maintains a database with all publishers and corresponding subscribers. (Wikipedia et al.)

Subscribers may register for specific messages at build time, initialization time or runtime. In GUI systems, subscribers can be coded to handle user commands (e.g., click of a button), which corresponds to build time registration. Some frameworks and software products use XML configuration files to register subscribers. These configuration files are read at initialization time. The most sophisticated alternative is when subscribers can be added or removed at runtime. This

latter approach is used, for example, in database triggers, mailing lists, and RSS. A subscriber application can subscribe to more than one topic in a single subscription.

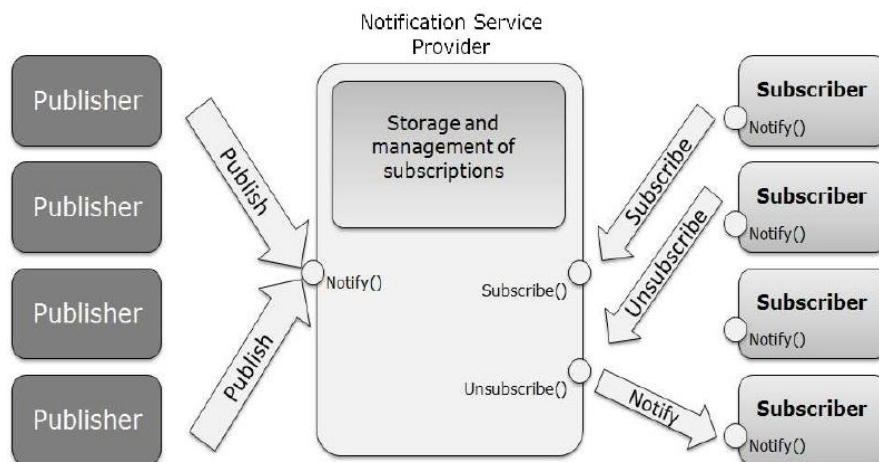


Fig 3: A Publisher-Subscriber and Notification system

3.1. Advantages of Publisher-Subscriber Model

The separation of concerns in the system design of a publisher-subscriber model, brings the following benefits (Wikipedia et al.) -

1. Loose coupling between sender and receiver.
 - a. Space decoupling: Publishers and subscribers do not need to know each other.
 - b. Time decoupling: Publishers and Subscribers need not be running at same time.
 - c. Synchronization decoupling: Application operations are not halted during publishing and receiving.
2. Allows scalability, as there is no dependency between applications, any number of Publisher and subscriber applications can be added to the system.
3. A Subscriber shows interest in a particular topic and receives only those notifications filtered based on that. It does not have to handle messages that are of no use to it.

4. RabbitMQ

RabbitMQ is an open-source message-oriented middleware based on Erlang language which is especially suited for distributed applications, as concurrency and availability are well-supported. It runs on almost all major platforms (at least almost all places where Erlang/OTP runs) and can be used build applications using different languages like Java, C++, .Net, Python, Ruby etc.

It is lightweight and can be easily deployed in public and private clouds. It allows distributed deployment as clusters for high availability and throughput federates across multiple availability zones and regions. It provides bindings support for multiple protocols like HTTP, STOMP, SMTP, POP3 etc. RabbitMQ offers high reliability, availability and scalability. It also provides a Management application for easy maintenance of exchanges and queues. (RabbitMQ et al.)

