RabbitMQ 개요

개요

RabbitMQ는 얼랭으로 만들어진 메시지 전송 브로커 오픈소스이다. (2007년 릴리즈)

시스템 간 비동기 메시징을 가능하게 하여 서비스 간 통신을 안정적이고 효율적으로 처리할 수 있게 해준다

플랫폼 중립적이며, 가볍고 다양한 언어들의 클라이언트 라이브러리를 제공하고, 유연한 성능과 안정성을 제공한다.

클러스터를 설정할 때 큐를 HA(High Available) 로 설정하여 여러 노드에 저장하므로서, 메시지 손실을 방지한다. 또한 쉽게 새로운 기능을 추가할 수 있고, Federation 플러그인을 통해 데이터 동기화와 다중 마스터 복제를 구성할 수 있다.

최근 MSA 환경에서 가장 일반적으로 쓰이는 기술중에 하나이며, 대량의 데이터 전송 시 발생할 수 있는 과부하를 분산시키며, 비동기 처리와 지연이 필요한 작업을 효과적으로 관리 해줄 수 있다.

대표적인 도메인 활용 사례

- EDA(Event-Driven Architecture) : 주문/결제/배송 등의 비즈니스 도메인을 이벤트 기반의 독립적인 도메인 모듈로 구성
- 로그 및 모니터링 : 로그 수집 및 준 실시간 처리 모니터링 시스템 구축
- 채팅 및 알람
- 비동기 데이터 처리 : 대용량의 이미지나 비디오 처리에 효율적인 분산 시스템 구축 가능
- 여러 디바이스의 요청 처리 : IoT와 같은 멀티 디바이스 환경에서의 요청 처리
- 비동기 아키텍처와 EDA에 대한 아마존 CTO 기조 연설 (https://zdnet.co.kr/view/?no=20221202183934)

RabbitMQ 설치하기 (Mac)

```
// brew install
https://brew.sh/
```

```
// 설치 전에 brew 업데이트
brew update

// rabbitmq 설치
brew install rabbitmq
```

도커의 경우

```
# latest RabbitMQ 4.0.x
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672
```

를 통해 도커 이미지를 띄울 수 있다.

만약 다른 운영 체제일 경우

```
// 바이너리나 인스톨러 등을 활용해서 설치
https://www.rabbitmq.com/docs/download
```

윈도우의 경우 환경 변수에서 RabbitMQ의 실행 bin 폴더를 지정해주면 된다.

```
C:\Program Files\RabbitMQ Server\rabbitmq_server-4.0.3\sbin
```

아직까지 구글에서 RabbitMQ설치를 검색해보면 윈도우 기준으로 설치하는 방식의 글들이 많으므로 참고하기 바란다.

기본 명령어

```
// starts a local RabbitMQ node
brew services start rabbitmq
```

```
// stops the locally running RabbitMQ node
brew services stop rabbitmq
```

삭제

```
brew uninstall rabbitmq
```

삭제 이후 관련 디렉토리도 삭제 해주어야 함

```
rm -r /opt/homebrew/etc/rabbitmq
rm -r /opt/homebrew/var/lib/rabbitmq
rm -r /opt/homebrew/var/log/rabbitmq
```

사용자 보안

http://localhost:15672 로 접근하여 로그인 할 수 있다.

최초 계정은 guest/guest 이기 때문에 admin 권한을 갖는 다른 계정을 추가해주고 default user 는 삭제 해주는게 보안상 권고된다.

