Message Queueing Concept and RabbitMQ

Prepared By Enes Erten

Content

- Message Queueing concept and Terminology about It
- Advanced Message Queuing Protocol (AMQP)
- RabbitMQ and Message Queueing
- RabbitMQ manual installation and configurations
- RabbitMQ Docker installation and configurations
- Controlling RabbitMQ on Linux and Docker

Message Queueing Concept

- **Message:** In message queuing, a message is a collection of data sent by one program and intended for another program. There lots of message types, following them are fundamental and most used ones.

Datagram : Do not require a reply from the application that receives the message (that is, gets the message from the queue). An example of an application that might use datagrams is one that displays flight information in an airport lounge. A message might contain the data for a whole screen of flight information. Such an application is unlikely to request an acknowledgment for a message because it probably does not matter if a message is not delivered. The application sends an update message after a short time.

Message Queueing Concept

Request Messages: Use a *request message* when you want a reply from the application that receives the message. An example of an application that could use request messages is one that displays the balance of a checking account. The request message could contain the number of the account, and the reply message would contain the account balance.

Reply Messages: Use a *reply message* when you reply to another message. It is mostly used with request messages. An example assume you want to pay out with your credit card the process is start with sending your information's such as your bank id and price you want to spend to the Business Support System (BSS) of bank and it control the transaction is ok (enough money in the account etc.) if it is ok it is sending a successful reply message or vice versa.

Message Queueing Concept

Report Messages: Inform applications about events such as the occurrence of an error when processing a message.

- exception report message
- expiry report message
- confirmation of arrival (COA) report message
- confirmation of delivery (COD) report message
- positive action notification (PAN) report message
- negative action notification (NAN) report message

Message Queueing Concept

- **Message Descriptor:** The control information is defined in a message descriptor structure (MQMD) and contains such things as: The type of the message, an identifier for the message, the priority for delivery of the message.
- **Queue:** A named destination to which messages can be sent. Messages accumulate on queues until they are retrieved by programs that service those queues. There are two types of queue, predefined queue and dynamic queue. Suitably authorized applications can retrieve messages from a queue according to the following retrieval algorithms, First-in-first-out (FIFO), Message priority, as defined in the message descriptor. Messages that have the same priority are retrieved on a FIFO basis, a program request for a specific message.
- **Queue Manager:** A *queue manager* is a system program that provides queuing services to applications. It provides an application programming interface so that programs can put messages on, and get messages from, queues. A queue manager provides additional functions so that administrators can create new queues, alter the properties of existing queues, and control the operation of the queue manager.

Message Queueing Concept

- **Channels:** Objects that provide a communication path from one queue manager to another. Channels are used in distributed queuing to move messages from one queue manager to another and they shield applications from the underlying communications protocols. The queue managers might exist on the same, or different, platforms.
- **Message channel agent:** A message channel agent is one end of a channel. A pair of message channel agents, one sending and one receiving, make up a channel and move messages from one queue manager to another.

Message Queueing Concept

- In past communication through indirect program-to-program communication done by connecting programs to a network channel. It has lots of Drawbacks;

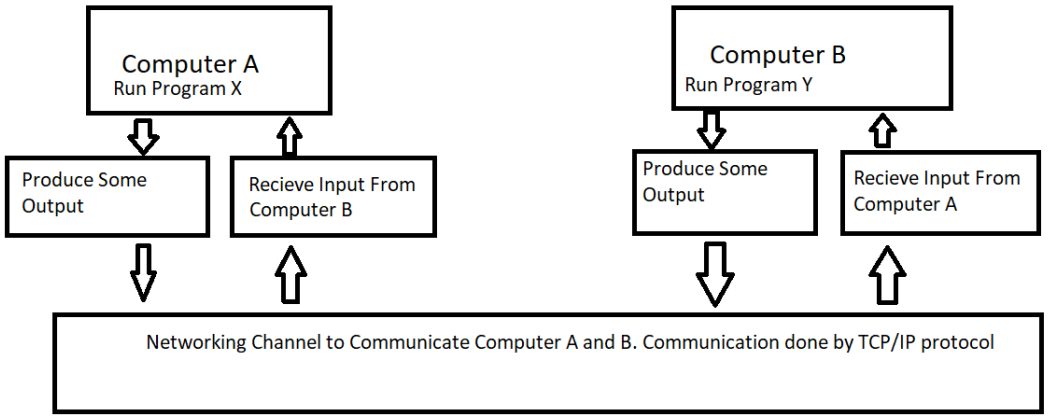
  - It would need to hear a reply before it could move on to the next task

  - If on of the software or computer down it would try to communicate repeatedly

  - Sending data size is limited to the channel circumstances. It means limited data can be transferred.

