개발 노트

RabbitMQ로 메시지 손실 최소화 하는 법(1)

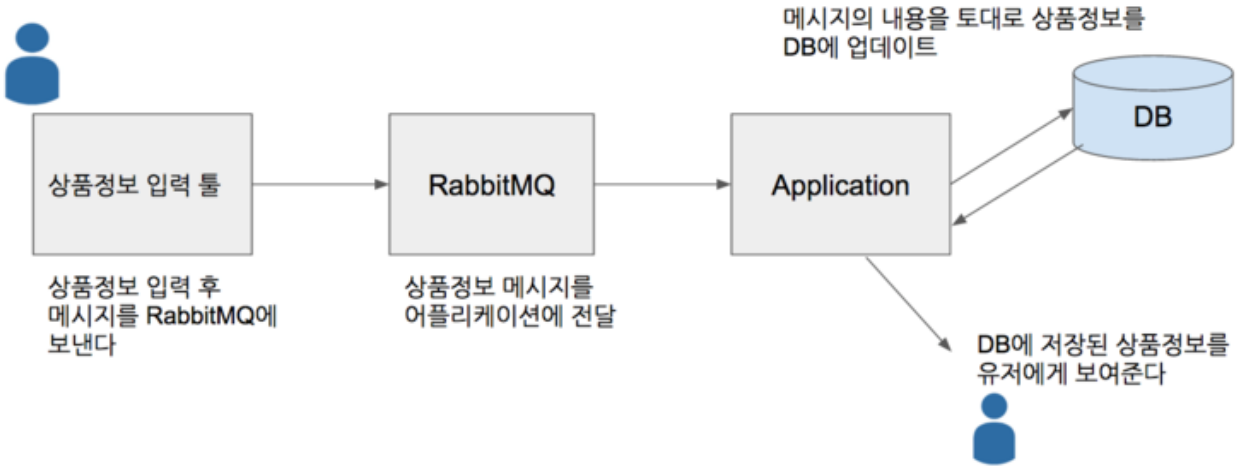
개발몬스터 · 2017. 7. 14. 17:40

우리는 메시지를 발생한 후 받아서 처리하는 메시징 시스템의 도입이 필요했다. 물론 저장소를 이용하여 직접 구현하여도 되지만 이미 검증된 다양한 솔루션이 나와있기 때문에 적절히 선택을 하면 되었다. 솔루션을 도입하기 위해 몇가지 고려해야 할 사항은 아래와 같았다.

- 다양한 플랫폼과 클라이언트 언어를 지원해야 한다.
- 메시지 신뢰성을 높여주는 기능을 지원해야 한다.
- UI관리 툴을 제공하여 쉽게 관리가 가능하도록 한다.

RabbitMQ는 위의 요구사항을 충족 할 뿐만 아니라 유연한 라우팅, 클러스터링, 오픈소스이며 상업적 지원 또한 가능하다는 여러 장점이 있다.

아래는 우리 시스템에서 어떻게 사용하는지에 대해서 보여준다.



위의 그림은 상품관리자가 상품정보를 입력하여 실제 DB에 적재한 후 사용자에게 보여주는 데이터 흐름을 나타낸다.

위에서 보듯이 상품정보를 업데이트 하기 위해서는 반드시 MQ를 통해 메시지를 받아야 한다. 가격이 변경되었음에도 불구하고 이전의 가격으로 사용자들에게 보여준다면 엄청난 재앙일 것이다. 따라서 우리는 MQ를 사용할 때 메시지의 신뢰성에 중점을 두고 운영을 해야한다.

RabbitMQ를 이용하여 메시지 손실을 최소화 하는 법을 살펴보자.

< RabbitMQ시스템 구성도 >

RabbitMQ를 이용하는 주체는 크게 3가지로 나뉜다.

