Part 2: RabbitMQ Best Practice for High Performance (High Throughput)

2018-01-08

Many variables feed into the overall level of performance in RabbitMQ. In **Part 2 of RabbitMQ Best Practice** are recommended setup and configuration options for maximum message passing throughput explained. We will mention standard settings, changes and plugins that can be used to receive a better throughput.

We have been working with RabbitMQ a long time, and we have probably seen way more configuration mistakes than anybody else. We know how to configure for optimal performance and how to get the most stable cluster. In this series are our knowledge shared! Please read part 1 for general best practice and dos and don'ts tips for RabbtitMQ.

Make sure your queues stay short

To get optimal performance, make sure your queues stay as short as possible all the time. Longer queues impose more processing overhead. We recommend that queues should always stay around 0 for optimal performance.

Set a queue max-length if needed

A feature that could be recommended for applications that often get hit by spikes of messages, is to set a max-length on the queue. This will keep the queue short by discarding messages from the head of the queues so that it's never larger than the max-length setting.

Remove the policy for lazy queues

CloudAMQP has enabled lazy queues by default. Lazy queues are queues where the messages are automatically stored to disk. Messages are only loaded into memory when they are needed. With lazy queues, the messages go straight to disk and thereby the RAM usage is minimized, but the throughput time will be larger.

Use transient messages

Persistent messages will be written to disk as soon as they reach the queue, which will affect throughput. Use transient messages for fastest throughput.

Use multiple queues and consumers

LOVISA JOHANSSON
Developer

Queues are single-threaded in RabbitMQ, and one queue can handle up to about 50k messages/s. You will achieve better throughput on a multi-core system if you have multiple queues and consumers. You will achieve optimal throughput if you have as many queues as cores on the underlying node(s).

The RabbitMQ management interface will keep information about all queues and this might slow down the server. The CPU and RAM usage may also be affected in a negative way if you have too many queues (thousands of queues). The RabbitMQ management interface collects and calculates metrics for each and every queue which uses some resources and CPU and disk contention can occur if you have thousands up on thousands of active queues and consumers.

Split your queues over different cores

Queue performance is limited to one CPU core. You will, therefore, get better performance if you split your queues into different cores, and also into different nodes if you have a RabbitMQ cluster. RabbitMQ queues are bound to the node where they were first declared. Even if you create a cluster of RabbitMQ brokers, all messages routed to a specific queue will go to the node where that queue lives. You can manually split your queues evenly between nodes, but the downside is that you need to remember where your queue is located.

We recommend two plugins that will help you if you have multiple nodes or a single node cluster with multiple cores.

Consistent hash exchange plugin

The consistent hash exchange plugin allows you to use an exchange to load-balance messages between queues. Messages sent to the exchange are consistently and equally distributed across many queues, based on the routing key of the message. The plugin creates a hash of the routing key and spread the messages out between queues that have a binding to that exchange. It could quickly become problematically to do this manually, without adding too much information about numbers of queues and their bindings into the publisher.

The consistent hash exchange plugin can be used if you need to get maximum use of many cores in your cluster. Note that it's important to consume from all queues. Read more about the consistent hash exchange plugin here.

RabbitMQ sharding The RabbitMQ sharding plugin will do the partitioning of queues automatically for you, i.e once you define an exchange as sharded, then the supporting queues will be automatically created on every cluster node and messages will be sharded across them. RabbitMQ sharding will show one queue to the consumer, but it could be many queues running behind it in the background. The RabbitMQ Sharding plugin gives you a centralized place where to send your messages, plus load balancing across many nodes, by adding queues to the other nodes in the cluster. Read more about RabbitMQ Sharding here.

Disable manual acks and publish confirms

Acknowledgement and publish confirms has a performance impact, for fastest possible throughput, manual acks should be disabled.

Avoid multiple nodes (HA)

One node will give you the highest throughput, compared to an HA cluster setup. Messages and queues are not mirrored to other nodes.

Enable RabbitMQ HiPE (still marked as experimental)

HiPE will increase server throughput at the cost of increased startup time. When you enable HiPE, RabbitMQ is compiled at start up. The throughput increases with 20-80% according to our benchmark tests. The drawback of HiPE is that the startup time increases quite a lot too, about 1-3 minutes. HiPE is still marked as experimental in RabbitMQ's documentation.

Disable plugins you are not using

You are able to enable many different plugins via the control panel in CloudAMQP. Some plugins might be super nice to have, but on the other hand, they might consume a lot. Therefore, they are not recommended for a production server. Make sure to disable plugins that you are not using.

GUIDE - RABBITMQ BEST PRACTICE

CONTINUE WITH PART 3
RabbitMQ Best Practice for High Availability

GO BACK TO PART 1
RabbitMQ Best practice


docker


heroku