Building a simple distributed system with RabbitMQ and Python

Alexander Abushkevich

Why RabbitMQ and Python? Application requirements

- Bandwidth, CPU, memory consuming application.
- Two types of computers available:
  - powerful computers with expensive network connection
  - virtually unlimited bandwidth but with strict CPU time limits
- How to use resources effectively?
- How to connect computers (no dedicated IP in some cases, etc.)?

Why RabbitMQ and Python? Alternative messaging solutions

- ZeroMQ-based application
- Apache QPID
- RabbitMQ
- … (any other solution)

First iteration

Data exchange implemented with bash scripts (ssh, cron)

Pros:

- Easy to setup from scratch
- Requires no additional tools and packages

Cons:

- Needs configuration updates often
- Load balancing is particularly difficult
- Error handling is not so easy - many assumptions, many possibilities of failure

Second iteration

- Crawlers and processors using RabbitMQ for data exchange

Pros:

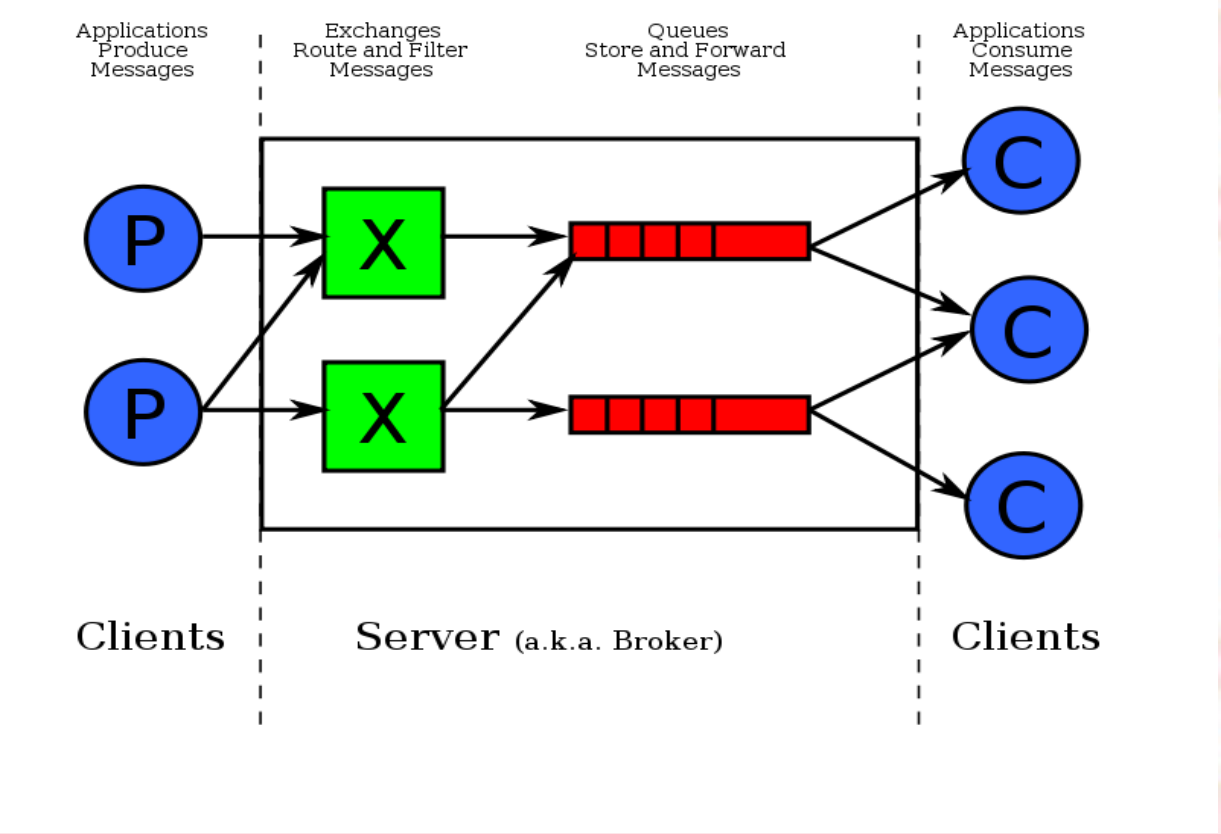
- Almost everything related to message transmission is handled by RabbitMQ
- Unified mechanism for data exchange
- No need to make configuration changes when adding or removing crawlers/processors

Cons:

- ?

RabbitMQ basics

- Exchanges
- Queues
- Routing keys



* http://en.wikipedia.org/wiki/File:The-amqp-model-for-wikipedia.svg

RabbitMQ basics

Exchange types:

- Fanout
  - No need for routing key. All queues, bound to fanout exchange will receive message.
- Direct
  - Messages delivered to queues according to routing keys. (No patterns in routing key allowed)
- Topic
  - The same as previous, but allows pattern matching in routing key.
  - Two previous types are just specializations of topic exchange.

Code example: Publisher

```
with conn.channel() as chan:
    chan.exchange_declare(exchange = EXCHANGE,
                          type = EXCHANGE_TYPE)

    chan.queue_declare(queue = QUEUE)

    msg = amqp.Message(msg_body)
    chan.basic_publish(msg,
                       exchange = EXCHANGE,
                       routing_key = ROUTING_KEY)
```

Code example: Consumer

```
with conn.channel() as chan:

    def on_msg_recv(msg):
        print msg

    chan.exchange_declare(exchange = EXCHANGE,
                          type = EXCHANGE_TYPE)
    queue = chan.queue_declare(QUEUE)
    chan.queue_bind(exchange = EXCHANGE,
                    queue = QUEUE,
                    routing_key = ROUTING_KEY)
    chan.basic_consume(queue = QUEUE,
                       callback = on_msg_recv)

    while True:
        chan.wait()
```

Visualization

- STOMP adapter for RabbitMQ
- Orbited server
- Infovis visualization library (Javascript)

Thank you

- Thank you for your attention!

Code examples and presentation:
https://github.com/AlexanderAA/nzpug