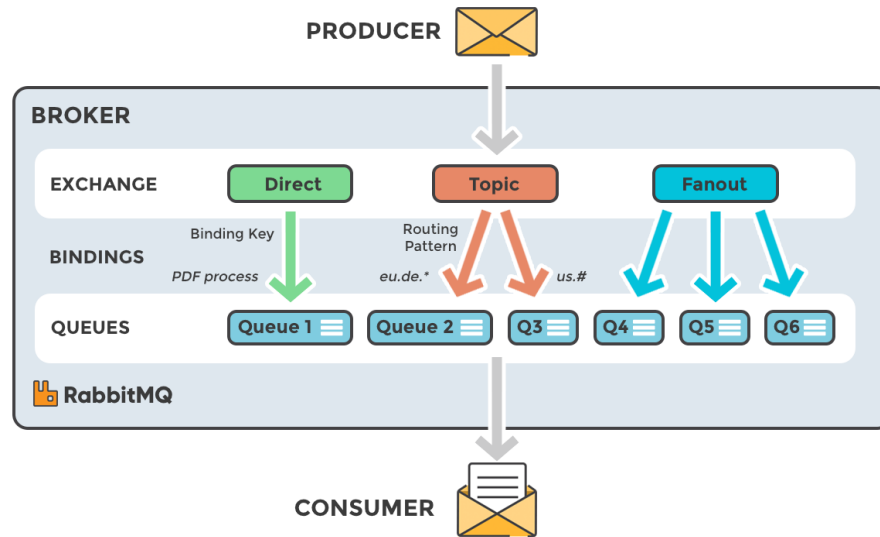


Fig 4: Different types of exchanges in RabbitMQ

The Queue specifics of RabbitMQ allows multiple consumers to be configured for a single queue, and they all get mutually exclusive messages; supports multiple types of queues such as direct, fan-out, etc. so semantics such as broadcast to multiple clients listening on multiple queues is achieved.

The different type of exchanges in RabbitMQ are -

- **Direct** - The Direct exchange type is used to distinguish messages published to the same exchange using a simple string identifier. It sends the message to subscribers based on a single string.
- **Fanout** - The Fanout exchange type routes messages to all bound queues indiscriminately.
- **Topic** - The Topic exchange type routes messages to queues whose routing key matches all, or a portion of a routing key.

5. Problem Statement

The important objectives of this implementation project are as follows :-

- To implement a publisher/subscriber model using RabbitMQ.

- Will be used for a News/ Weather /Sports alerts publishing application.
- Will also have a user registration and feedback publisher.
- Applications subscribed to the topic of their interest will get the corresponding message. User registration subscriber will also send a mail welcoming the user.

6. Project Environment

The application has been created using multiple tools and frameworks.

- Message Broker - RabbitMQ Server 3.7.0
- Language - Java (JRE 1.8)
- Framework - Spring
- Database - MySql 5.7.20
- Development tool - Spring Tool Suite 3.9.1
- Email Service - Gmail
- Application Server - Apache Tomcat

