Tìm hiểu RabbitMQ - Phần 2

rabbitMQ 2 message broker 2 queue 4

manhdung viết ngày 04/06/2015

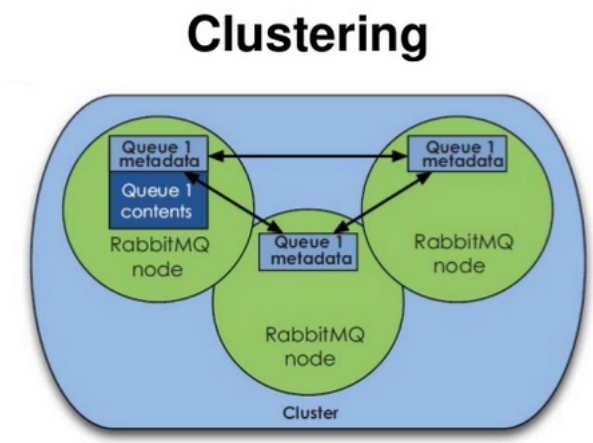
Trong phần 1, tôi đã giới thiệu về sơ lược rabbitmq, vai trò của rabbitmq trong hệ thống phân tán và hướng dẫn cài đặt. Trong phần này, tôi sẽ trình bày cách về cluster và cấu hình cluster trong rabbitmq

Cluster là gì

Cluster là nhóm các thành phần mà hoạt động cùng với nhau để cung cấp một dịch vụ nào đó. Thành phần ở đây gọi là một node. Mỗi node này là một process hoạt động. Thường thì node được đồng nhất với một server do mỗi node thường được cài đặt trên một server riêng rẽ (để tránh bị chết chùm). Khái niệm cluster này xuất hiện trong rất nhiều kiến trúc như galera, mysql, redis... Mục đích sử dụng cluster là để load balancing, high availibility (đảm bảo hệ thống vẫn hoạt động khi có sự cố), scale hệ thống.

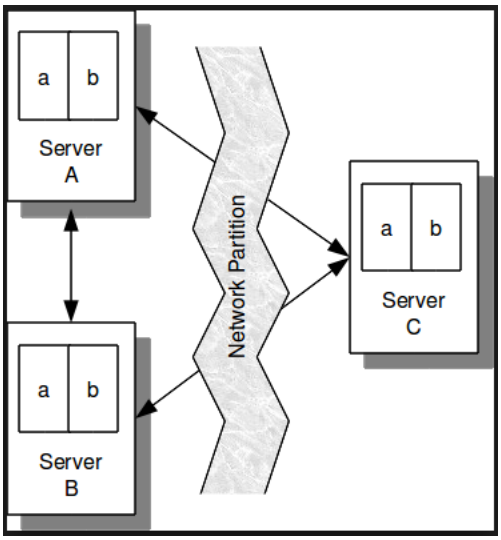
Cluster trong rabbitmq

Trong rabbitmq, một cluster là một nhóm các erlang node làm việc cùng với nhau. Mỗi erlang node có một rabbitmq application hoạt động và cùng chia sẻ tài nguyên: user, vhost, queue, exchange...



Một số đặc điểm cần chú ý

- Metadata của một node được replicate đến các node còn lại trong cluster ngoại trừ queue. Queue được tạo ra trên node nào thì vẫn nằm trên node đó, không có replicate gì hết nhưng bạn hoàn toàn có thể nhìn thấy một queue tạo ra trên một node khi truy xuất qua các node còn lại do đó đối với client một cluster rabbitmq chẳng khác gì một single rabbitmq.
- Vì queue không được replicate nên bản thân cluster rabbitmq chưa cung cấp high availibility (HA). Bạn vẫn cần cấu hình thêm chút nữa nhưng cluster là tiền đề để rabbitmq có thể thực hiện được HA.
- Một node có thể là disc node (mặc định) hoặc ram node
- Như trong tài liệu của rabbitmq có khẳng định, rabbitmq không xử lý tốt network partition nên không khuyến khích sử dụng cluster của rabbitmq trên WAN. Có thể bạn sẽ thắc mắc network partition là gì ? Một network partition hay còn gọi là split brain là tình huống rất hay gặp trong hệ cluster.



