RabbitMQ の分散構成はどうするのが良さそうか？

RabbitMQ

はじめに

モジュール間連携のイベントバスとして RabbitMQ を使用したいのですが，クラスタを組む場合にどうするのが良いのかな？ということを調べてみました *1.

なお，以降では「CAP のうち CP 特性が欲しい」ということを念頭において調査しています.

動作確認について

RabbitMQ は 3.6.1 です．ローカルのマシンに VM を起動して動作確認しています.

https://github.com/krdlab/examples/tree/master/distributed-rabbitmq-brokers

RabbitMQ の分散構成について

分散構成に関する公式の情報は以下の通りです.

- https://www.rabbitmq.com/distributed.html
- https://www.rabbitmq.com/clustering.html
  - https://www.rabbitmq.com/ha.html
- https://www.rabbitmq.com/federation.html
  - https://www.rabbitmq.com/federated-exchanges.html
  - https://www.rabbitmq.com/federated-queues.html
- https://www.rabbitmq.com/shovel.html

Clustering

- 動作環境として LAN を想定している
- 複数のノードを単一のブローカとして構成する
- "All nodes connect to all other nodes in both directions"
- ノードには disk と RAM の 2 種類がある
  - disk ノードはランタイム情報をメモリとディスクの両方に保存
  - RAM ノードはメモリのみに保存
  - ただし `delivery-mode = 2` (persistent) のメッセージは disk/RAM に関係なくディスクに保存される
- どのノードからも exchange や queue を利用 (publish や consume) できる
- queue を除くすべてのデータ/ランタイム情報は，クラスタを構成するすべてのノードにレプリケーションされる
- queue だけは最初に declare したノードに保存される
  - publisher/consumer の接続先と queue のロケーションとの関係によってはルーティングが発生するためスループットに影響を与える

しかしこれだけだと，キューを保持したノードがダウンした場合にそのデータをロストしてしまいます．これを防ぐ仕組みとして mirrored queue があります．

Mirrored Queue

- queue 単位で master/slave が構成される
  - "Each mirrored queue consists of one master and one or more slaves"
  - master queue の配置は設定 (`queue_master_locator`) で変更可能
  - いくつのノードにレプリケーションするかは `ha-mode` で指定
- queue に対する操作は master から slave へ伝搬される
  - master と同じ順序で適用し，同じ状態を維持する
- consumer は接続先のノードに関係なく queue のメッセージデータを消費できる
  - master queue が ACK を受け取ると slave queue からも削除される
- 新しくノードを追加した場合，そのノードの slave queue は空の状態
  - 追加後に publish されたメッセージが蓄積される
  - 強制的に同期可能だが，設定 (`ha-sync-mode`) で自動的に同期させることも可能
- master queue がダウンした場合は最古の slave が昇格する
  - このとき slave が完全に同期しきっていない分のメッセージはロストする

具体的なポリシー設定については Configuring Mirroring に説明があります．

master への昇格が発生した場合，昇格の最中に publish されたメッセージはロストしません．publish は常に master とすべての slave に直接行われているため，新しく master になる slave がメッセージを受けているからです．一方 consume は以下のような影響を受けます．

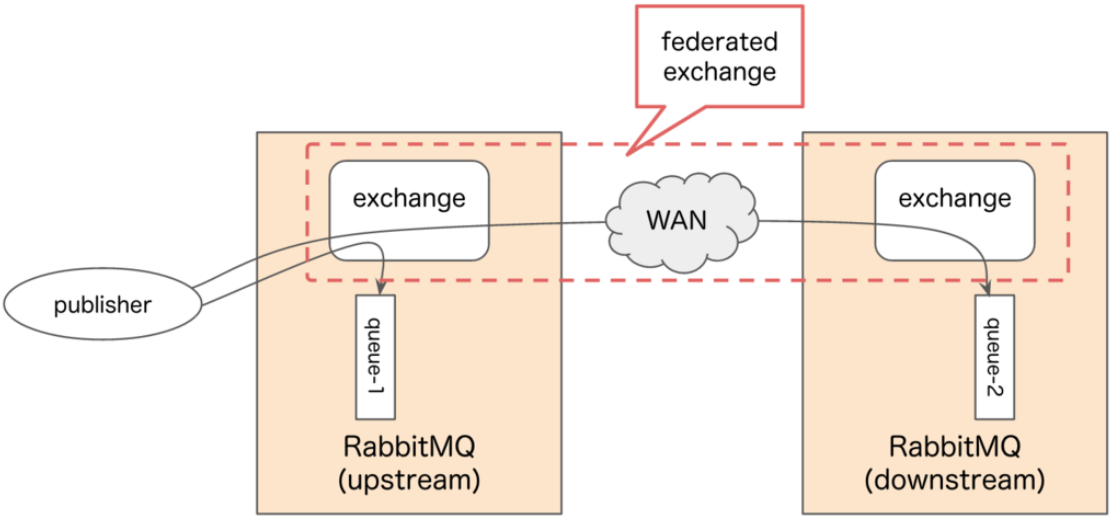
- slave queue と consumer の接続は切断される
  - consumer はこれを検知して re-consume する必要あり
- 以下のいずれかに該当するメッセージは re-consume 時に再配信される
  - consumer からの ACK が master queue に伝わっていなかった
  - master queue のメッセージ削除が slave queue に伝わっていなかった