7. Project Technical Architecture

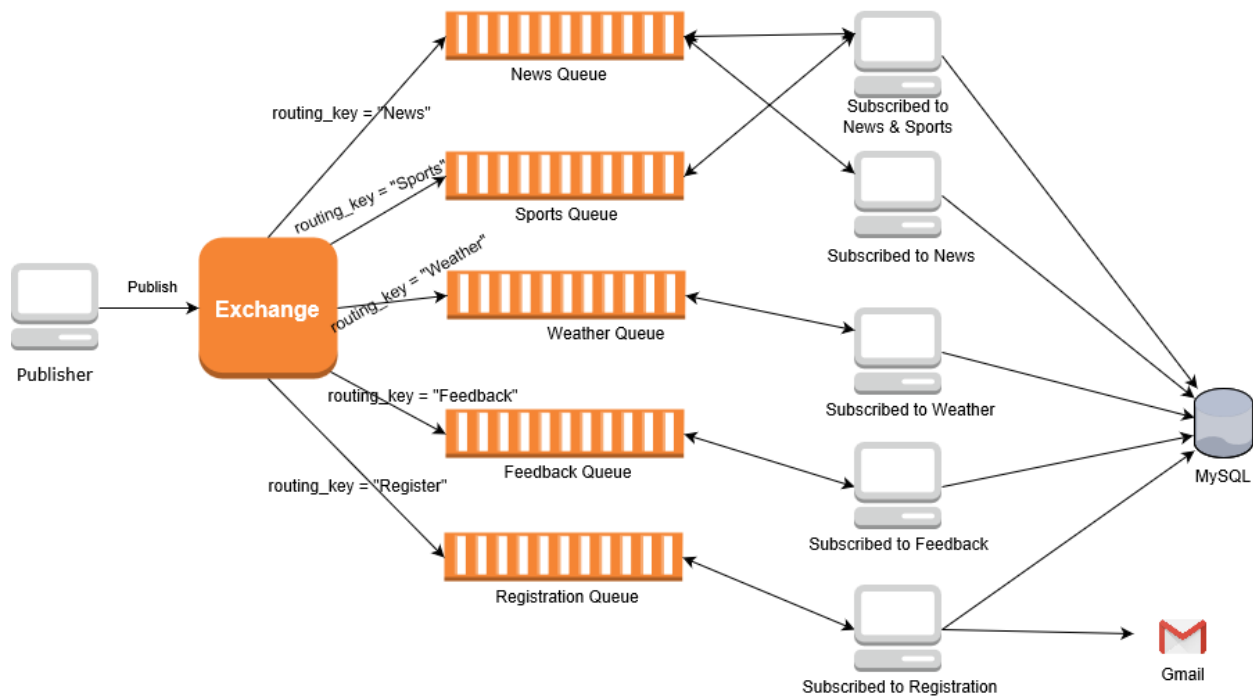


Fig 5: Technical architecture

Our project application consists of the following components -

1. **Publisher** - Application that can publish alerts related to News/Sports/Weather. It can also publish a User Registration message and a Feedback message.
2. **Exchange** - It is a component of RabbitMQ that routes the message to particular Queues as per the routing keys provided by Publisher.
3. **Queues** - It is a component of RabbitMQ that temporarily stores the published message, until an active subscriber reads from it.
4. **Subscribers** - Application that is subscribed to a particular alert like News or Sports or Weather. Another subscriber receives notifications related to Feedback. Another subscriber application receives User registration information and sends a welcome mail using Gmail. All the subscribers save received messages in a local database and read from the database before displaying on the screen.
5. **MySQL database** - Stores all the alerts, feedbacks and User registration information received by subscribers.