Khi network partition xảy ra, hệ cluster bị chia đôi và mỗi phần không thể liên lạc được với phần còn lại nên đâm ra bản thân từng partition lại cứ ngỡ nó là toàn bộ cluster. Vấn đề chính là ở đây. Mỗi partition khi đó sẽ không đồng bộ được với phần còn lại nên chúng ta sẽ có hai tập dữ liệu riêng biệt trên mỗi partition. Nguyên do dẫn đến network partition thường là network không ổn định hoặc quá tải trên node (Khi quá tải do CPU hoặc IO bị nghẽn network của server thường rất chập chờn). LAN network so với WAN network thì thường ổn định hơn nhiều nên rabbitmq cluster thích hợp khi các node được liên kết với nhau qua LAN network. Về network partition, tôi sẽ trình bày trong một phần khác.

- Clustering rabbitmq chỉ là một trong ba cách cấu hình hệ phân tán rabbitmq. Clustering phù hợp cho môi trường LAN network còn với môi trường WAN network thì các mô hình như federation hay shovel lại được khuyến khích. Trong trường hợp của tôi, tôi không sử dụng rabbitmq trong môi trường WAN nên hai mô hình shovel và federation tôi không trình bày.

Điều kiện để thiết lập clustering

- Tất cả các node phải cùng erlang version và rabbitmq version
- Các node liên kết qua LAN network
- Tất cả các node chia sẻ cùng một erlang cookie

Trong mô hình cluster, các node sử dụng phương thức trao đổi giữa của erlang. Khi sử dụng phương thức này, hai erlang node chỉ nói chuyện được với nhau khi có cùng erlang cookie. Erlang cookie chỉ là một chuỗi ký tự. Khi startup một rabbitmq server lần đầu tiên, mặc định một erlang cookie ngẫu nhiên được sinh ra nằm trong `/var/lib/rabbitmq/.erlang.cookie`

Làm viêc với cluster

Thực hiện tạo cluster

Chuẩn bị ba rabbitmq server thỏa mãn đủ điều kiện.

Đầu tiên, chúng ta cần chọn một node, sau đó copy erlang cookie của node đó sang các node còn lại.

Vì node name của rabbitmq có dạng `rabbit@hostname` nên bạn cần chuẩn bị sẵn hostname cho mỗi node. Hostname là bất cứ string nào bạn muốn nhưng tốt nhất đừng sử dụng các ký tự đặc biệt trong này. Sau đó đưa vào `/etc/hosts` của cả ba node:

```
1
2.168.3.252 rabbit3  ### gọi là node 3
```

Như bạn thấy, tôi lựa chọn cách đặt tên an toàn để tránh rắc rối.

Bước 1:
Khởi động rabbitmq-server trên mỗi node.
Trên mỗi node, bạn có thể chạy

```
service rabbitmq-server start
```

hoặc

```
rabbitmq-server -detached
```

Trong cách thứ hai, bạn có thể sẽ gặp một warning:
Warning: PID file not written; -detached was passed.
Đừng quá lo lắng. Trong manual của rabbitmq-server có cho biết khi chạy với tham số -detached server process sẽ hoạt động ở chế độ background và điều đó khiến cho pid của process không được ghi vào pid file.

Bước 2:
Chọn một node làm khởi điểm sau đó các node còn lại sẽ join với node khởi điểm để hình thành lên cluster.

Trước khi thực hiện, chúng ta thử xem cluster status của từng node trước khi join với nhau. Trên mỗi node, bạn thực hiện lệnh `rabbitmqctl cluster_status`

Và đây là kết quả:

Trên rabbit@rabbit1

```
bit1 ...
nodes,[{disc,[rabbit@rabbit1]}]}]
```

Trên rabbit@rabbit2

```
bit2 ...
nodes,[{disc,[rabbit@rabbit2]}]}]
```

Trên rabbit@rabbit3

```
bit3 ...
nodes,[{disc,[rabbit@rabbit3]}]}]
```

Có thể thấy mỗi node đang là một cluster riêng biệt. Tôi sẽ phải gom cả ba node này để hình thành một cluster duy nhất.

Giả sử, tôi chọn node rabbit@rabbit2 làm node khởi điểm. Với vai trò node khởi điểm, node rabbit@rabbit2 sẽ không cần cấu hình gì thêm.

