Development notes

Thoughts, notes and ideas about development

RabbitMQ Cheat Sheets

NOVEMBER 6, 2020 / 3 MIN READ / ALEXEY BOGDANOV

This is a small cheat sheets for *RabbitMQ*. More detailed explanations and detailes examples can be found in the official tutorial and documentation. Links can be found at the bottom of this blog post.

`RabbitMQ` is a lightwight and easy to deploy message broker: it accepts and forwards messages. It's written in *Erlang* language.

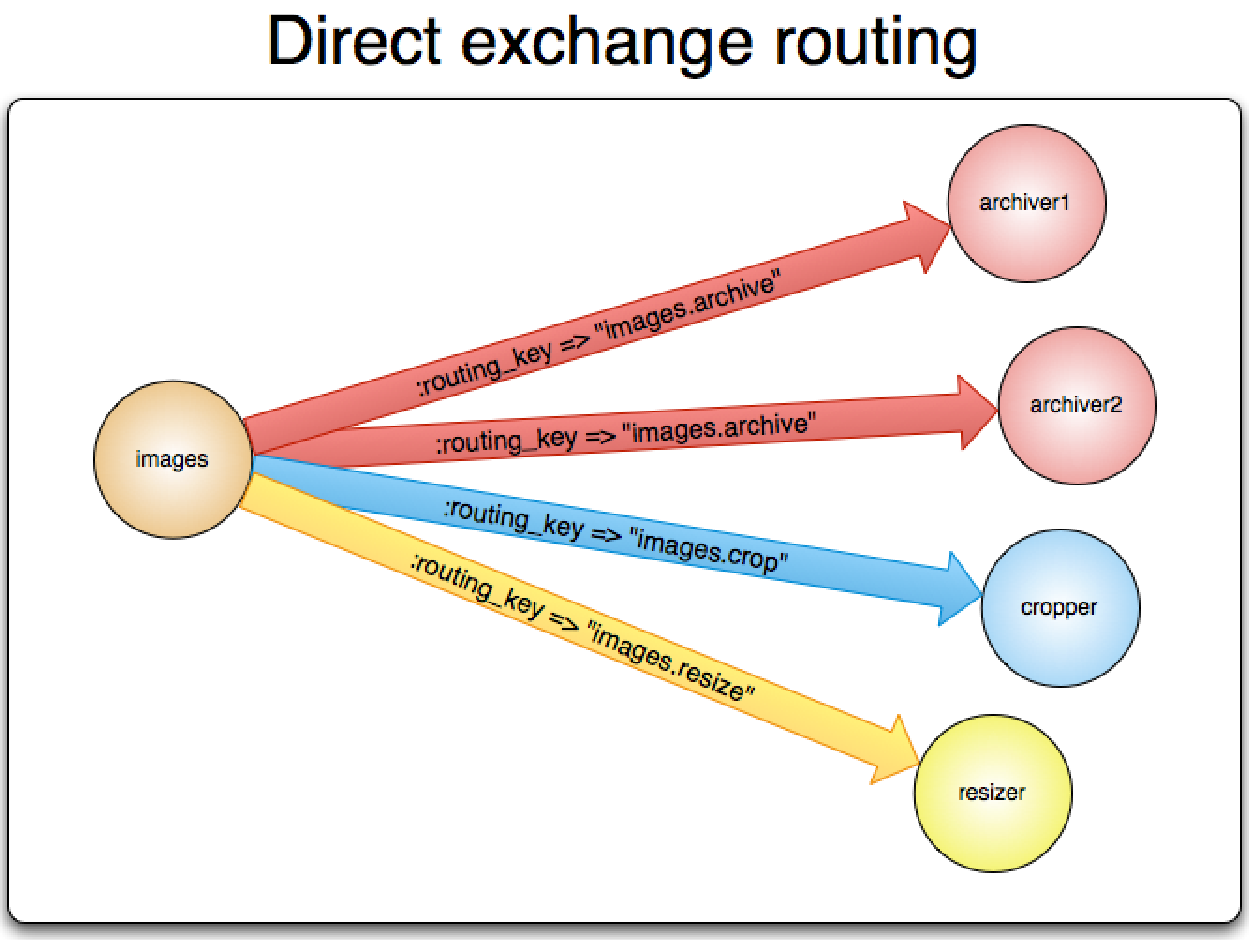
- `Producer` - a programm that sends messages.
- `Consumer` - a programm that reads messages.
- `Queue` - a buffer that stores messages.
- `Bindings` - rules that exchanges use to route messages to queues.
- `Exchange` - takes a message from producers and routes it into zero or more queues.