8. Project Screenshots

8.1. Publisher application

Following are the different screens in our Publisher application -

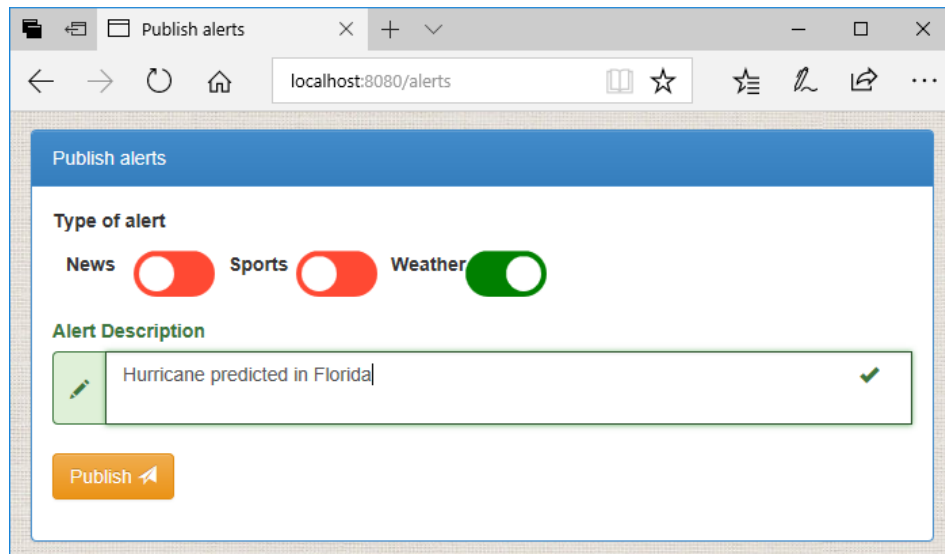


Fig 6: Publish alerts related to News/Sports/Weather

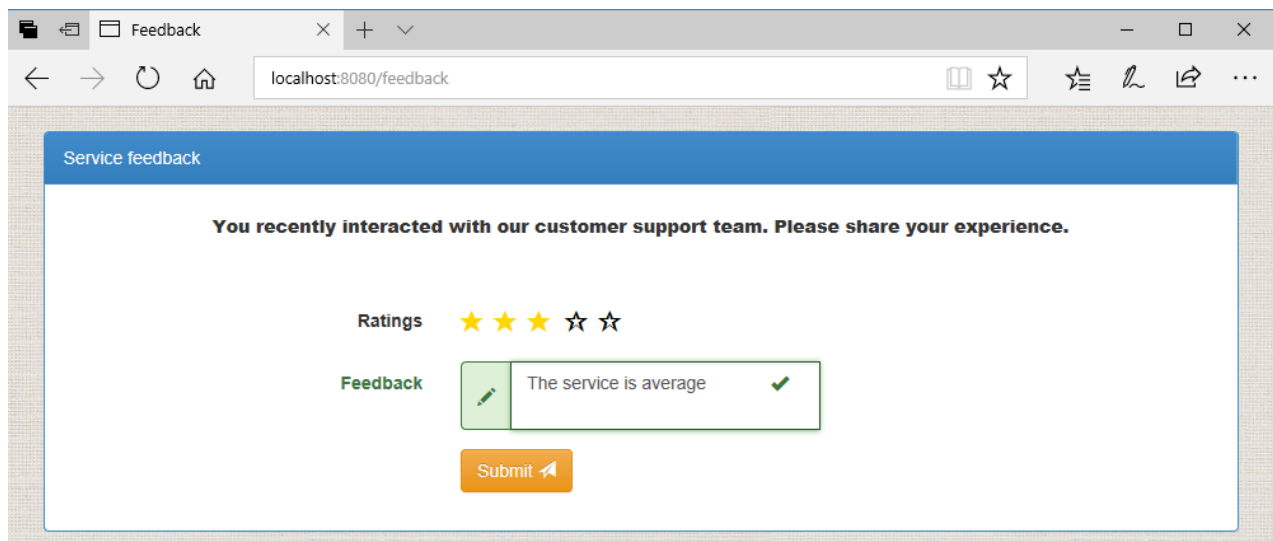


Fig 7: Publish feedback ratings and message

Register Now!

First Name: Jithin ✓

Last Name: Eapen ✓

E-Mail: jithin.john@csu.fullerton.edu ✓

Last 4 digits of SSN: 8282 ✓

Address: 1942, N Deerpark Dr ✓

City: Fullerton ✓

State: California ✓

Zip Code: 92831

Register

Fig 8: Publisher application to register new users

8.2. Subscriber applications

Following are the different screens in our Subscriber application -

Alerts Summary

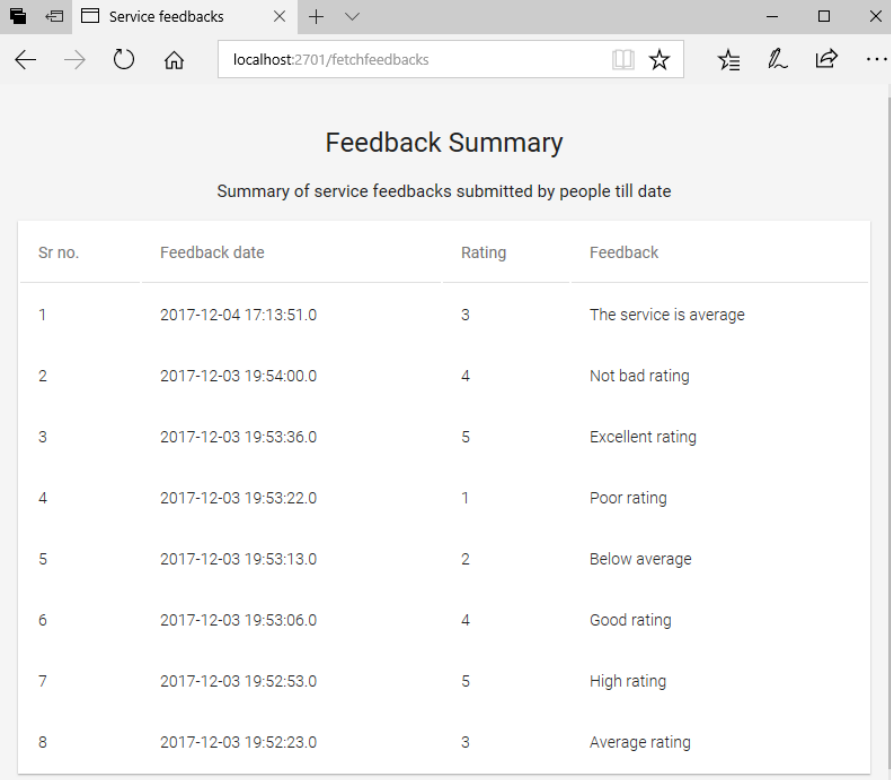
Summary of alerts and their subscribers

First Subscriber		
Alert date	Type	Description
2017-12-04 04:37:58.0	news	North Korea tests new missiles
2017-12-03 19:54:31.0	sports	Hockey match scheduled on Saturday
2017-12-03 19:50:14.0	sports	Titans vs UCLA basetball match on Friday
2017-12-03 19:47:53.0	sports	Virat Kohli gets 6 double century

