

Системы хранения и передачи данных: Очереди RabbitMQ



Роман
Гордиенко



Роман Гордиенко

Backend Developer, Factory5



План занятия

1. [Введение](#)
2. [Что такое RabbitMQ?](#)
3. [Архитектура очередей](#)
4. [Как работать с очередями сообщений?](#)
5. [Отказоустойчивость очередей](#)
6. [Итоги](#)
7. [Домашнее задание](#)



Введение



Что такое очереди сообщений?

Очереди сообщений предоставляют асинхронный протокол передачи данных. Это означает, что отправитель и получатель сообщения не обязаны взаимодействовать с очередью сообщений одновременно. Размещённые в очереди сообщения хранятся до тех пор, пока получатель не получит их.

Особенности очередей сообщений

- Очереди сообщений имеют неявные или явные ограничения на размер данных, которые могут передаваться в одном сообщении, и количество сообщений, которые могут оставаться в очереди.
- Многие реализации очередей сообщений функционируют внутренне: внутри операционной системы или внутри приложения. Такие очереди существуют только для целей этой системы.
- ...



Особенности очередей сообщений

- Другие реализации позволяют передавать сообщения между различными компьютерными системами, потенциально подключая несколько приложений и несколько операционных систем. Эти системы очередей сообщений обычно обеспечивают расширенную функциональность для обеспечения устойчивости, чтобы гарантировать, что сообщения не будут «потеряны» в случае сбоя системы.

FIFO и LIFO (ФИФО и ЛИФО) — что это такое?

Сервисы обмена сообщениями между серверами делятся на два типа:

- **Очереди (ФИФО-метод)** — «первый пришел и первый ушел» (First in First out). Принцип FIFO означает: сообщение, которое первым попало в очередь, первым же отправляется на обработку.
- **Стеки (LIFO)** — «пришел последним, а ушел первым» (Last in First out). В отличие от системы FIFO, стек можно представить в виде стопки книг: вы кладете книги друг на друга и сначала берете верхние книги.

Для серверных приложений наиболее востребованы очереди (метод FIFO), но большинство сервисов для организации очередей могут работать и в режиме стека (LIFO).

Брокеры сообщений

Для реализации очередей сообщений используются брокеры сообщений:

- **RabbitMQ** — самый популярный брокер, держит большую нагрузку, умеет кластеризоваться;
- **Kafka** — больше напоминает распределенный журнал, умеет разбивать топики на шарды, используется в хайлоаде, так же умеет кластеризоваться, для работы нужен Zookeeper.
- **ActiveMQ** — брокер сообщений, написанный на Java. Компонент, реализующий очереди называется Artemis, реализует спецификацию JMS 1.1.
- etc....

Хорошая книга по Kafka “Арасче Kafka. Потокковая обработка и анализ данных”.
Ния Нархид, Гвен Шапира, Тодд Палино.



Что ты такое, RabbitMQ?