  - Networking channel need maintenance and more equipment's it would cost money and more workload .

Message Queueing Concept

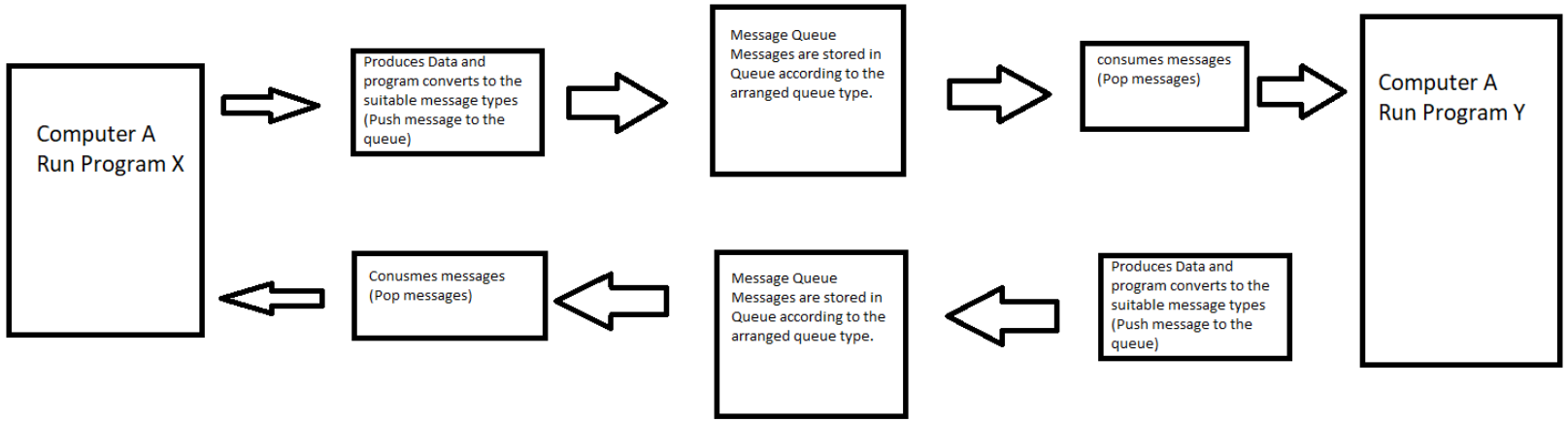


Tradition Communication indirect program to program communication. Assume Computers A and B are running software components and working together

Message Queueing Concept

- Message Queue sits in between the two services that need to communicate with one another. So, with a message queue, A software can add a message to the queue and immediately move on to the next task. And then similarly, the communicated software can consume from the queue, process the message and then immediately consume the next message.

Message Queue Concept



Message Queue Concept

- Main Features are

  - There are no direct connections between programs.

  - Communication between programs can be independent of time.

  - Work can be carried out by small, self-contained programs.

  - Communication can be driven by events.

  - Applications can assign a priority to a message.

  - Security.