- 프로듀서 (메시지를 보내는 주체)
- 브로커 (메시지를 컨슈머에게 전달하는 주체)
- 컨슈머 (메시지를 받아 소비하는 주체)

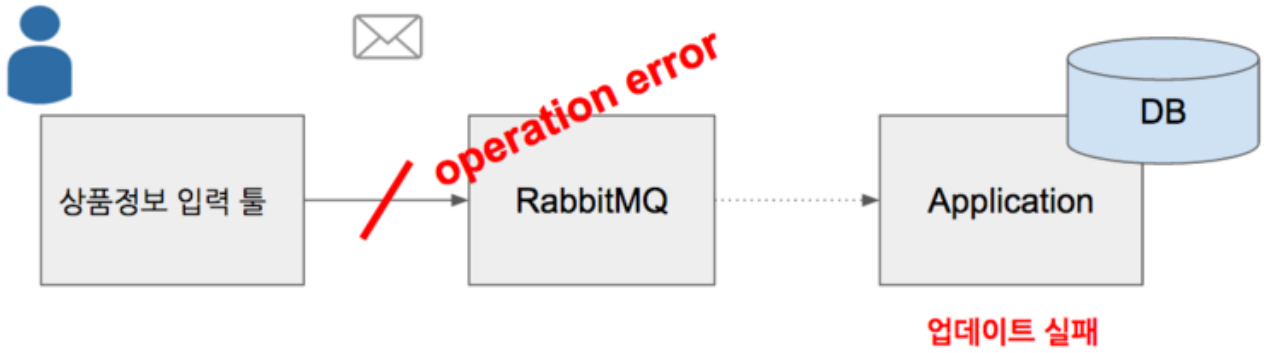
1부에서는 브로커의 설정에 대해서 알아본 후 2부에서는 컨슈머에 대해서 알아본다.

**RabbitMQ에서 나오는 용어나 기본적인 동작은 어느정도 알고있다는 가정하에 설명할 것이다. 예제들은 모두 Java Client Library를 이용하여 작성한 것이다.*

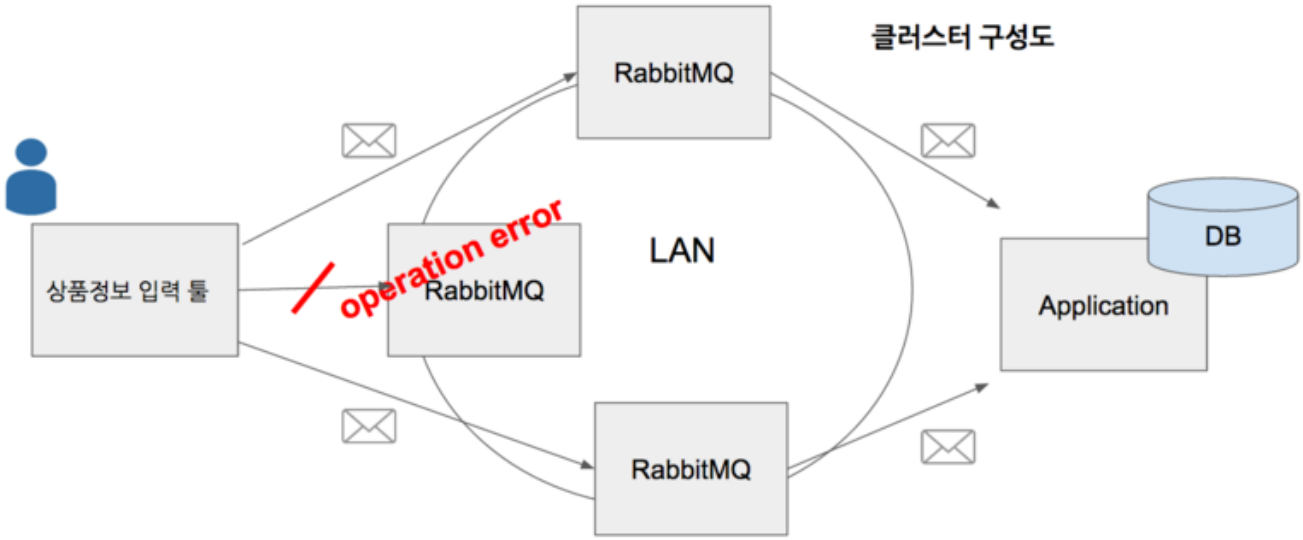
브로커

1. Clustering

한개의 노드를 운영 시 예기치 못한 에러로 인하여 노드가 작동불능 상태가 된다면 모든 상품정보가 업데이트 되지 못할 것이다. 이것은 시스템 전면 장애로 이어진다. 이러한 사태를 막기 위해서 여러 개의 노드를 하나의 클러스터로 운영을 하면 장애를 방지할 수 있다.



한개의 노드 운영 시 노드가 동작을 하지 않는다면 상품정보를 업데이트 하지 못하게 된다.