Third Subscriber		
Alert date	Type	Description
2017-12-04 17:04:43.0	weather	Hurricane predicted in Florida
2017-12-03 19:50:27.0	weather	Cyclone in India

Second Subscriber		
Alert date	Type	Description
2017-12-04 04:37:58.0	news	North Korea tests new missiles

Fig 9: Subscriber application to display alerts subscribed by it



Sr no.	Feedback date	Rating	Feedback
1	2017-12-04 17:13:51.0	3	The service is average
2	2017-12-03 19:54:00.0	4	Not bad rating
3	2017-12-03 19:53:36.0	5	Excellent rating
4	2017-12-03 19:53:22.0	1	Poor rating
5	2017-12-03 19:53:13.0	2	Below average
6	2017-12-03 19:53:06.0	4	Good rating
7	2017-12-03 19:52:53.0	5	High rating
8	2017-12-03 19:52:23.0	3	Average rating

Fig 10: Subscriber application displays list of Feedbacks

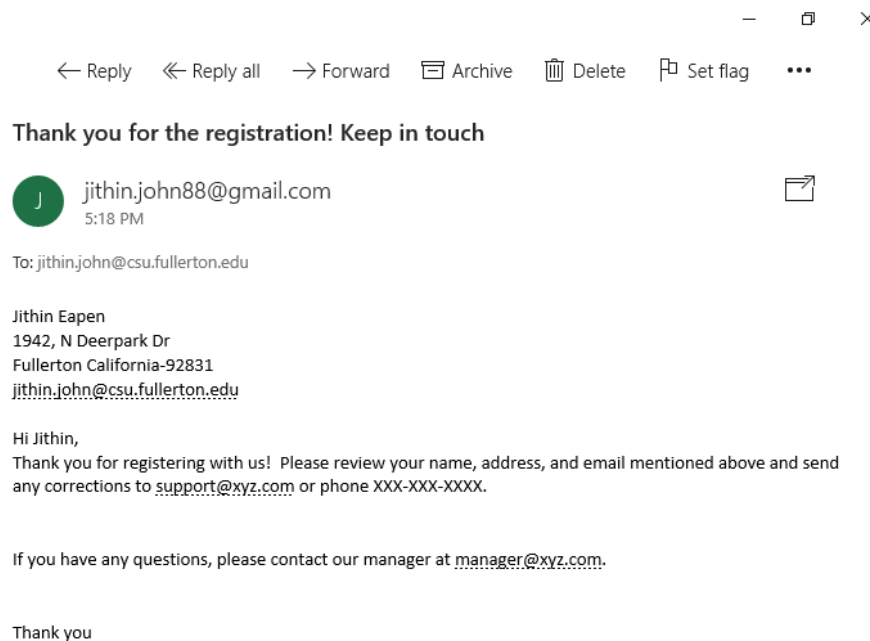


Fig 11: Subscriber application sends mail to new user welcoming him

9. Conclusions

We have studied the different types of Message-Oriented Communication methods, and have understood the advantages of Message Queues and Publisher-Subscriber model of distributed systems.

We have implemented an application based on the Publisher-Subscriber model for News, Sports, Weather, Registration and Feedback alert, using an open-source message broker RabbitMQ to route the messages from Publisher to the correct Subscriber.

10. References

- Gropp, W., Lusk, E.:** Using MPI-2 - Advanced Features of the Message-Passing interface. The MIT Press.
- Tanenbaum, A.S., Steen, M.V.:** Distributed Systems - Principles and Paradigms, Second Edition, PHI Learning.
- Kim, J.:** Dissertation on Message-Oriented Communication. <http://kjk.skhu.ac.kr/MOM.html>.
- Hara, J.O.:** Toward A Commodity Enterprise Middleware, ACM Queue.
- Wikipedia:** Publish subscribe pattern. https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern.
- RabbitMQ:** Features of RabbitMQ. <https://www.rabbitmq.com/features.html>.