Đảm bảo node 2 đang chạy

```
tl start_app

status
t2 ...
,

"rabbit@rabbit2">>},
{partitions,[]}]
```

Tôi cần join node 1 với node 2

```
stop_app

it2
bit@rabbit2 ...
```

```
art_app
arting node rabbit@rabbit1 ...
```

Làm tương tự với node 3

```
stop_app
it2
bit@rabbit2 ...
art_app
arting node rabbit@rabbit3 ...
```

Bước 3:
Xem cluster status trên mỗi node.

Trên node 1

```
tatus
,
t@rabbit3,rabbit@rabbit1]},
"rabbit@rabbit2">>},
partitions,[]}]
```

Trên node 2

```
tatus
,
t@rabbit3,rabbit@rabbit2]},
"rabbit@rabbit2">>},
{partitions,[]}]
```

Trên node 3

```
tatus
,
t@rabbit2,rabbit@rabbit3]},
"rabbit@rabbit2">>},
partitions,[]}]
```

Bạn có thể cấu hình cluster trong config. Trên mỗi node, bạn chỉ cần khai báo dòng sau trong /etc/rabbitmq/rabbitmq.config

```
{cluster_nodes, {['rabbit@rabbit1','rabbit@rabbit2','rabbit@rabbit3'], disc}}
```

Nhận xét:
Có thể thấy tất cả các node đã nhìn thấy nhau. Tất cả đều là disc node (mặc định). Tất cả các node đều đang running. Cluster_name được lấy theo node name của node khởi điểm và không có một partition nào trong cluster.

Một node trong cluster có thể bị stop/start (dùng rabbitmqctl stop_app/rabbitmqctl start_app trên chính node đó) đồng nghĩa ngừng cung cấp dịch vụ trong cluster nhưng bản thân node đó vẫn không bị loại khỏi cluster

Thực hiện restart cluster

Chúng ta sẽ restart lần lượt từng node.

Giả sử tôi thực hiện restart theo thứ tự sau:
stop node 3 -> stop node 1 -> start node 3 -> start node 1. Node 2 vẫn giữ hoạt động để đảm bảo dịch vụ không down.

Stop node 3, node 1

Trên node 3:

```
opping and halting node rabbit@rabbit3 ...
```

Bạn có thể dùng `service rabbitmq-server stop` thay cho `rabbitmqctl top`

Trên node 1:

```
r stop
opping rabbitmq-server: rabbitmq-server.
```

Xem cluster status trên node 2:

```
r_status
bit@rabbit2,rabbit@rabbit3]}]},
{running_nodes,[rabbit@rabbit2]},
```

```
"rabbit@rabbit2">>},
partitions,[]}]
```

Chỉ còn một mình node 2 đang hoạt động

Start node 3, node 1

Trên node 3:

```
tmq-server start
q-server: SUCCESS
bbitmq-server.
```

Trên node 1:

```
tmq-server start
q-server: SUCCESS
bbitmq-server.
```

Xem cluster status trên node 2:

```
r_status

,
t@rabbit3,rabbit@rabbit2]},
"rabbit@rabbit2">>},
{partitions,[]}]
```

Như vậy, ngay sau khi được start trở lại, các node sẽ tự động tham gia vào cluster và running luôn.

Trong các trường hợp có sự cố nghiêm trọng như toàn bộ các node đều down lần lượt hoặc tất cả đều down đồng thời thì quy trình start cluster lại hơi khác một chút. Chúng ta đi vào từng trường hợp một.

Trường hợp thứ nhất: Tình huống xảy ra khi bạn cần restart cluster để upgrade cho rabbitmq hoặc erlang. Sau khi node 1, node 2 được bạn stop thì thảm họa xảy ra với node còn lại. Node còn lại bị down ngoài ý muốn. Trong trường hợp này việc khởi động lại cluster đòi hỏi thứ tự: Node cuối cùng bị down phải là node đầu tiên được start. Giả sử các node bị down theo thứ tự: node 3 -> node 1 -> node 2. Sau đó tôi cố gắng start các node 3 hoặc node 1 đầu tiên. Tôi sẽ không thành công. Rabbitmq để lại vài dòng log sau:

```
ning.
rst, then
e on other
uster nodes after this one was shut down may be lost.
```

Để khởi động được cluster, bạn chỉ cần tuân theo nguyên tắc, start node 2 đầu tiên. Với các node sau, thứ tự không quan trọng. Bạn có thể dùng thứ tự node 2 - > node 1 -> node 3 hoặc node2 -> node 3 -> node1.