하나의 노드가 작동하지 않더라도 다른 노드에 의해서 메시지를 전달 할 수 있다.

rabbitmqctl의 join_cluster 명령을 통해서 클러스터를 구성할 수 있다. 자세한 사용법은 여기를 참고하자.

클러스터링 시 주의할점

- 모든 노드는 같은 Erlang 쿠키 정보를 가지고 있어야 한다.
- 네트워크는 LAN을 통해서 구성돼야 한다.
- 모든 노드는 같은 버전의 RabbitMQ와 Erlang으로 작동 되어야 한다.
- 클러스터에 참여하기 전 노드는 DISK or RAM 모드를 설정할 수가 있다.

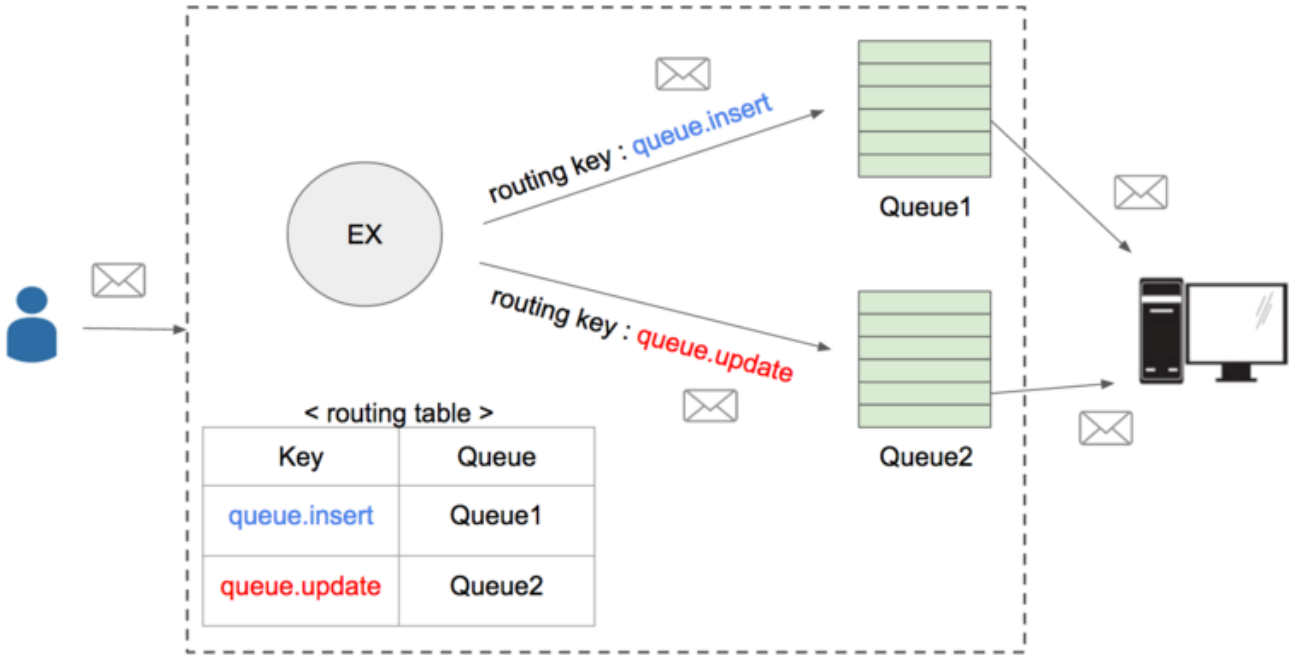
(RAM 모드 설정 시 메시지 손실을 방지할 수 없기 때문에 DISK 모드를 권장한다)