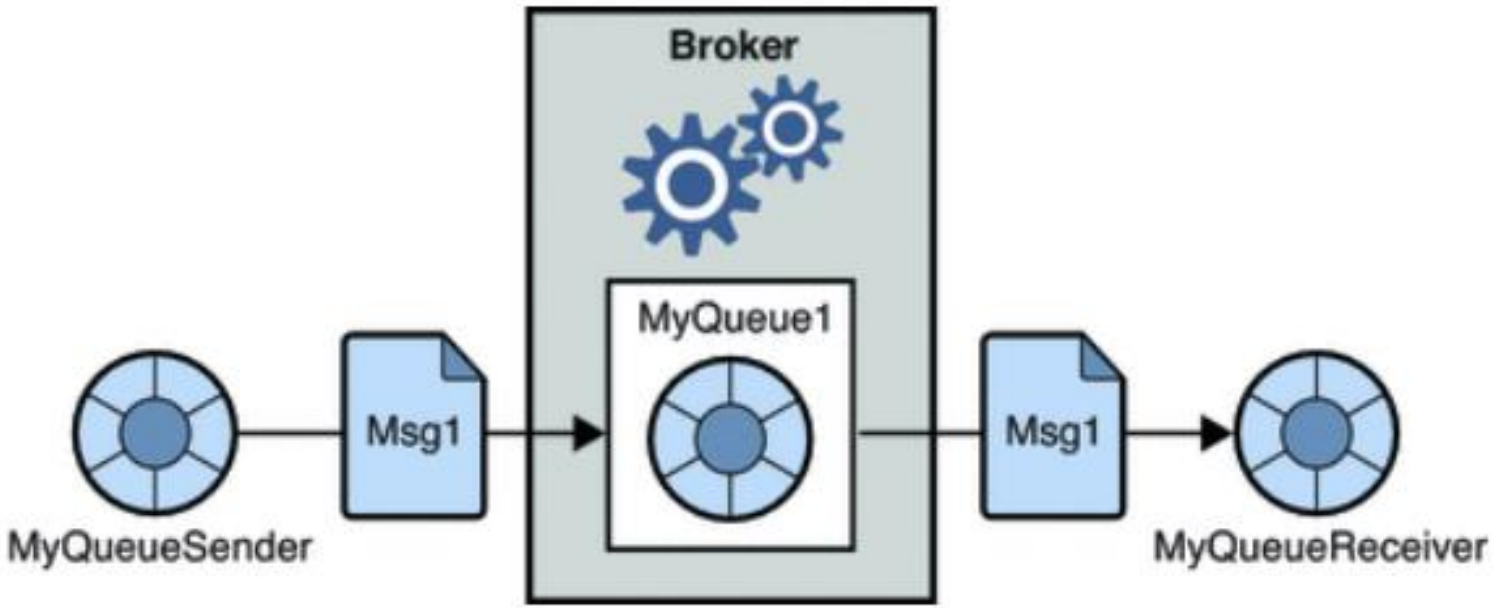
  - Data integrity.

  - Recovery support.