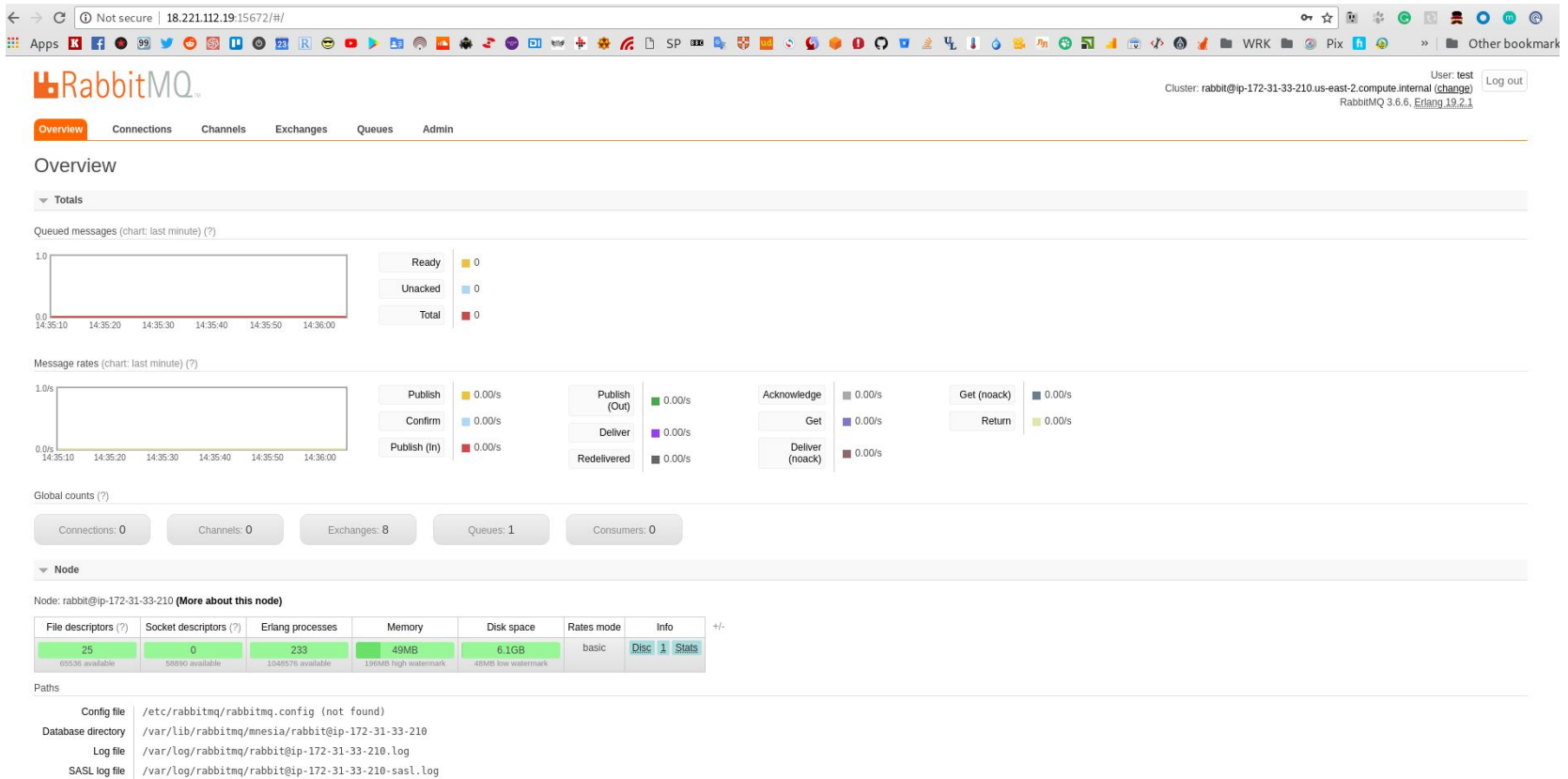
Что за кролик этот RabbitMQ?

RabbitMQ — программный брокер сообщений на основе стандарта.

AMQP — тиражируемое связующее программное обеспечение, ориентированное на обработку сообщений. Создан на основе системы Open Telecom Platform, написан на языке Erlang, в качестве движка базы данных для хранения сообщений использует Mnesia.

- Умеет масштабироваться горизонтально (ноды можно объединять в кластер) и вертикально, имеет большой набор плагинов (например management, shovel, etc...);
- Имеет огромное количество поддерживаемых клиентов: Ruby, Python, Go, Java ...

WEB интерфейс RabbitMQ



Источник



Архитектура очередей

Протокол AMQP

1. Сообщение (message) — единица передаваемых данных, основная его часть (содержание) никак не интерпретируется сервером, к сообщению могут быть присоединены структурированные заголовки.

Протокол AMQP

2. Точка обмена (exchange) — в неё отправляются сообщения. Точка обмена распределяет сообщения в одну или несколько очередей. При этом в точке обмена сообщения не хранятся. Точки обмена бывают трёх типов:

- fanout — сообщение передаётся во все прицепленные к ней очереди;
- direct — сообщение передаётся в очередь с именем, совпадающим с ключом маршрутизации (routing key) (ключ маршрутизации указывается при отправке сообщения);
- topic — нечто среднее между fanout и direct, сообщение передаётся в очереди, для которых совпадает маска на ключ маршрутизации, например, `app.notification.sms.#` — в очередь будут доставлены все сообщения, отправленные с ключами, начинающимися с `app.notification.sms`.

Протокол AMQP

3. Очередь (queue) — здесь хранятся сообщения до тех пор, пока не будут забраны клиентом. Клиент всегда забирает сообщения из одной или нескольких очередей.

Терминология

- **Exchange** — сущность которая получает сообщения от приложений и при необходимости перенаправляет их в очереди сообщений.
- **Binding** — отношение между очередью сообщений и точками обмена.
- **Routing key** — виртуальный адрес, который точка обмена использует для принятия решения о дальнейшей маршрутизации.