Federation

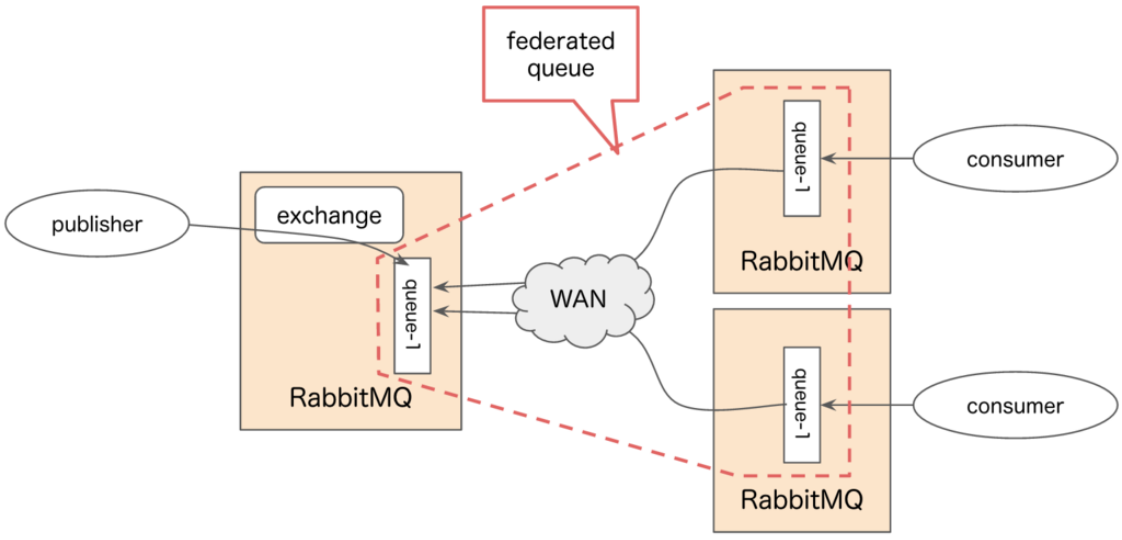
- WAN を想定して設計されている
- clustering とは異なり複数のブローカを「ゆるく」結合する
  - virtual host とか user とかが異なっても良い
  - 指定したポリシに従って結合されるため "all or nothing" ではない
  - 双方向に設定しなくても良い
- federated exchange
  - upstream から downstream に publish する
    - 1 つの upstream から複数の downstream にばらまいたり
    - 複数の upstream を 1 つの downstream に集約したり
  - あくまで publish であり状態を同期しているわけではない

        - downstream 側の消費を upstream は知らない
    - 離れたクラスタ間の pub/sub だけでなく，クラスタを無停止で入れ替えるといった用途でも有用
- federated queue
    - downstream から upstream を consume する
        - upstream 側のキューに積まれたメッセージは消費される
    - downstream 側を複数用意することで分散が可能となる
        - それぞれが別々のメッセージを消費できる
    - worker の job queue として使用するイメージ

federated exchange のイメージはこんな感じ．



federated queue のイメージはこんな感じ．



federation は downstream 側に設定します．`rabbitmq_federation` プラグインを有効化するのも downstream 側だけで良いみたい．また exchange 間を双方向に upstream 設定すると，自分以外のブローカで publish されたメッセージも federated exchange を通して受け取ることができるようになります．