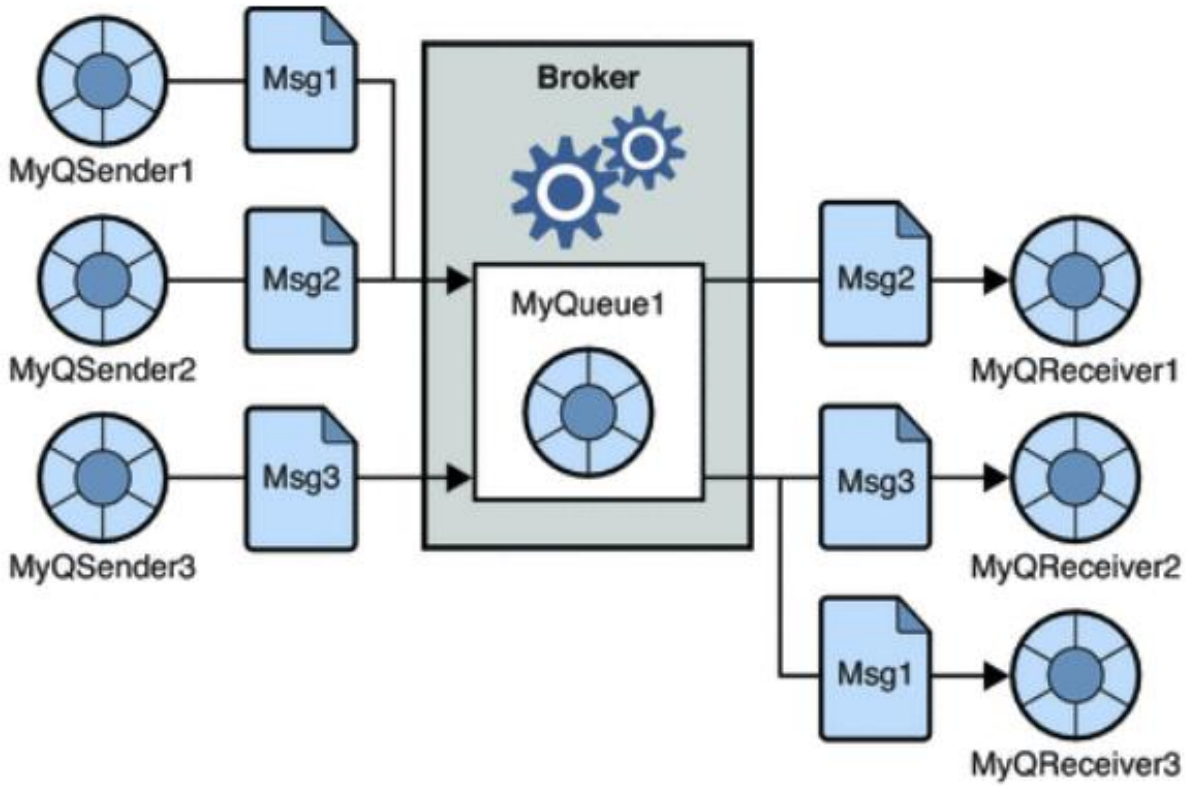
Message Queue Concept

- Point to point messaging: In the point-to-point domain, message producers are called senders and consumers are called receivers. They exchange messages by means of a destination called a queue: senders produce messages to a queue; receivers consume messages from a queue. What distinguishes point-to-point messaging is that a message can be consumed by only one consumer.