Exchange

Принимает сообщения от поставщика и направляет их в **message queue** в соответствии с predetermined критериями. Такие критерии называют **bindings**.

Exchange — механизм согласования и маршрутизации сообщений. На основе сообщений и их параметров (bindings) принимают решение о перенаправлении в очередь или другой exchange. **Не хранят сообщения.**

Термин exchange означает алгоритм и экземпляр алгоритма. Также говорят exchange type и exchange instance.

AMQP определяет набор стандартных типов exchange. Приложения могут создавать свои exchange instance.

Алгоритмы маршрутизации

Существует несколько стандартных типов exchange, описанных в стандарте. Из них два являются важными:

- **Direct exchange** — маршрутизация на основе routing key.
Базовый exchange — это direct exchange
- **Topic exchange** — маршрутизация на основе шаблона маршрутизации.

У RabbitMQ существует еще **Fanout** отправляет сообщения во все связанные очереди без проверки ключа маршрутизации или заголовка сообщения. И **Headers** работает по заголовкам, которые удобней и легче ключей маршрутизации.

Routing Key

В общем случае exchange:

- проверяет свойства сообщения, поля заголовка и содержимое его тела;
- используя эти и, возможно, данные из других источников, решает, как направить сообщение.

В большинстве простых случаев exchange рассматривает одно ключевое поле, которое мы называем **Routing Key**.

Routing Key — это виртуальный адрес, который сервер exchange может использовать для принятия решения о направлении сообщения.

Routing Key

- Для маршрутизации типа **point-to-point** ключом маршрутизации обычно является имя очереди сообщений.
- Для маршрутизации **pub-sub** ключ маршрутизации обычно является значением иерархии топика.
- В более сложных случаях ключ маршрутизации может быть объединен с маршрутизацией по полям заголовка сообщения и/или его содержанием.

Message Queue

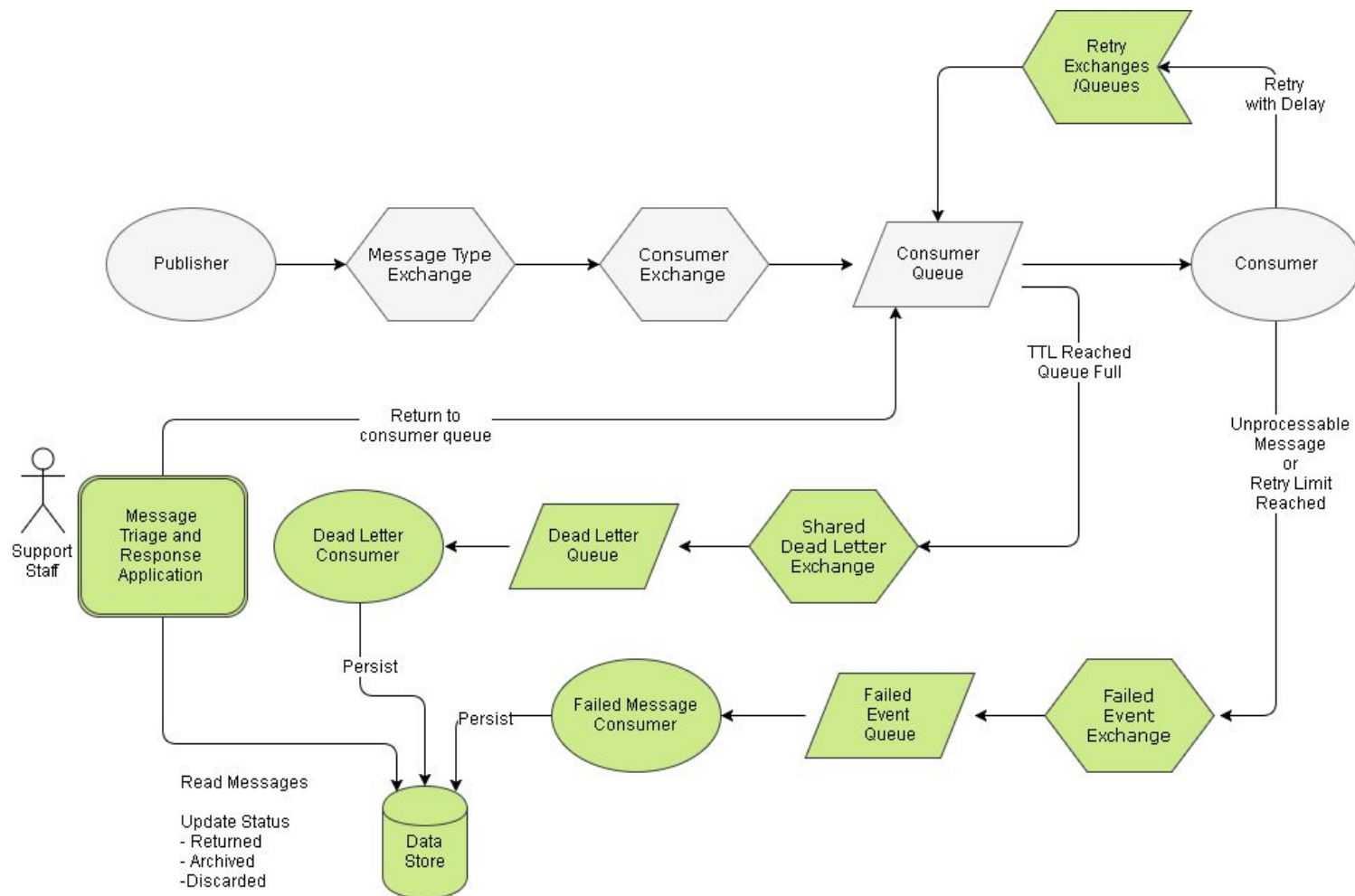
Когда клиентское приложение создает очередь сообщений, оно может указать следующие свойства:

- `name` — если не указано, сервер сам выбирает имя и отправляет его клиенту. Как правило, когда приложения совместно используют очередь сообщений, они заранее договариваются об имени очереди сообщений, и когда приложение нуждается в очереди сообщений для своих собственных целей, оно позволяет серверу предоставлять имя.
- ...

Message Queue

- `exclusive` — если этот параметр установлен, то очередь существует, пока существует текущее соединение. Очередь удаляется при разрыве подключения.
- `durable` — если установлен, очередь существует и активна при перезагрузке сервера. Очередь может потерять сообщения посланные во время перезагрузки сервера.

Жизненный цикл сообщения



Источник

Жизненный цикл сообщения

- Сообщение создается **producer**.
- **Producer** маркирует сообщение с помощью маршрутной информации.
- Затем **producer** отправляет сообщение в **exchange**.
- **Exchange** (обычно) направляет его в набор очередей.
- Когда сообщение поступает в очередь сообщений, она немедленно пытается передать его потребителю через AMQP.
- Если отправить не удастся возвращаем **Producer** или **dead-letter**.
- После получения ставим **ACK** и сообщение из хранилища удаляется.

Producer

Producer — клиентское приложение, которое публикует сообщения в **exchange**.

По аналогии с устройством электронной почты, можно заметить, что **producer** не отправляет сообщения непосредственно в очередь (message queue). Иное поведение нарушило бы модель AMQ. Это было бы похоже на жизненный цикл сообщения электронной почты: разрешение электронной почты, обход таблиц маршрутизации MTA и попадание непосредственно в почтовый ящик. Это сделало бы невозможной вставку промежуточной фильтрации и обработки, например, обнаружение спама.



Producer

Модель AMQ использует тот же принцип, что и система электронной почты: все сообщения отправляются в одну точку **exchange** или **MTA**, который проверяет сообщения на основе правил и информации, которая скрыта от отправителя, и направляет их к точкам распространения, которые также скрыты от отправителя.



Consumer

Consumer — клиентское приложение, которое получает сообщения из очереди сообщений.

Наша аналогия с электронной почтой начинает разрушаться, когда мы смотрим на **consumer** (получателей). Почтовые клиенты пассивны — они могут читать почтовые ящики, но они не оказывают никакого влияния на то, как эти почтовые ящики заполняются.



Consumer

С помощью AMQP **consumer** также может быть пассивным, как и почтовые клиенты. То есть мы можем написать приложение, которое прослушивает определённую очередь сообщений и просто обрабатывает поступающую информацию. При этом очередь сообщений должна быть готова до старта приложения и должна быть «привязана» к нему.

Возможности Consumer

- создавать/удалять очереди сообщений;
- определять способ заполнения очереди используя **bindings**;
- выбирать разные **exchanges**, что может полностью изменить семантику маршрутизации.



Как работать с очередями сообщений?

