International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 8 | ISSN : 2456-3307
DOI : https://doi.org/10.32628/CSEIT183858

Comparison of JMS Products

P. Xavier Jeba[1], P. Mercy[2], J. Arockia Mary[3]
[1]Assistant Professor, Department of Computer Science, Holy Cross College, Tiruchirappalli, Tamil Nadu, India
[2]Department of Computer Science, Holy Cross College, Tiruchirappalli, Tamil Nadu, India

ABSTRACT

Before the introduction of Java Messaging Service API (JMS),developers had to absorb the complex branded APIs of each specific messaging server from different supplier and this made creating messaging applications hard and gave rise to minor transportability. Additionally, messaging servers from different suppliers were not interoperable and ended in supplier limitations. JMS API is not a product but it is a Java specification for messaging middleware from Sun and its partners. JMS defines an enterprise messaging Java API that enables creating business applications effortlessly that can interchange business data and events not occurring at the same time and reliably in a supplier doubting manner. The messaging server suppliers offer the service provider interface that assists the standard JMS API. There are quite a lot of suppliers producing messaging products which promote JMS API and choosing a certain messaging product needs to evaluate the merits and demerits of the product.

Keywords: JMS, point-to-point, Publisher/Subscriber

I. INTRODUCTION

This paper provides a detail portrayal of Messaging and Java support for Messaging via the Java Messaging Service (JMS) API. This paper also provides a wide comparison of popular Messaging servers widely used in real time applications.

Messaging Definition:

- Messaging is a mode of communication between software applications
- A messaging system works on a peer-to-peer basis.
- In a messaging system there will be Sender (also known as Producer or Client) and Receiver (also known as Consumer).
- Each client links to a messaging agent that provides services for creating, sending, receiving, and reading messages
- Messaging enables distributed communication that is loosely coupled.
- Java Messaging Service (JMS) API Definition:
- The JMS is a Java API that lets software applications to create, send, receive, and read messages.
- It was designed by Sun and several partner companies and defines a common set of interfaces and semantics that let Java applications to communicate with other messaging implementations.
- It maximizes the portability of JMS applications across several JMS providers in the same messaging domain.
- It enables communication among software applications that are loosely couples but also makes it asynchronous and reliable.

JMS Usage Provides The Following :

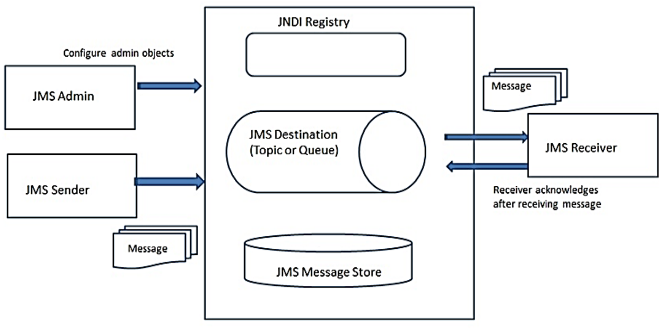
- It lets a client component (Sender) to send information to another (Receiver) and to carry on operating without waiting for immediate response from the Receiver.
- It provides loose coupling between message Sender and Receiver compared to a tightly coupled Remote Procedure Call (RPC).
- JMS API improves the Java Enterprise Edition (JEE) platform by simplifying enterprise software applications development, letting loosely coupled, asynchronous, reliable interactions among JEE components and legacy systems capable of messaging.
- JMS API support following messaging styles –
- Point-To-Point (PTP) whichis also known as Queue
- Publish/Subscribe (pub/sub) which is also known as Topic
- It enables communication between diverse mechanisms of a distributed application.

JMS API Architecture:

A JMS application has the following parts-

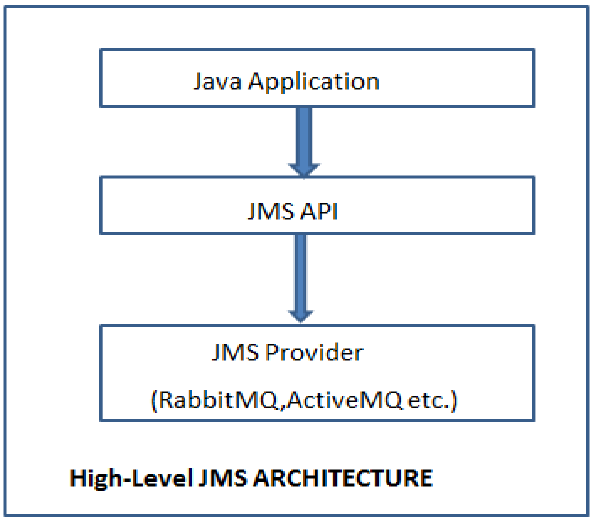
- JMS Provider: It is a messaging system that implements the JMS interfaces and provides administrative and control features. E.g.: Rabbit MQ
- JMS Clients: They are the Java software programs that produce and consume messages.
- Messages: They are the objects that communicate information between JMS clients (Sender and Receiver)

Administered Objects: They are preconfigured JMS objects created by an administrator for the use of clients, in the JMS system (E.g.: Rabbit MQ) of the JMS provider. The two kinds of JMS administered objects are Destinations and connection factories.