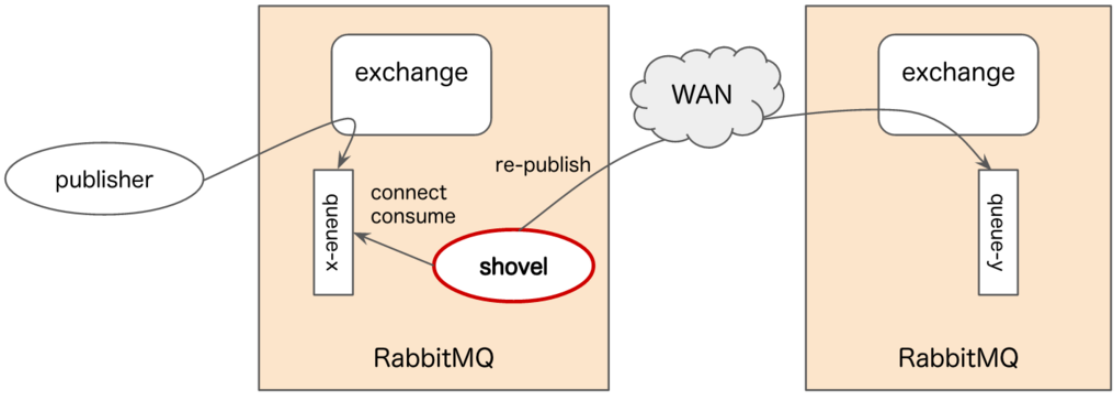
federation の Getting Started には exchange の設定しか載っていませんが，ポリシー設定を `--apply-to queues` にすると federated queue になります．

```
$ sudo rabbitmqctl set_policy --apply-to queues federate-queue "^federation-queue\." '{"fe
```

Shovel

基本的には federation とよく似た役割 (broker から broker へメッセージを移動させる) ですが，細かく対象や付随する動作を設定できるみたいです．

https://www.rabbitmq.com/shovel-static.html



イメージはこんな感じ．

ただ今回の目的から少しズレてしまうので省略．

どれを選択するか？

今回の目的からすると適切なミラーリングを設定した clustering を選択することになります．ただ，

> The network links between machines in a cluster must be reliable

とあるように，LAN のような信頼性の高いネットワーク上に組むことが推奨されています． サービスを AWS 上の Multi-AZ 構成とする場合，クラスタを構成するノードは AZ をまたいでも大丈夫なのか心配になります*2．

実際 mirrored queue にネットワーク分断が発生すると，分断後の各パーティション上にそれぞれ master queue が生じてしまいます． 目的次第ではこれでも構わないと思いますが，今回の目的に限ればこの状態を避けるために孤立したノードには停止して欲しいところです．

Clustering の Network Partitions 発生時の動作について

何も設定しないと分断後もそれぞれのパーティションが動作し続けるのですが，それ以外の動作をさせるための設定もあります．

"Automatically handling partitions" を見ると network partitions が検出された場合の動作を設定できるようです．ハンドリングの設定としては以下の 4 つがあって，それぞれ分断が発生した場合の動作は以下のようになります．

- `ignore`
  - 何もしない
- `pause_minority`

    - 少数派のパーティションに属するノードが停止する
- `pause_if_all_down`
    - 指定したリストのノードとの接続が切れると，接続できなくなった側が停止する
- `autoheal`
    - 自動的に生き残るパーティションが選択され，残りは再起動される

今回の目的に限れば，3 ノード以上でクラスタを構築する場合は `pause_minority` が良さそうです．2 ノードの場合は `ignore` にしてノードへのアクセスを工夫する必要があります．

ちなみに CloudAMQP も 3 ノード構成の場合は `pause_minority` にしているそうです．

https://www.cloudamqp.com/blog/2015-12-29-cloudamqp-plan-setup-pause-minority-mirrored-nodes-and-the-cap-theorem.html

cluster_patition_handling についてもう少しだけ

3 ノード (rabbit1, rabbit2, rabbit3) でクラスタを構築して各設定における動作をざっと確認します．

pause_minority の場合

rabbit1 -> rabbit3, rabbit2 -> rabbit3 の通信を遮断して rabbit3 を孤立させると，rabbit3 はその状態を検出して停止します．

```
=WARNING REPORT==== 13-May-2016::11:16:50 ===
Cluster minority/secondary status detected - awaiting recovery

...

=INFO REPORT==== 13-May-2016::11:16:58 ===
Stopped RabbitMQ application
```