DISK or RAM 모드

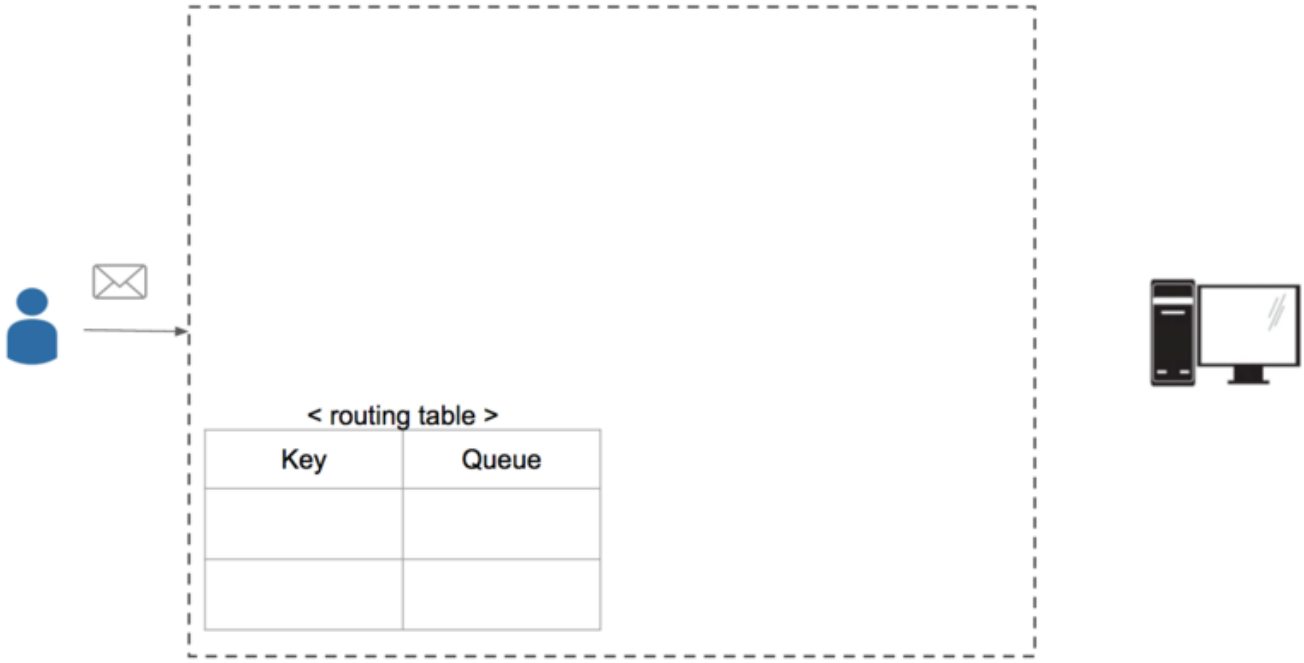
클러스터 구성 시 각 노드는 램 or 디스크 모드를 선택 할 수 있다. 램 모드를 선택하게 되면 메시지 처리 시간이 줄어들어 퍼포먼스를 높일 수 있지만 메시지의 영속성을 보장하지 못하게 되는 단점이 있다. 이와 반대로 디스크 모드 선택 시 메시지 처리 시간은 늘어나지만 메시지의 손실을 막을 수 있게 된다는 장점이 있다. 우리의 시스템인 경우 평균 1초에 1~2개 정도만 처리하고 큐에 쌓아두지 않고 즉시 처리가 가능하기 때문에 굳이 퍼포먼스를 생각하여 램 모드를 선택하지 않아도 된다.

2. Exchange and Queue Durability

브로커가 재시작이 되면 Exchanger 및 Queue는 이전의 정보를 가지고 동일하게 재구성이 되어야 한다. 그렇지 않다면 메시지를 보내지 못할 뿐만 아니라 메시지를 받을 수도 없는 상황이 발생하게 된다.



위의 그림에서는 라우팅 테이블을 참조하여 각 라우팅 키에 해당하는 큐에 메시지를 전달하고 있다. 아래의 그림은 노드가 재가동했을 시 Exchanger와 Queue가 삭제되어 있는 경우를 보여준다.