Trường hợp thứ hai: Cũng giống trường hợp một nhưng đáng tiếc là node 2 bị sự cố quá nghiêm trọng không thể phục hồi được. Vậy là node cuối cùng không thể boot được. Lúc này bạn phải ép một node không phải node down cuối cùng làm node khởi điểm

```
ia/rabbit@rabbit1 ...
itmq-server start
q-server: SUCCESS
bbitmq-server.
```

Sau đó bạn khởi động lại các node kế tiếp.

Trường hợp thứ ba: Khủng khiếp hơn ! Bạn chẳng làm gì nhưng cụm server mà chứa rabbitmq cluster bị crash đột ngột. Lúc này thì bạn chẳng thể biết node nào down trước hay down sau cả. Cách xử lý giống hệt trường hợp thứ hai

Rời một node khỏi cluster

Cách 1: Để cho bản thân node đó quên rằng nó đã từng ở trong cluster. Giả sử tôi muốn tách node 2 khỏi cluster hoàn toàn.

```
tl stop_app

reset

start_app

status
t2 ...
,
"rabbit@rabbit2">>},
partitions,[]}]
```

Để reset thành công, bạn không được config cluster trong file cấu hình.
Reset đồng thời sẽ xóa mọi data của node 2 như vhost, user, exchange, queue...

Cách 2: Làm cho các node còn lại trong cluster hắt hủi node cần được tách khỏi cluster :(

```
[root@rabbit2 root]# rabbitmqctl stop_app
Stopping node rabbit@rabbit2 ...

[root@rabbit3 root]# rabbitmqctl forget_cluster_node rabbit@rabbit2
Removing node rabbit@rabbit2 from cluster ...
```

Lúc này các node còn lại trong cluster đều đã không coi node 2 nằm trong cluster nhưng node2 vẫn không chịu chấp nhận thực tế phũ phàng đó. Nếu bạn start_app node 2

```
Error: {error,{inconsistent_cluster,"Node rabbit@rabbit2 thinks it's clustered with node rabbit@rabbit3, but rabbit@rabbit3 disagrees"}}
```

Để node 2 hoạt động được bình thường, bạn phải làm nó quên đi nó từng thuộc về cluster.

```
l reset
start_app
status
t2 ...
,
"rabbit@rabbit2">>},
partitions,[]}]
```

Thêm một node vào cluster

```
tl stop_app
abbit3
@rabbit3 ...
start_app
s
,
t@rabbit3,rabbit@rabbit2]},
"rabbit@rabbit2">>},
partitions,[]}]
```

Nếu cố join_cluster từ một running node, bạn sẽ gặp:

```
Error: mnesia_unexpectedly_running
```

Thêm một RAM node

So sánh RAM node với disc node
Sự khác biệt lớn nhất là ram node chỉ giữ metadata của nó trong memory còn bản thân các queue data vẫn lưu xuống disk. Sự khác biệt này cho phép ram node ít tạo ra các hoạt động IO hơn nên performance tốt hơn disc node. Một cluster hoàn toàn chỉ có ram node thì rất có nguy cơ mất metadata. Giải pháp an toàn hơn cả là trộn lẫn ram node và disc node. Trong cluster, phần metadata được replicate giữa các node (disc node lưu metadata trên disk) nên sẽ không lo mất sạch metadata.

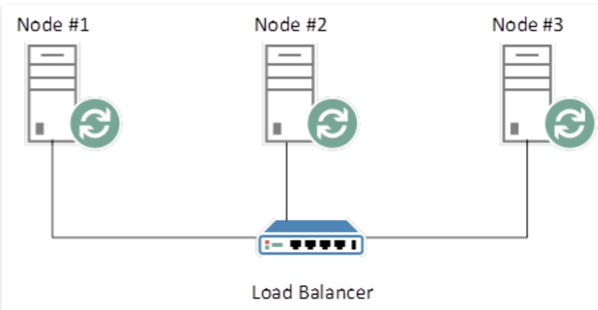
```
tl stop_app
bbit@rabbit3
@rabbit3 ...
start_app
status
bit2]}]},
t@rabbit3,rabbit@rabbit2]},
"rabbit@rabbit2">>},
partitions,[]}]
```