Message Queue Concept



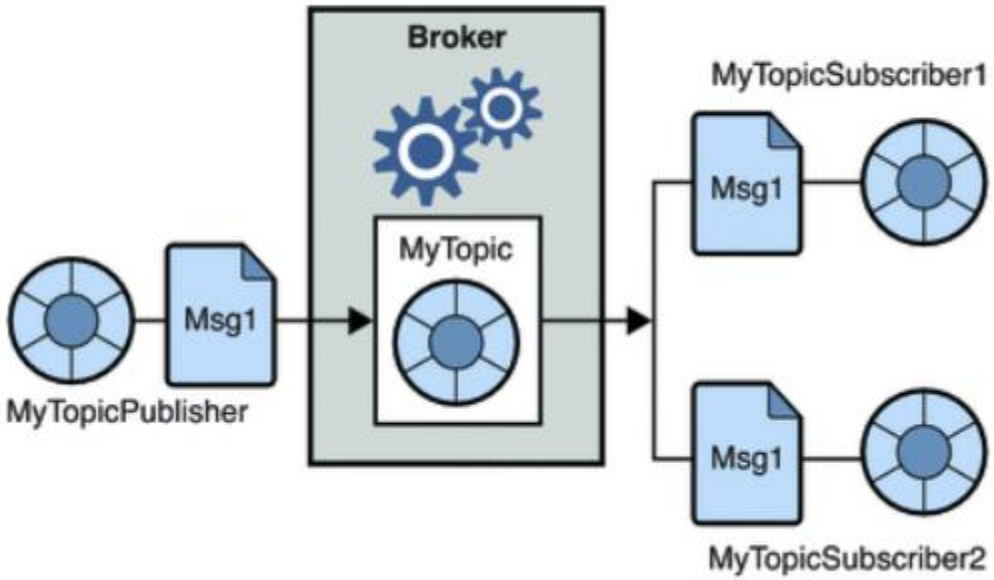
Message Queue Concept



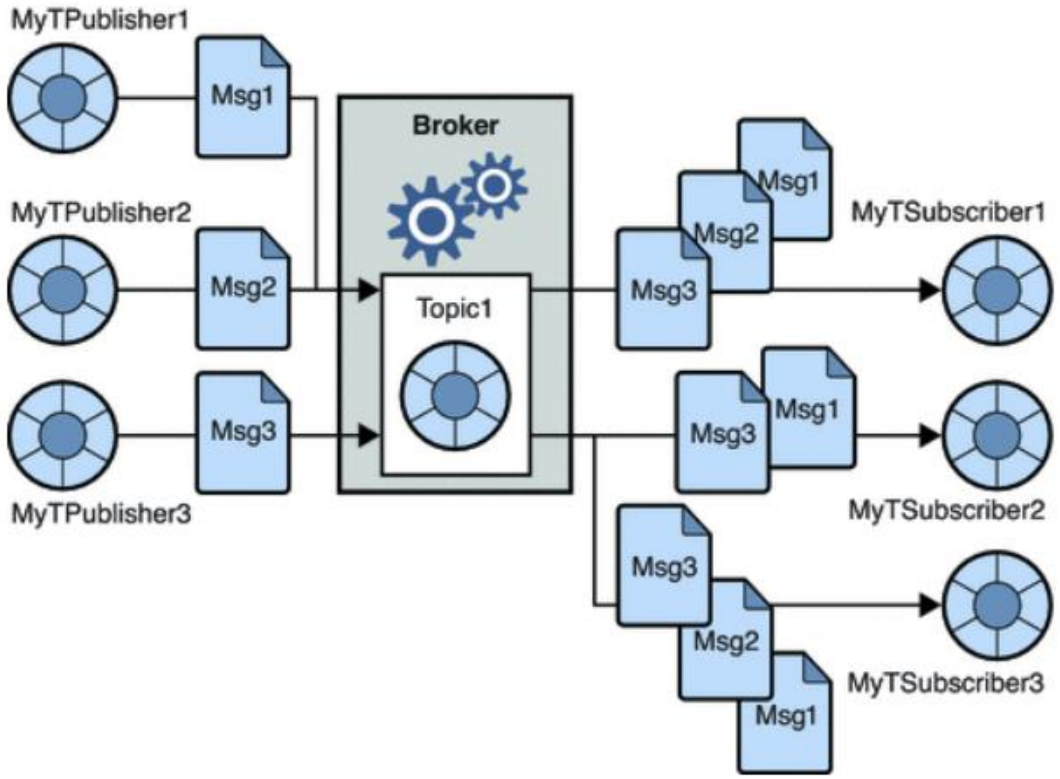
Message Queue Concept

- In the publish/subscribe domain, message producers are called publishers and message consumers are called subscribers. They exchange messages by means of a destination called a topic: publishers produce messages to a topic; subscribers subscribe to a topic and consume messages from a topic. While the publish/subscribe model does not require that there be more than one subscriber, two subscribers are to emphasize the fact that this domain allows you to broadcast messages. All subscribers to a topic get a copy of any message published to that topic.