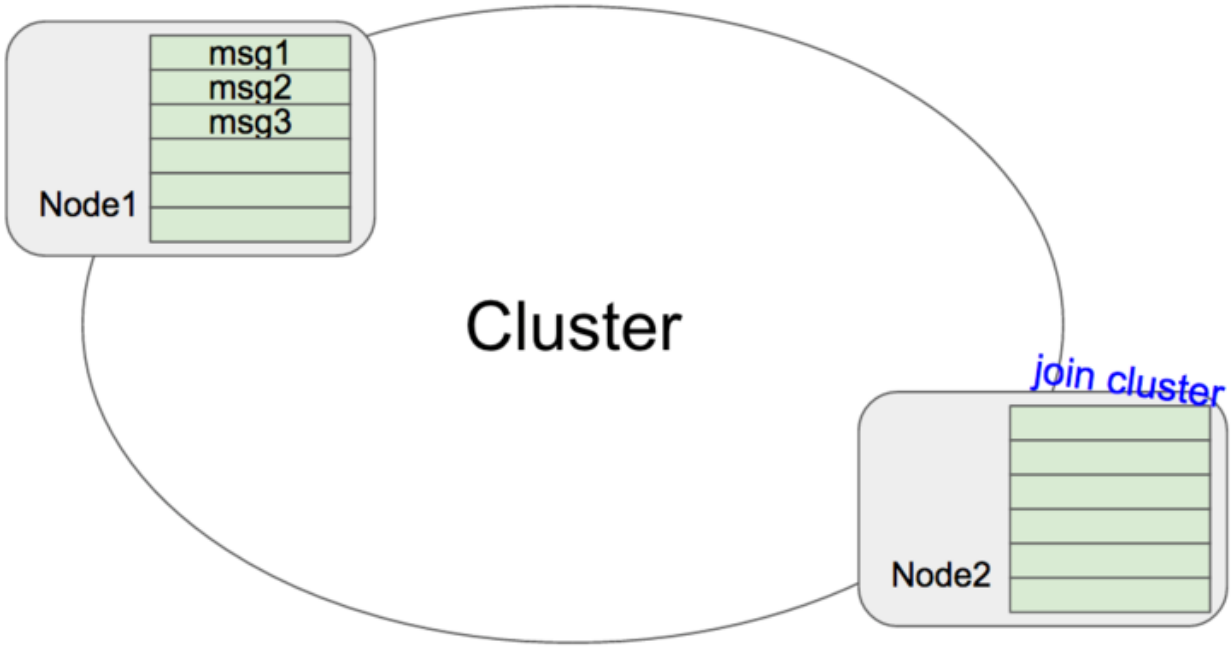
위의 그림에서 알 수 있듯이 메시지를 보내게 되면 해당 메시지를 처리하는 Exchanger가 존재하지 않아 메시지 손실이 발생한다. 이러한 경우 Producer가 Confirm Mode일 경우 라우팅 되지 못하였다는 메시지를 받을 수 있다.

아래의 함수를 통해 Exchanger, Queue Durability를 설정하여 브로커가 재가동 시에도 이전의 정보를 가지고 복원되게 할 수 있다. durable 값을 true로 넘겨준다.

```
Channel.exchangeDeclare(exchange_name, exchange_type, durable, auto_delete, args);
Channel.queueDeclare(queue_name, durable, exclusive, autoDelete, args);
```

3. Queue Mirroring

같은 클러스터에 있는 모든 노드의 큐는 동일한 메시지를 가지지 않는다. 이러한 경우 하나의 노드가 동작하지 않는다면 큐에 있는 메시지는 노드가 다시 동작하지 않는 한 복구되지 않는다. 큐 미러링을 통해서 이러한 점을 보완할 수 있다.



Node2가 클러스터에 참여했지만 메시지가 동기화 되지않아 큐가 비어있다.

위의 그림에서 Node2가 클러스터에 참여하게 되면 큐의 메시지를 제외한 모든 요소들은 동일하게 구성된다. Node1이 동작하지 않더라도 Node2를 통해서 메시지를 보내기 원한다면 아래의 방법으로 큐 미러링 설정이 가능하다.

큐 미러링 설정

rabbitmqctl set_policy 명령을 통해서 미러링 정책을 정할 수 있다.

ex) rabbitmqctl set_policy -p {호스트명} {정책명} {큐이름(정규식표현가능)} {정책파라미터}

1. host1에 ha-all라는 정책 이름을 가지고 'ha.' 이름으로 시작하는 큐를 모든 노드에 미러링

```
rabbitmqctl set_policy -p host1 ha-all "^ha\." '{"ha-mode":"all"}'
```

2. host1에 ha-two라는 정책 이름을 가지고 'two.' 이름으로 시작하는 큐를 2개의 노드에만 미러링

```
rabbitmqctl set_policy -p host1 ha-two "^two\." '{"ha-mode":"exactly","ha-params":2}'
```

