AMQP makes the "**Difficult**" become "**Practical**"

- All AMQP clients interoperate with all AMQP servers
- Diverse programming languages can communicate easily
- Legacy message brokers can be retrofitted to remove proprietary protocols from your networks
- Enables messaging as a cloud service
- Advanced publish-and-subscribe
- Transactional messaging functionality

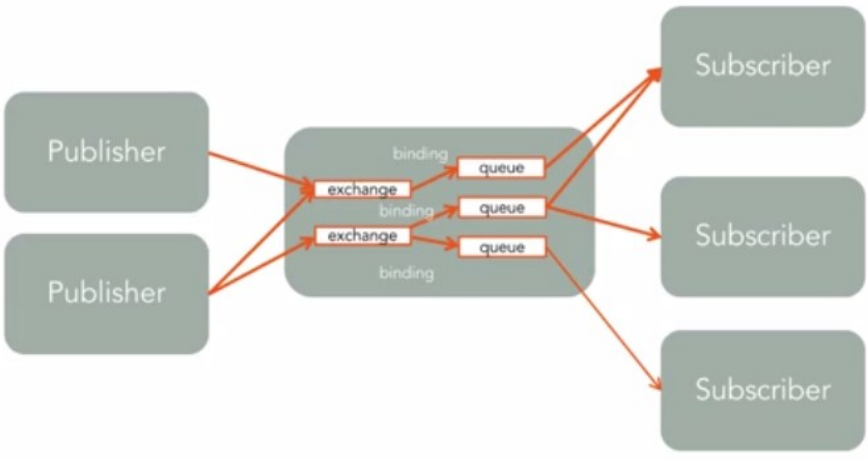
## Use Cases for AMQP

- Want a real time feed of constantly updating information? **No problem.**
- Want an encrypted assured transaction? **No problem.**
- Want your message to be delivered when the destination comes online? **No problem.**
- Want to send an enormous message while still receiving status updates over the same network connection? **No problem.**
- Want things to work on all popular operating systems and languages? **No problem.**

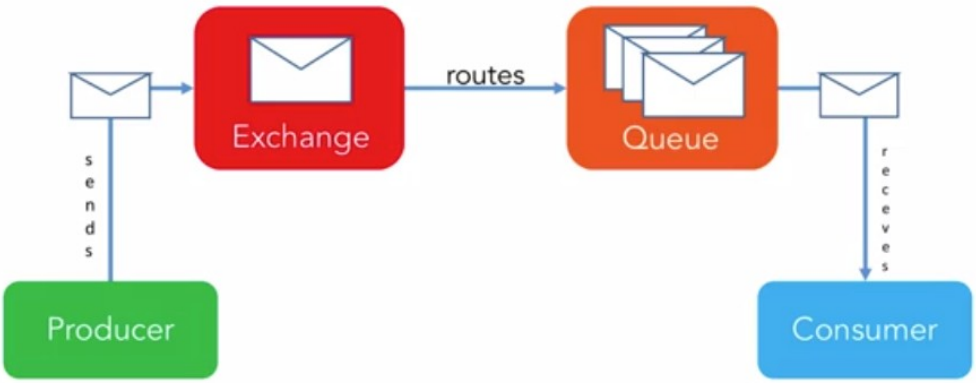
# What is RabbitMQ?

- A powerful Open Source message broker (message-oriented middleware)
- Most popular implementation of AMQP!
- Provides a robust and flexible messaging platform designed to interoperate with other messaging systems
- Developed using Erlang programming language so it boasts on its high throughput and low latency.
- Supports clustering for fault tolerance and scalability
- Protocol defines exchanges, queues and bindings
- Allows multiple connection channels inside a single TCP connection in order to remove the overhead of opening a large number of TCP connections to the message broker

# RabbitMQ: A Design Overview



# 4 Actors of Messaging with RabbitMQ




# Exchanges


- Actual AMQP elements where messages are sent at first
- Takes a message and routes it into one or more queues
- Routing algorithm decides where to send messages from exchange
- Routing algorithms depends on the exchange type and rules called "bindings"
- Bindings are simply used to bind exchanges to queues for message delivery

Four Exchange Types:

| Types | Default pre-declared names |
|---|---|
| Direct Exchange | (Empty string) and amq.direct |
| Fanout Exchange | amq.fanout |
| Topic Exchange | amq.topic |
| Headers Exchange | amq.match (and amq.headers in RabbitMQ) |

# Queues


- A core element in any MQ protocol especially for RabbitMQ
- Messages are routed to queues from exchanges
- Queues are final destinations in RabbitMQ before being received by subscribers
- Routing algorithms depends on the exchange type and rules called "bindings"
- Bindings are simply used to bind exchanges to queues for message delivery

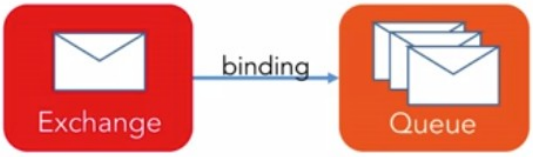
Properties of a Queue:

| | |
|---|---|
| Name | The name of the queue |
| Durable | Either persist the queue to the disk or not |
| Exclusive | Delete the queue if not used anymore |
| Auto-Delete | Delete the queue when consumer unsubscribes |

# Topics


- Topics are simply the "*subject*" part of the messages
- Defined as **routing_key** for message grouping
- Special formatting for better use:
  - "app.logs.error"
- Optional parameters for message exchange
- You can send and receive messages without any topic information
- Topic Exchanges are defined using Topics for message delivery

# Bindings




- Rules that exchanges use to route messages to queues
- To instruct an exchange E to route messages to a queue K, **K has to be bound to E!**
- May have an optional routing key attribute used by some exchange types
- So the routing key acts like a filter
- Binding Analogy:
  - Queue is like your destination in Istanbul city of Turkey
  - Exchange is like Atatürk airport in Istanbul city
  - Bindings are routes from Atatürk to your destination. There can be zero or many ways to reach it
- If message cannot be routed to any queue (there are no bindings for the exchange it was published to) it is either dropped or returned to the publisher, depending on message attributes the publisher has set.

# RabbitMQ vs Others: A Comparison

| | RabbitMQ | ActiveMQ | ZeroMQ | IronMQ | Apache Kafka | Apache Qpid |
|---|---|---|---|---|---|---|
| Brokerless / Decenterilized | No | No | Yes | Distributed & Cloud Based | Distributed | No |
| Clients | C, C++, Java,Others | C, C++, Java, Others | C, C++, Java, Others | C, C++, Java, Others | C, C++, Java, Others | C, C++, Java, Others |
| Transaction | Yes | Yes | No | No | No but can be implemented with plugin | Yes |
| Persistence / Reliability | Yes (configurable) | Yes (built-in) | No persistence; requires higher layers for it | Yes (built-in) | Yes /built-n File System) | Yes (extra plugin required) |
| Routing | Yes (easier to implement) | Yes (easier to implement) | Yes (complex to implement) | No | Can be implemented | No |
| Failover / HA | Yes | Yes | No | Yes | Yes | Yes |