Thay đổi node type

```
tl stop_app
ster_node_type disc
.
start_app
status
,
t@rabbit3,rabbit@rabbit2]},
"rabbit@rabbit2">>},
stop_app
uster_node_type ram
start_app
Starting node rabbit@rabbit2 ...
```

```
status
bit2]}]},
t@rabbit3,rabbit@rabbit2]},
"rabbit@rabbit2">>},
{partitions,[]}]
```

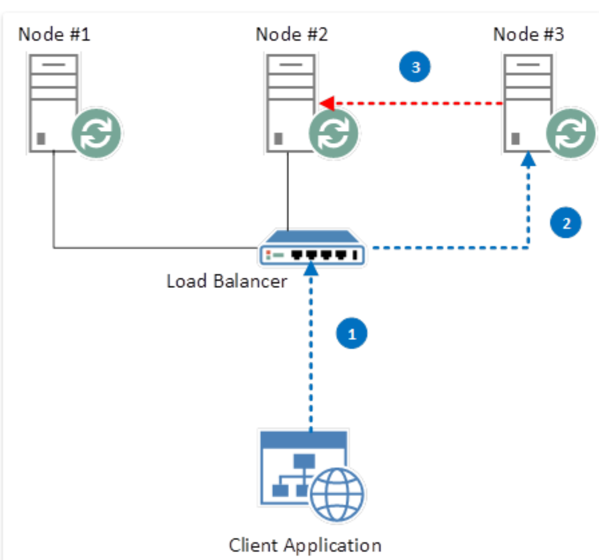
Mô hình load balancing với rabbitmq cluster



RabbitMQ Cluster with Load Balancer

Nếu bạn cấu hình HA queue, queue data giữa các node sẽ được đồng bộ (Trong phần tới tôi sẽ trình bày cách cấu hinh HA cho queue). Nghe có vẻ rất giống mysql replication phải không. Nhưng thực tế thì không giống vậy. Trong hệ rabbitmq cluster, khi áp dụng HA policy, một queue sẽ có master và các slave. Queue được tạo ra trên node nào thì queue đó sẽ là master các replicate của queue đó trên các node còn lại sẽ là slave queue. Tính chất master-slave không áp dụng trên cụ thể một node mà trên cụ thể một queue. Một node có thể là master với queue này nhưng lại là slave với queue khác. Điểm thú vị là request đến queue đi từ client qua load balancer sẽ được chuyển hướng đến node mà chứa master queue

Xem hình minh họa ở dưới



RabbitMQ Cluster Exhibiting Extra Network-hop

Master queue nằm trên node 2 (nơi queue đó được tạo ra). Request từ client đến queue đó đi qua load balancer đập vào node 3 nhưng node 3 sẽ thay vì tự phục vụ luôn thì chuyển hướng request đến node 2 (nơi chứa master queue). Vì nguyên tắc hoạt động này mà HA policy trên rabbitmq không giúp chia tải trên các node được mà chỉ cho phép đảm bảo dịch vụ vẫn vận hành khi có sự cố. Khi node chứa master queue bị down, rabbitmq cluster sẽ promote node chứa slave queue có thời gian hoạt động lâu nhất lên làm master.

Vậy có cách nào khắc phục để chia tải giữa các node không ?

- Bạn có thể tạo queue đều trên tất cả các node trong cluster do vậy các master queue sẽ trải đều trên toàn bộ cluster nên hạn chế các extra network hop giống như trong hình minh họa. Cách này bạn vẫn có thể sử dụng load balancer để điểm trup cập từ client được tập trung.
- Client có một danh sách các queue và biết được master queue nằm trên node nào để truy cập trực tiếp. Cách này sẽ không còn cần đến load balancer nữa.

Vấn đề timeout

Bản thân client sẽ luôn giữ kết nối đến rabbitmq. Sẽ không có timeout nếu như bạn kết nối trực tiếp nhưng khi qua một proxy thì vấn đề xuất hiện. Proxy sẽ không giữ kết nối liên tục giữa client và backend nên trong quá trình sử dụng bạn có thể thấy hiện tượng client bị mất kết nối sau một quãng thời gian không sử dụng. Đáng tiếc rabbitmq client không có cơ chế reconnect lại.