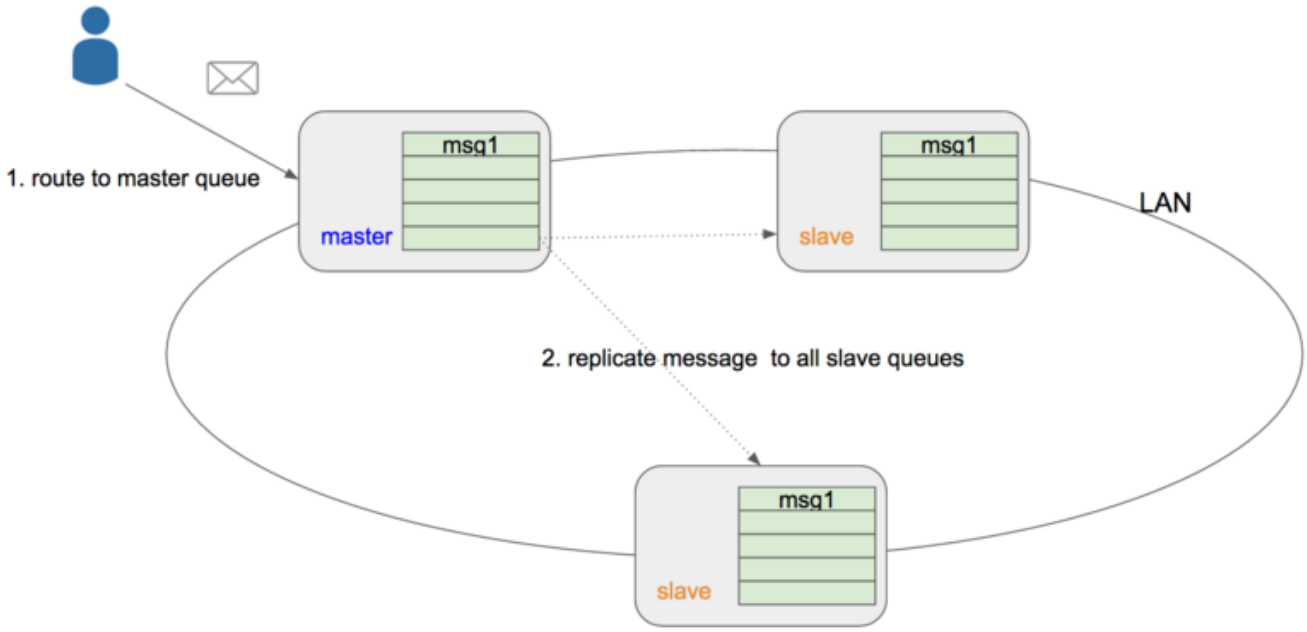
3. host1에 ha-nodes라는 정책 이름을 가지고 'nodes.' 이름으로 시작하는 큐를 특정 노드에 미러링

```
rabbitmqctl set_policy -p host1 ha-nodes "^nodes\." '{"ha-mode":"nodes", "ha-params":["r
```

모든 노드의 큐에 미러링을 설정하게 된다면 메시지 신뢰성은 높아지겠지만 성능은 저하될 수 있다. 따라서 미러링 큐의 갯수는 신뢰성, 속도 두 요소의 중요도에 따라서 적절히 조정해야한다.