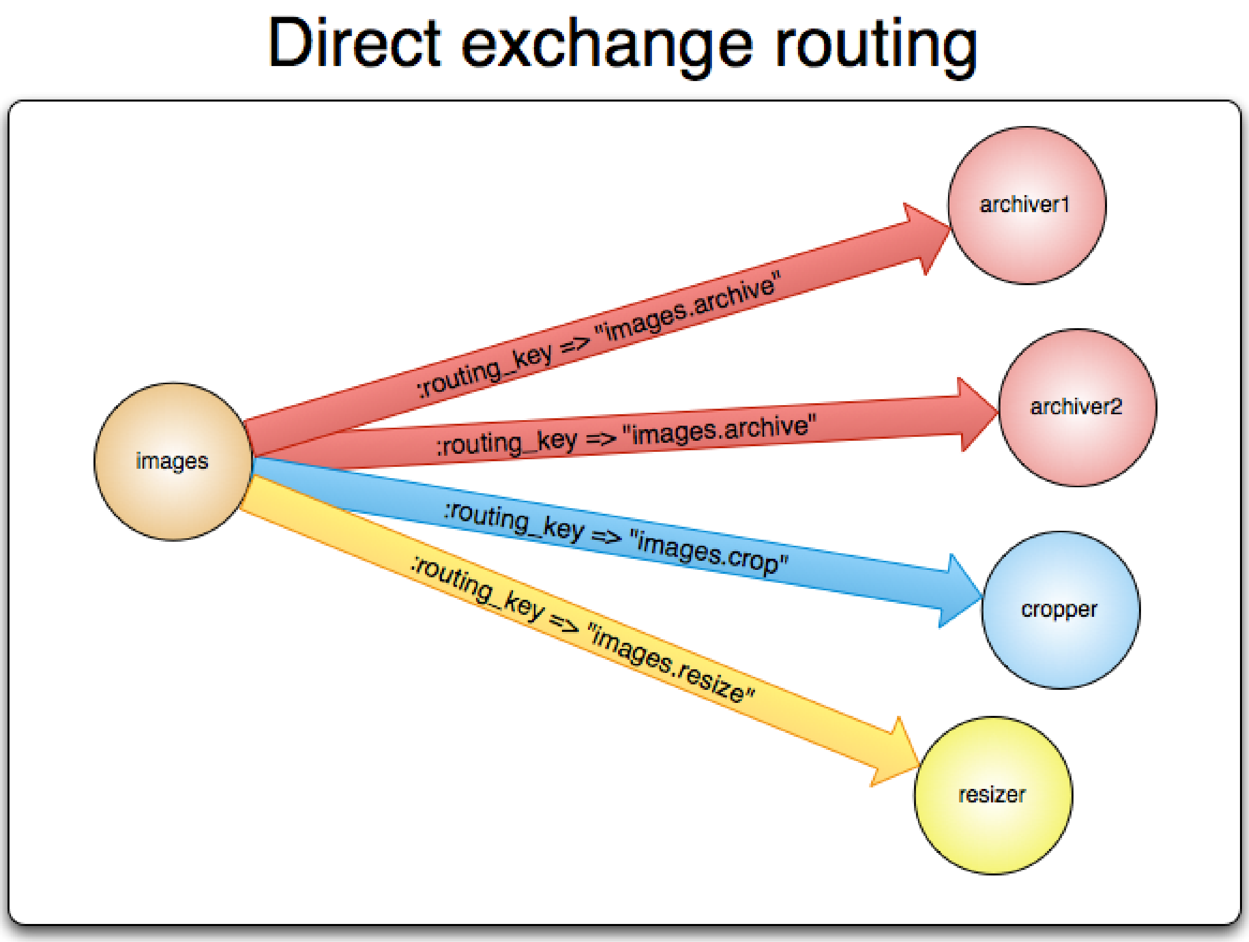
Exchange types

- `default` - exchange with no name pre-declared by the broker. Every queue that is created is automatically bound to it with a routing key which is the same as the queue name.
- `direct` - delivers messages to queues based on the message routing key. It's ideal for the unicast routing of messages. It's often used to distribute tasks between multiple workers.

- `topic` - routes messages to one or many queues based on matching between a message routing key and the pattern that was used to bind a queue to an exchange. The limitation for `routing key` is **255 bytes**.
- `fanout` - routes messages to all of the queues that are bound to it and the routing key is ignored
- `headers` - routes multiple attributes that are more easily expressed as message headers than a routing key. `routing key` is ignored.

Examples for Exchange types:


Direct exchange routing
images
:routing_key => "images.archive"
archiver1
:routing_key => "images.archive"
archiver2
:routing_key => "images.crop"
cropper
:routing_key => "images.resize"
resizer


Direct exchange routing
images
:routing_key => "images.archive"
archiver1
:routing_key => "images.archive"
archiver2
:routing_key => "images.crop"
cropper
:routing_key => "images.resize"
resizer

Exchange tyes are well explained *here*.

Binding keys

- `* (star)` can substitute for exactly one word.
- `# (hash)` can substitute for zero or more words.

When a queue is bound with `"#" (hash)` binding key - it will receive **all the messages**, regardless of the routing key - like in fanout exchange.

When special characters `"*" (star)` and `"#" (hash)` aren't used in bindings, the topic exchange will behave just like a `direct` one.

Message properties

The AMQP 0-9-1 protocol predefines a set of 14 properties that go with a message. Most used properties:

- `persistent` : Marks a message as `persistent` (with a value of `true`) or `transient` (`false`).

- `content_type` : Used to describe the `mime-type` of the encoding. For example for the often used JSON encoding it is a good practice to set this property to: `application/json` .
- `reply_to` : Commonly used to name a `callback` queue.
- `correlation_id` : Useful to correlate RPC responses with requests.

How to run RabbitMQ in Docker

For experimentation with RabbitMQ *community Docker* image can be used:

```
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-
```

Other intallation Guides can be found here: *Downloading and Installing RabbitMQ*

RabbitMQ official docs

- *Official RabbitMQ Tutorials*: detaled explaned with examples for different programming languages.
- *AMQP 0-9-1 Model Explained*
- *AMQP 0-9-1 Quick Reference*

Other Queue related links

- *Understanding When to use RabbitMQ or Apache Kafka*
- *RabbitMQ vs Kafka Series Introduction*. Series of blog posts, webinars about RabbitMQ and Kafka.
- *Developing Transactional Microservices Using Aggregates, Event Sourcing and CQRS - Part 1*
- *Developing Transactional Microservices Using Aggregates, Event Sourcing and CQRS - Part 2*

`RabbitMQ` / `Queue`