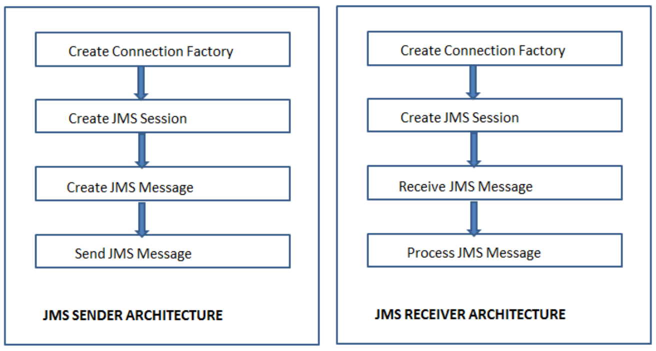


JMS ARCHITECTURE

High-Level JMS Architecture

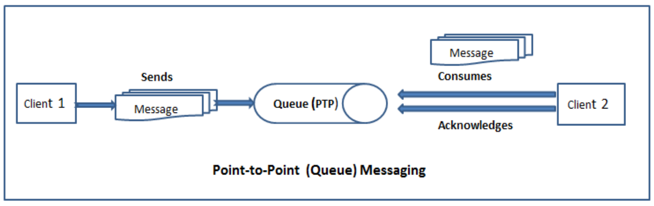


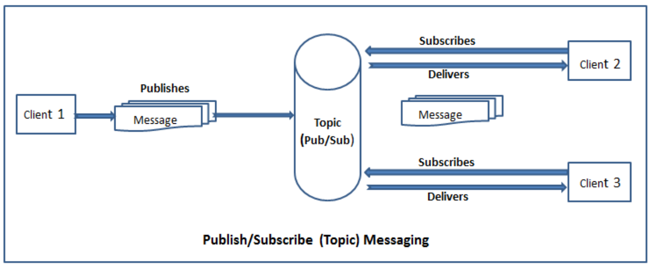
JMS Sender and Receiver Architecture



Difference Between Point-To-Point (Queue) And Publish/Subscribe (Topic):

There are two types of message models which are the point-to-point model and publish or subscribe model. The point-to-point model is also known as Queue and the publisher or subscriber model is also known as Topic.





The following table depicts the differences between Queue and Topic models:

Queue	**Topic**
Point-to-point model	Publish/subscribe model
Only one receiver gets the message	Multiple receivers subscribe to the message
Messages will be delivered in the order sent	Messages will not be delivered in the order sent
Each message is processed only once in Queue	Each message is processed multiple in Topic
Consumer client acknowledges on receiving message	Consumer client does not acknowledges on receiving
Queue knows theconsumer of the message.	Topic does not know theconsumer of the message.

JMS Message Brokers:

There are several message brokers (sever) available which fully implement JMS API. Each message broker has pros and cons.We will compare the following two JMS compliant message products

- RabbitMQ
- Apache Kafka

	RabbitMQ	**Apache Kafka**
Open source	Yes	Yes
Language written	Erlang	Scala and Java
Protocol	TCP	TCP
Scheduled Message	Supported	Not Supported
Message Storage	Uses a custom database to store the messages	Distributed and partitioned based storage. Messages in partition represented as a log stream.
Message Filter		Supported
JMS Support	Yes	Yes
Push/Pull model	Push model	Pull model
Partition support	No	Kafka can divide among Consumers by partition and send those message/records in batches.
Maximum messages handling	Tests show about 20,000 messages handled per second on a single server	Tests show about 1,00,000 messages handled per second on a single server
Consumer type	For slow consumers	For fast and slow consumers
Clustering and replication support	Partial support	Full support
Design purpose	It is designed as a general purpose message broker	It is designed for high volume publishes-subscribe messages and streams, meant to

		be fast, scalable and durable.
Responsibility of reading the messages	Consumer does not take the responsibility of reading the messages	Consumer has the responsibility of reading the messages.
Read and unread message retention.	It tracks which messages were read by each consumer and retains unread messages for a set amount of time	It does not attempt to track which messages were read by each consumer and retains both read and unread messages for a set amount of time

II. CONCLUSION

The above table provides a wide comparison between RabbitMQ and Apache Kafka. It also provides listing of the various features, advantages and limitations. Based on the above comparison we conclude that, Apache Kafka has more advantages over the RabbitMQ, because it is fast, scalable and durable. RabbitMQ could be recommended for applications which are slow and fewer messages are handled while Apache Kafka is recommended for applications which require handling fast and larger messages.

III. WEB REFERENCES

[1]. https://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html
[2]. https://www.rabbitmq.com/#features
[3]. https://kafka.apache.org