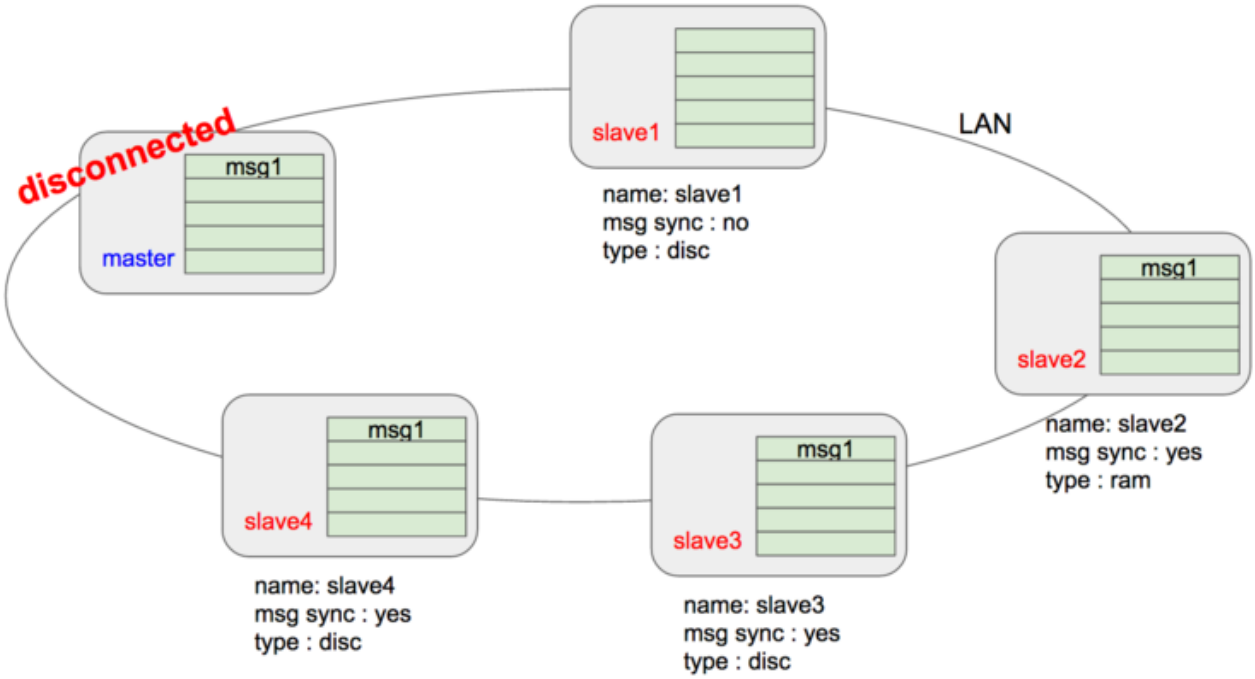
4. Master and Slave

큐에는 마스터와 슬레이브 두가지 타입이 있다. 모든 노드에 존재하는 각 큐에는 반드시 하나의 마스터 큐가 있어야 한다. 큐에 메시지가 라우팅 되면 가장 먼저 마스터 큐에 메시지가 들어오게 되고 이 후 슬레이브 큐에 전달된다. 이것은 FIFO방식의 메시지 전달 방식에서 필수적인 요소이다.



마스터 큐 노드에 메시지가 들어오게 되고 이 후 슬레이브 큐에 복제가 된다.

큐에는 반드시 마스터가 존재해야하기 때문에 마스터 큐를 가지고 있는 노드가 동작이 멈추게 된다면 후보자들 중에서 마스터를 선택해야 한다. 마스터를 선정하는데 있어서 몇가지 규칙이 있다. 아래의 그림에서 보듯이 마스터의 동작이 멈추게 되었을 시 어떠한 노드가 마스터가 선정되는지 알아보자.



클러스터링에 조인한 순서는 slave1 -> slave2 -> slave3 -> slave4이다.

결론부터 말하면 slave3이 마스터 노드가 된다.

각 노드가 마스터가 되지 못한 이유는 아래와 같다.

- slave1 : 메시지 동기화가 되지 않아서
- slave2 : 디스크 타입으로 동작하지 않기 때문에
- slave4: slave3보다 클러스터에 늦게 조인하여서

위의 이유를 종합해 보면 마스터 선정 기본 정책은 가장 오래 살아남고 디스크모드로 동작하며 메시지 동기화가 되어있는 노드가 선정된다. 그러나 클러스터 조인 순서 및 메시지 동기화에 따른 선정기준은 정책에 따라서 변할 수 있다. (동기화되지 않은 노드도 마스터가 될 수는 있지만 메시지 손실이 발생한다는 것을 알아야 한다)

요약하며...

메시지 시스템을 운영하면서 우리가 중점적으로 다룬 내용은 메시지의 손실을 최소화 하는 방법이다. 이를 위해 RabbitMQ에서는 많은 기능들을 제공하고 있고 이를 사용하는 방법을 알아보았다. 아래와 같다.

첫번째, 클러스터를 구성하여 가용할 수 있는 노드의 수를 늘리자.
두번째, 서버의 오류로 인해서 재가동 시 이전의 설정으로 자동 복원이 되도록 한다.
세번째, 큐 미러링을 통해서 모든 노드에 메시지를 동기화 시킨다.
네번째, 큐의 마스터와 슬레이브에 대해서 고려하여 서버를 구성한다.

다음 2부에서는 메시지를 소비하는 컨슈머에서 메시지 손실을 방지하기 위한 방법에 대해서 알아보도록 하겠다.

참고자료

- https://www.cloudamqp.com/docs/index.html
- https://www.rabbitmq.com/documentation.html