Message Queue Concept



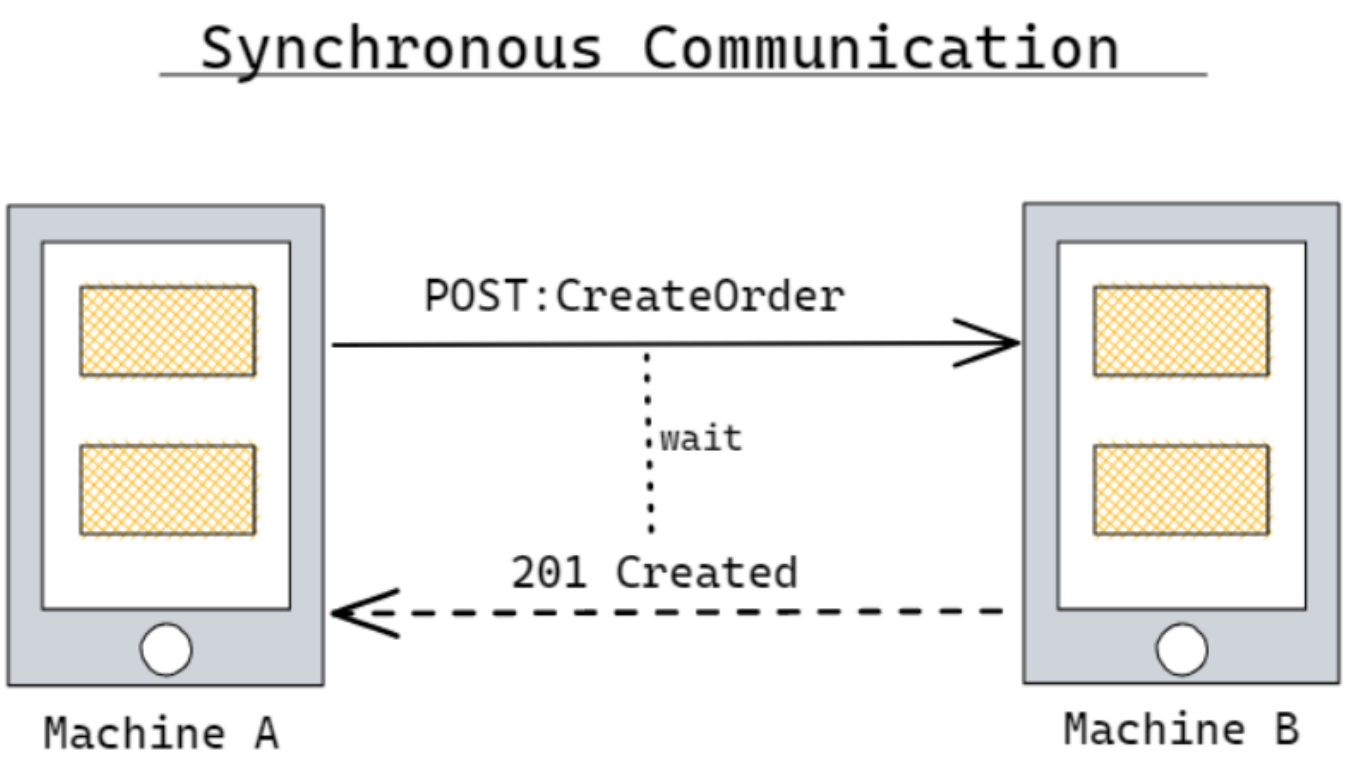
Message Queue Concept



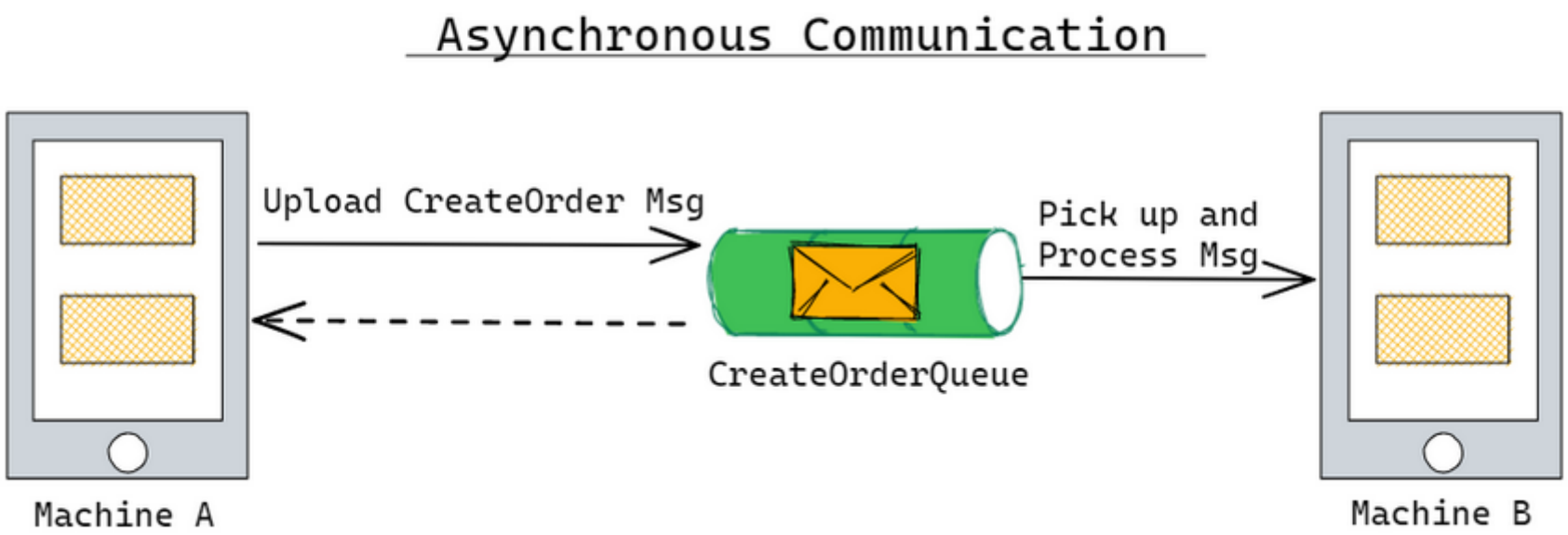
Message Queue Concept

- When having an IT landscape consisting of multiple interacting applications, it being microservices or not, it is common to implement asynchronous communication to minimize or eliminate tight coupling between the applications. Tight coupling between applications is often caused by synchronous communication, which is when one application is calling another and waiting until it gets a response. But when using asynchronous communication, in the form of messaging, the interaction is one-way. This prevents the machine from publishing the message to get a response directly from the machine processing the message, as part of the same flow. In situations where you just want to hand off some processing of a message to another application and you do not care about the response or result, then all is good. But if the sender needs to act on something or needs to get the result of the processing when it has finished, another way of providing the result must be implemented.