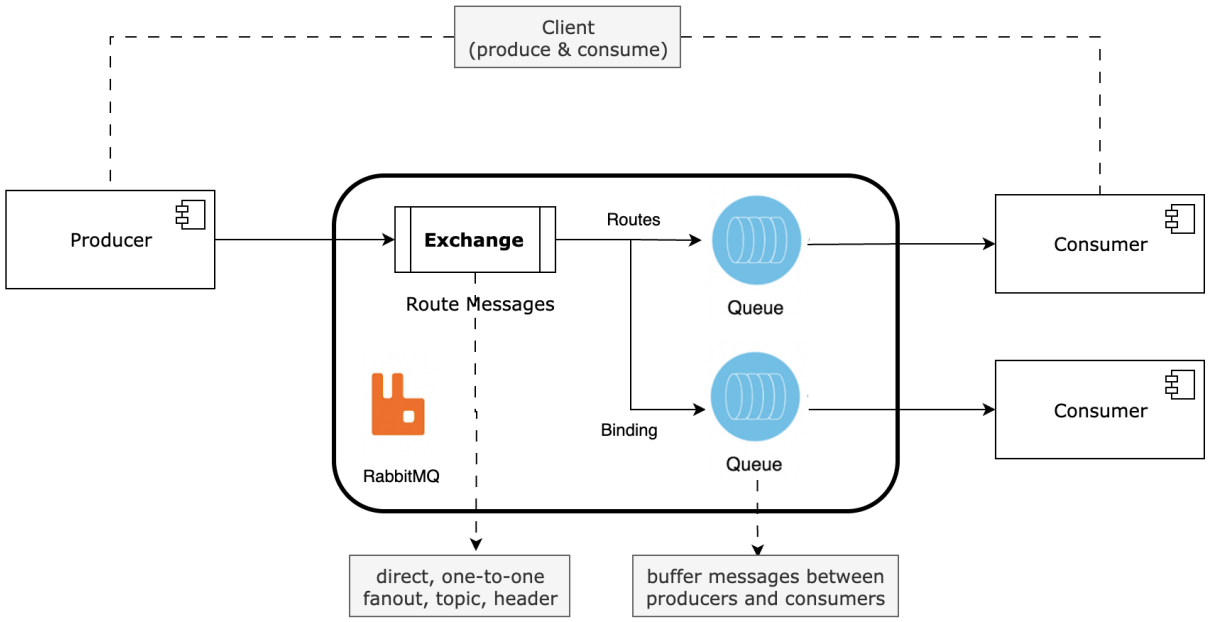
메세지큐를 사용하는 이유

- 비동기 메시지를 사용하여 다른 응용프로그램 사이에 데이터를 송수신
- 클라이언트에 대한 동기 처리는 병목의 요인이므로 비동기로 처리 해도 될 영역에 대해서는 큐를 통해 분리해서 처리한다.
- 결국 분산환경에서 응용프로그램들을 분리하고 독립적으로 확장하기 위해서 사용, 기능 별로 모듈 구성이 용이 (Scalability)
- 요청에 대한 응답을 기다릴 필요가 없기 때문에 각 영역의 역할만 신경쓰면 된다. 어플리케이션 레벨에서 분리 할 수 있다. (Abstraction of concerned)

- 데이터를 메모리 대신에 디스크에 저장하여 데이터 유실을 방지 (Reliability)
  - 즉시 처리하지 않아도 나중에 다시 처리가 가능하다.
  - 메시지 영구 저장, 메시지 확인, 장애 복구 메커니즘을 통해 메시지의 신뢰성을 보장합니다.
- 확장성: 여러 노드에 걸쳐 쉽게 확장할 수 있어 높은 가용성을 제공하며, 필요에 따라 메시지를 클러스터링하거나 페더레이션(federation) 방식으로 확장할 수 있다.
- 유연성: 다양한 exchange 유형과 라우팅 규칙을 지원하여 메시지를 효과적으로 라우팅하고 관리할 수 있다.
  - 광범위한 프로토콜 지원: 기본적으로 AMQP를 사용하지만, STOMP, MQTT, HTTP 등을 포함한 다양한 프로토콜을 지원
  - 풍부한 클라이언트 라이브러리: Java, Python, Ruby, .NET 등 다양한 언어로 클라이언트 라이브러리를 제공하여 다양한 애플리케이션에서 사용할 수 있다

AMQP

Advanced Message Queing Protocol의 약자로, 흔히 알고 있는 MQ의 오픈소스에 기반한 표준 프로토콜을 의미한다.



여기서 말하는 one to one fanout의 경우 Fanout 익스체인지는 여러 큐에 메시지를 브로드캐스트하지만, 각 큐에 연결된 소비자는 한 명이므로 **메시지는 각각의 큐에서 하나의 소비자에게만 전달되므로** 1:1 방식의 메시지 소비를 뜻한다.

여러 소비자에게 동시에 메시지를 전달하되 각 메시지가 단일 소비자에게만 전달되도록 보장하고 싶은 경우를 의미한다고 보면 된다.

AMQP 특징

이전에도 상용화된 MQ 제품들은 많았지만, 한가지 문제가 있다면 대부분 플랫폼 종속적인 제품들이었기 때문에 서로 다른 이기종간에 메시지를 교환하기 위해서는 메시지 포멧 컨버전을 위한 메시지 브릿지를 이용하거나 (속도 저하 발생) 시스템 자체를 통일시켜야 하는 불편함과 비효율성이 있었다. 즉, AMQP의 목적은 서로 다른 시스템간에 (비용/기술/시간적인 측면에서) 최대한 효율적인 방법으로 메시지를 교환하기 위한 MQ 프로토콜로 아래와 같은 특징이 있다.

- 모든 broker들은 똑같은 방식으로 동작할 것
- 모든 client들은 똑같은 방식으로 동작할 것
- 네트웍상으로 전송되는 명령어들의 표준화
- 프로그래밍 언어 중립적

Routing Model Components

AMQP의 라우팅 모델은 아래와 같은 3개의 중요한 component 들로 구성된다.

- Exchange
- Queue
- Binding

각 컴포넌트들을 아래와 같은 상세 기능들을 수행한다.

- Exchange : Publisher로부터 수신한 메시지를 적절한 큐 또는 다른 exchange로 분배하는 라우터의 기능을 한다.
- Queue : 일반적으로 알고 있는 큐이다. 메모리나 디스크에 메시지를 저장하고, 그것을 consumer에게 전달하는 역할을 한다.
- Binding : exchange와 큐와의 관계를 정의한 일종의 라우팅 테이블이다. 같은 큐가 여러 개의 exchange에 bind 될 수도 있고, 하나의 exchange에 여러 개의 큐가 binding 될 수도 있다.

Routing Key : 라우팅 키는 발행된 메시지와 큐가 라우팅 테이블을 통해 매칭되는 키를 뜻합니다. Publisher, 혹은 producer로 칭하는 송신부에서 송신한 메시지 헤더에 포함되는 것으로 일종의 가상 주소라고 보면 된다.