Работа с брокером из командной строки

Для просмотра доступных очередей:

```
rabbitmqctl list_queues
```

Для просмотра содержимого очереди:

```
rabbitmqadmin get queue='hello'
```


Простейший Producer

Код написан на языке Python, для работы нужно установить библиотеку pika командой **pip install pika**:

```
#!/usr/bin/env python
# coding=utf-8
import pika

connection =
pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
channel.basic_publish(exchange='', routing_key='hello', body='Hello
Netology!')
connection.close()
```

Простейший Consumer


```
#!/usr/bin/env python
# coding=utf-8
import pika

connection =
pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')

def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)

channel.basic_consume(queue='hello', on_message_callback=callback,
auto_ack=False)

channel.start_consuming()
```



Отказоустойчивость очередей

Создаем кластер из двух нод

Rabbitmq-ноды и CLI-инструменты (например, rabbitmqctl) используют Erlang cookie-файл для аутентификации между собой.

Каждый нода кластера должна иметь такой файл с одинаковым содержимым на всех нодах кластера.

Содержимое файла `/var/lib/rabbitmq/.erlang.cookie` на `app02` должно совпадать с содержимым файла на `app01`.

Права на файл 400, владелец/группа – `rabbitmq:rabbitmq`.

Создание политики репликации

Для того чтобы реплицировать очереди, необходимо создать политику, которая будет описывать режим и тип репликации. Политики могут создаваться в любое время. Политики могут применяться ко всем очередям или к выборочным очередям (с фильтрацией имени очереди по шаблону регулярного выражения)

Можно создавать не реплицируемые очереди, а потом позже делать их реплицируемыми через создание политики.

```
rabbitmqctl set_policy ha-all ""  
'{"ha-mode":"all","ha-sync-mode":"automatic"}'
```

Объединение нод в кластер

Для того, чтобы создать кластер из двух нод мы присоединим, например, вторую ноду к первой ноде(app01).

Для этого на второй ноде:

```
rabbitmqctl stop_app  
rabbitmqctl join_cluster {ip_addr or dns name}  
rabbitmqctl start_app  
rabbitmqctl cluster_status
```

Полезные CLI команды

Просмотр списка очередей:

```
# rabbitmqctl list_queues
```

Просмотр списка очередей с выводом имен политик, которые применены к этим очередям:

```
# rabbitmqctl list_queues name policy pid slave_pid
```

Ручная синхронизация очереди:

```
# rabbitmqctl sync_queue <имя очереди>
```

Отмена синхронизации очереди:

```
# rabbitmqctl cancel_sync_queue <имя очереди>
```

Полезные CLI команды

Проверка состояния RabbitMQ ноды:

rabbitmqctl node_health_check

Просмотр статуса RabbitMQ-ноды:

rabbitmqctl status

Полный отчет (включая состояние кластера, нод, политик, параметров, пользователей, вирт.хостов и т.д.):

rabbitmqctl report | less

Больше команд доступно по:

rabbitmqctl --help



Итоги

Итоги

Сегодня мы:

- узнали, что такое очереди сообщений;
- поняли, как работает очередь сообщений;
- разобрали жизненный цикл сообщения;
- познакомились с RabbitMQ;
- научились делать отказоустойчивые очереди;
- поработали с очередью сообщений.





Домашнее задание



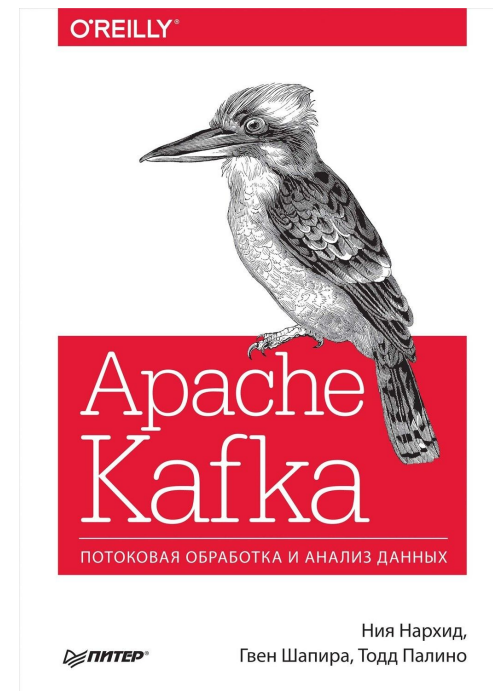
Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

Дополнительные материалы

«Apache Kafka. Потокковая обработка и анализ данных» Ния Нархид, Гвен Шапира, Тодд Палино



**Задавайте вопросы и
пишите отзыв о лекции!**

Роман Гордиенко