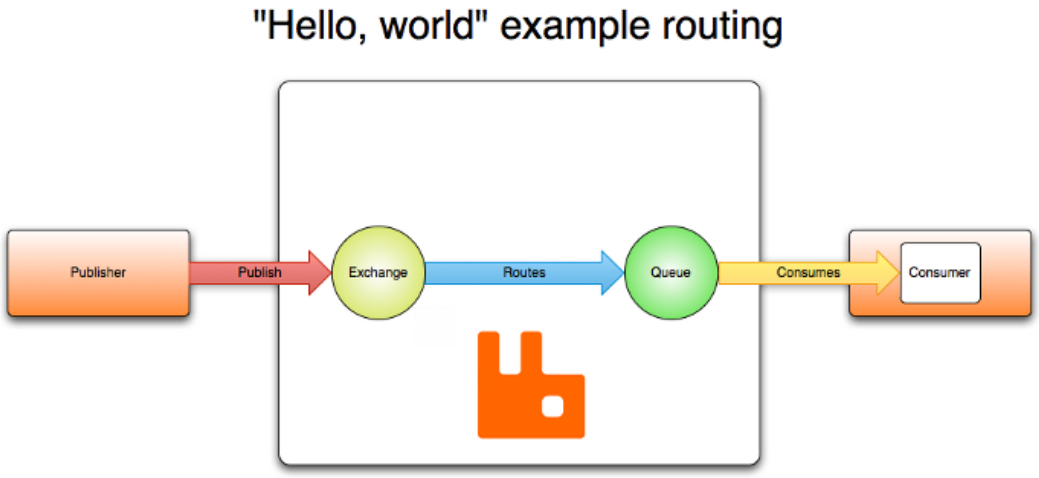
RabbitMQ

Overview

RabbitMQ uses the AMQP 0-9-1 message protocol. The AMQP 0-9-1 Model has the following view of the world: messages are published to *exchanges*, which are often compared to post offices or mailboxes. Exchanges then distribute message copies to *queues* using rules called *bindings*. Then AMQP brokers either deliver messages to consumers subscribed to queues, or consumers fetch/pull messages from queues on demand.



When publishing a message, publishers may specify various *message attributes* (message meta-data). Some of this meta-data may be used by the broker, however, the rest of it is completely opaque to the broker and is only used by applications that receive the message.

Networks are unreliable and applications may fail to process messages therefore the AMQP model has a notion of *message acknowledgements*: when a message is delivered to a consumer the consumer *notifies the broker*, either automatically or as soon as the application developer chooses to do so. When message acknowledgements are in use, a broker will only completely remove a message from a queue when it receives a notification for that message (or group of messages).

In certain situations, for example, when a message cannot be routed, messages may be *returned* to publishers, dropped, or, if the broker implements an extension, placed into a so-called "dead letter queue". Publishers choose how to handle situations like this by publishing messages using certain parameters.

Queues, exchanges and bindings are collectively referred to as *AMQP entities*.

Using RabbitMQ

Docker-Compose

The following docker-compose file will bring up rabbitMQ service running on the default port of 5672.

This particular version also includes a web management user interface accessible at http://localhost:15672/

docker-compose.yml

```
version: '3.0'

services:

  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - '5672:5672'
      - '15672:15672'
    environment:
      RABBITMQ_DEFAULT_USER: rabbit
      RABBITMQ_DEFAULT_PASS: password
    #restart: always
    volumes:
      - rabbit-volume:/var/lib/rabbitmq

volumes:
  rabbit-volume:
    driver: local
    driver_opts:
      type: 'none'
      o: 'bind'
      device: '$PWD/rabbitmq'
```

Java Samples

Producer - Basic Publish

TestProducer.java

```
package com.irdeto.keystone.service.notification;

import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;

public class TestProducer {

    private final static String NOTIFICATION_QUEUE = "keystone_notifications";

    public static void main(String[] argv) throws Exception {
        String message = "{\n" +
                "    \"type\": \"notificationType\",\n" +
                "    \"payload\": {       \n" +
                "        \"name\": \"value\",\n" +
                "        \"name\": \"value\"\n" +
                "    }\n" +
                "}";

        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setUsername("rabbit");
        factory.setPassword("password");

        try (Connection connection = factory.newConnection();

            Channel channel = connection.createChannel()) {
            channel.queueDeclare(NOTIFICATION_QUEUE,true,false, false, null);
            channel.basicPublish("",NOTIFICATION_QUEUE,null, message.getBytes("UTF-8"));

            System.out.println(" [x] Sent '" + message + "'");
        }
    }
}
```

Consumer - Basic Consume

TestConsumer.java

```
package com.irdeto.keystone.service.notification;


import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DeliverCallback;

public class TestConsumer {

    private final static String NOTIFICATION_QUEUE = "keystone_notifications";

    public static void main(String[] argv) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setUsername("rabbit");
        factory.setPassword("password");
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        channel.queueDeclare(NOTIFICATION_QUEUE,true,false, false, null);
        System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

        DeliverCallback deliverCallback = (consumerTag, delivery) -> {
            String message = new String(delivery.getBody(), "UTF-8");
            System.out.println(" [x] Received '" + message + "'");
        };
        channel.basicConsume(NOTIFICATION_QUEUE, true, deliverCallback, consumerTag -> { });
    }
}
```

Management API

You can install an optional management plugin for RabbitMQ. This will allow you to query RabbitMQ from CLI and REST.

CLI:

See `https://www.rabbitmq.com/management-cli.html`

REST:

https://pulse.mozilla.org/api/

Sample Rest queries

```
GET http://localhost:15672/api/vhosts
GET http://localhost:15672/api/queues
GET http://localhost:15672/api/queues/test
```

References

Reference	URL

Tutorials	http://www.rabbitmq.com/getstarted.html
Management CLI	`https://www.rabbitmq.com/management-cli.html`
Rest Reference	https://pulse.mozilla.org/api/
Postman Collection	RabbitMQ.postman_collection.jsonRabbitMQ.postman_collection.json
Enabling TLS with Java Examples	https://www.rabbitmq.com/ssl.html#enabling-tls