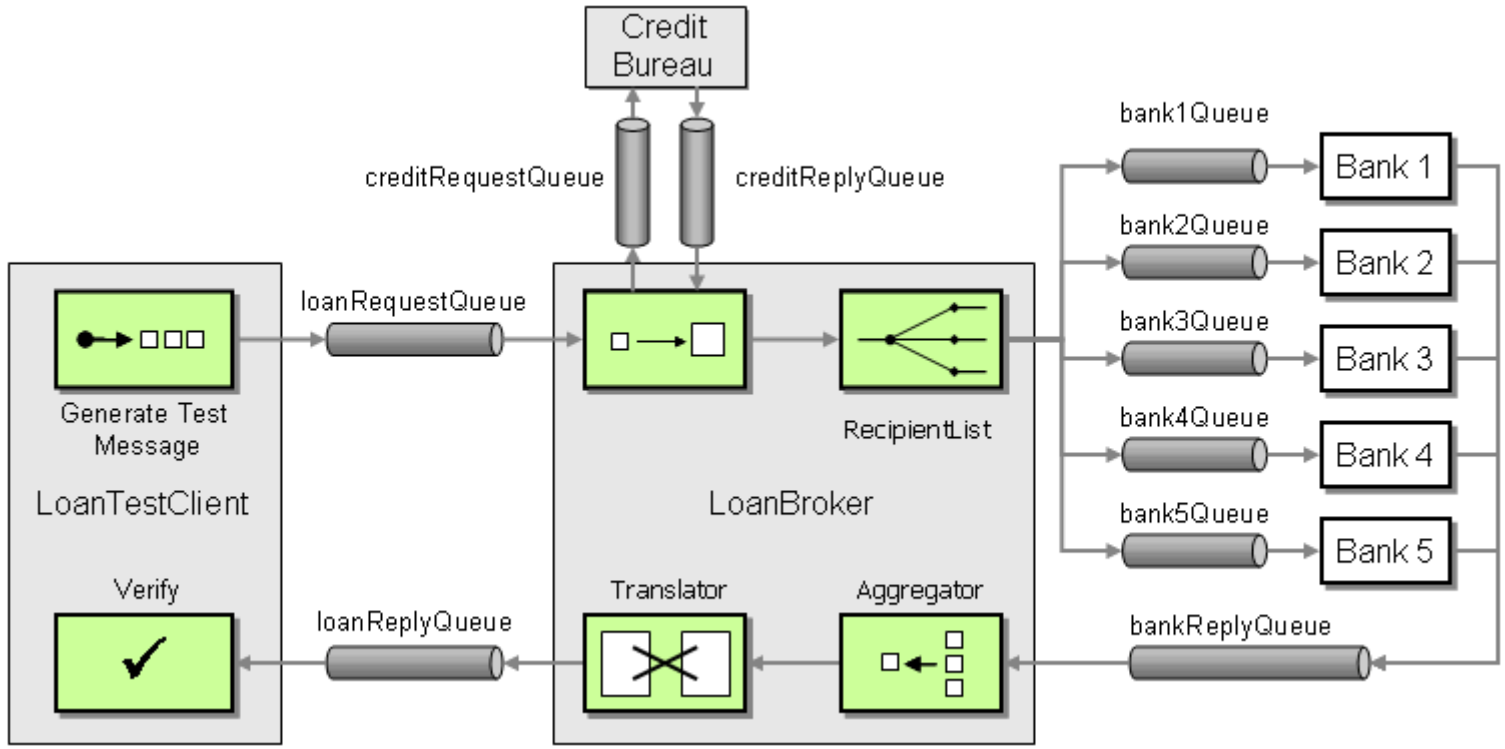
Message Queue Concept



Message Queue Concept



Message Queue Concept



Advanced Message Queueing Protocol

- The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security. AMQP mandates the behavior of the messaging provider and client to the extent that implementations from different vendors are interoperable, in the same way as SMTP, HTTP, FTP, etc. have created interoperable systems. Previous standardizations of middleware have happened at the API level (e.g. JMS) and were focused on standardizing programmer interaction with different middleware implementations, rather than on providing interoperability between multiple implementations. Unlike JMS, which defines an API and a set of behaviors that a messaging implementation must provide, AMQP is a wire-level protocol. A wire-level protocol is a description of the format of the data that is sent across the network as a stream of bytes. Consequently, any tool that can create and interpret messages that conform to this data format can interoperate with any other compliant tool irrespective of implementation language.

Advanced Message Queueing Protocol

- AMQP is a binary, application layer protocol, designed to efficiently support a wide variety of messaging applications and communication patterns. It provides flow controlled, message-oriented communication with message-delivery guarantees such as at-most-once (where each message is delivered once or never), at-least-once (where each message is certain to be delivered but may do so multiple times) and exactly-once (where the message will always certainly arrive and do so only once), and authentication and/or encryption based on SASL and/or TLS. It assumes an underlying reliable transport layer protocol such as Transmission Control Protocol (TCP). The AMQP specification is defined in several layers: (i) a type system, (ii) a symmetric, asynchronous protocol for the transfer of messages from one process to another, (iii) a standard, extensible message format and (iv) a set of standardised but extensible 'messaging capabilities.

RabbitMQ and Message Queueing

- RabbitMQ is an open-source message-broker software (sometimes called message-oriented middleware) that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), MQ Telemetry Transport (MQTT), and other protocols. Written in Erlang, the RabbitMQ server is built on the Open Telecom Platform framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages. The source code is released under the Mozilla Public License.

Resources and Useful Links

- What is message queue -> https://www.youtube.com/watch?v=xErwDaOc-Gs
- What is RabbitMQ -> https://www.youtube.com/watch?v=7rkeORD4jSw
- Introduction to Message queueing? -> https://www.ibm.com/docs/en/ibm-mq/9.0?topic=overview-introduction-message-queuing
- Main features and benefits of message queuing -> https://www.ibm.com/docs/en/ibm-mq/9.0?topic=queuing-main-features-benefits-message
- Message queuing terminology -> https://www.ibm.com/docs/en/ibm-mq/9.0?topic=queuing-message-terminology
- Messages and queues -> https://www.ibm.com/docs/en/ibm-mq/9.0?topic=queuing-messages-queues

Resources and Useful Links

- RabbitMQ tutorial -> https://www.rabbitmq.com/getstarted.html
- Messaging Systems -> https://docs.oracle.com/cd/E19316-01/820-6424/aerap/index.html
- Point-to-point messaging -> https://docs.oracle.com/cd/E19316-01/820-6424/aerbj/index.html
- Publish/Subscribe messaging -> https://docs.oracle.com/cd/E19316-01/820-6424/aerbk/index.html