Một linux client có cơ chế tự động gửi lại keep-alive packet để duy trì kết nối nhưng quãng thời gian này quá lâu. `cat /proc/sys/net/ipv4/tcp_keepalive_time` trả về giá trị 7200 nghĩa là cứ sau 2 tiếng mới có một cú gửi keep-alived. Muốn proxy duy trì kết nối thì keep-alived packet phải được gửi trước khi timeout của proxy kết thúc. Trong trường hợp của tôi proxy là haproxy. Tôi điều chỉnh chút ít về cấu hình. Tôi bổ sung ba dòng sau vào cụm backend rabbitmq

```
                    3h
tion          clitcpka
```

Kết thúc phần hai. Trong phần tới, tôi sẽ trình bày về network partition trong rabbitmq cluster

Nguồn tham khảo:
http://insidethecpu.com/2014/11/17/load-balancing-a-rabbitmq-cluster/
https://stackoverflow.com/questions/10461808/how-to-load-distribution-in-rabbitmq-cluster
https://deviantony.wordpress.com/2014/10/30/rabbitmq-and-haproxy-a-timeout-issue/

Bình luận

Vu Nhat Minh hơn 3 năm

Bài viết thật là công phu quá @.@

Hay
1

Tú Phạm hơn 3 năm

Hay quá, xin cảm ơn bác.

Hay
1

Cuong Nguyen hơn 3 năm

cám ơn bạn vì những bài viết hay. tiếp tục chia sẻ nhé.

Hay
1

Vũ Đình Hiệp hơn 3 năm

Hóng part 3, 4 mà bác bận quá nên chưa viết nốt

Hay
1

manhdung hơn 3 năm

ừm dạo này tớ bận, chắc sang tháng sau tớ thu xếp thời gian :D

Hay
0

Dat gần 3 năm

hay qua

Hay
0

Tuan Nguyen gần 3 năm

quá hay

Hay
0

Minhphu hơn 2 năm

Cám ơn vì bài viết quá công phu, hóng phần 3 👍

Hay
0

manhapt hơn 2 năm

Cảm ơn tác giả về chuỗi bài viết về rabbitMQ dễ hiểu, chi tiết.
Không có nút Like cho bài viết nhỉ, vì nhiều khi cảm thấy hay muốn like nhưng nếu ai cũng comment là hay thì loãng comment quá. Đề nghị kipalog cho thêm button like cho bài viết :)

Hay
0

kuthanhitc gần 2 năm

Cảm ơn bạn rất nhiều, bài viết quá hay & công phu

Hay
0

Nguyen Hoang Ha 10 tháng

Rất cảm ơn tác giả. Bài viết rất chất lượng. Tiếc là chưa có bài tiếp theo 😄

Hay
0

bonghongthuytinh 6 tháng

@manhdung Anh ơi trở lại đi anh :(

Hay
0

Đăng nhập để bình luận :)

manhdung

44 bài viết. 274 người follow

Kipalog Follow

Cùng một tác giả

82 11

Tìm hiểu RabbitMQ - Phần 1

rabbitMQ queue message broker

Giới thiệu RabbitMQ là một message broker (messageoriented middleware) sử dụng giao thức AMQP Advanced Message Queue Protocol (Đây là giao thức ph...

manhdung viết hơn 3 năm trước

82 11

48 4

Tìm hiểu phần cứng qua các thông tin server

hardware Server

Giả định bạn tiếp nhận một server mới toanh, bạn cần tìm một số thông tin về nó như loại CPU, loại main, loại memory, memory dùng của hãng nào... c...

manhdung viết 2 năm trước

48 4

41 7

Giới thiệu MongoDB

mongodb

Giới thiệu MongoDB là một giải pháp nosql database. Data được lưu ở dạng các bson document. Hỗ trợ vertical scaling và horizontal scaling, dynamic...

manhdung viết hơn 3 năm trước

41 7

Bài viết liên quan

15 14

Send mail with Laravel, gmail

Laravel Gmail smtp queue

1. Tản mạn ngoài lề Như chúng ta đã biết việc gửi mail về cho người dùng trong web là một vấn đề phổ biến. Điển hình như gửi confirm mail khi mem...

Văn Đức Thái viết 4 tháng trước

15 14

82 11

Tìm hiểu RabbitMQ - Phần 1

rabbitMQ queue message broker

Giới thiệu RabbitMQ là một message broker (messageoriented middleware) sử dụng giao thức AMQP Advanced Message Queue Protocol (Đây là giao thức ph...

manhdung viết hơn 3 năm trước

82 11