rabbit1 -> rabbit3 の通信のみを遮断した場合は "Partial patition detected" となりますが，やはり (この場合は rabbit3 が) 停止します．

```
=ERROR REPORT==== 13-May-2016::13:59:43 ===
Partial partition detected:
 * We saw DOWN from rabbit@rabbit1
 * We can still see rabbit@rabbit2 which can see rabbit@rabbit1
 * pause_minority mode enabled
We will therefore pause until the *entire* cluster recovers

=WARNING REPORT==== 13-May-2016::13:59:43 ===
```

```
Cluster minority/secondary status detected - awaiting recovery

...

=INFO REPORT==== 13-May-2016::13:59:43 ===
Stopped RabbitMQ application
```

pause_if_all_down の場合

`{pause_if_all_down, ['rabbit@rabbit1'], ignore}` と設定しておきます．この状態で rabbit1 -> rabbit3 の通信を遮断すると，rabbit3 は以下のように停止します．

```
=WARNING REPORT==== 13-May-2016::12:26:17 ===
Cluster minority/secondary status detected - awaiting recovery

=INFO REPORT==== 13-May-2016::12:26:17 ===
Stopping RabbitMQ

=INFO REPORT==== 13-May-2016::12:26:17 ===
Partial partition detected:
 * We saw DOWN from rabbit@rabbit1
 * We can still see rabbit@rabbit2 which can see rabbit@rabbit1
We are about to pause, no need for further actions

...

=INFO REPORT==== 13-May-2016::12:26:17 ===
Stopped RabbitMQ application
```

次に rabbit2 -> rabbit3 の通信を遮断すると，rabbit3 では

```
=ERROR REPORT==== 13-May-2016::12:28:47 ===
Partial partition detected:
 * We saw DOWN from rabbit@rabbit2
 * We can still see rabbit@rabbit1 which can see rabbit@rabbit2
We will therefore intentionally disconnect from rabbit@rabbit1
```

のようにわざと rabbit1 との接続も切って一旦孤立し，自身を停止した後は rabbit2 との通信が回復するまで ERROR REPORT が出続けます．

また `{pause_if_all_down, ['rabbit@rabbit1'], autoheal}` については今のところ挙動がよくわかっていません*3.

autoheal の場合

rabbit1 -> rabbit3 の通信を遮断すると，rabbit1 で以下のような検出ログが出力されます.

```
=ERROR REPORT==== 13-May-2016::12:52:03 ===
Partial partition detected:
 * We saw DOWN from rabbit@rabbit3
 * We can still see rabbit@rabbit2 which can see rabbit@rabbit3
We will therefore intentionally disconnect from rabbit@rabbit2
```

この後，各ノード間で "Autoheal request" を送りあって最終的に勝者が決定されました.

```
=INFO REPORT==== 13-May-2016::12:53:38 ===
Autoheal decision
  * Partitions: [[rabbit@rabbit1],[rabbit@rabbit3,rabbit@rabbit2]]
  * Winner:     rabbit@rabbit3
  * Losers:     [rabbit@rabbit1]
```

負けたノードは再起動されます.

```
=WARNING REPORT==== 13-May-2016::12:53:38 ===
Autoheal: we were selected to restart; winner is rabbit@rabbit3

=INFO REPORT==== 13-May-2016::12:53:38 ===
Stopping RabbitMQ
```

こちらは勝ったノードのログ.

```
=INFO REPORT==== 13-May-2016::12:53:38 ===
Autoheal: I am the winner, waiting for [rabbit@rabbit1] to stop
```

ミラーリング設定について

mirrored queue を有効化するにはクラスタリングを構築した後で `ha-mode` を設定する必要があります. `ha-mode` には以下の 3 種類があります.

- `all`
  - すべてのノードへミラーリング

- `exactly`
  - 指定した数のノードへミラーリング
- `nodes`
  - 指定したノードへミラーリング

ノード数が多くなると `all` はスループットが低下します．PerfTest を使用して計測してみると以下のような傾向がみられました．

```
$ ./runjava.sh com.rabbitmq.examples.PerfTest -a -h 'amqp://guest:guest@rabbit1/%2F' -u ha
```

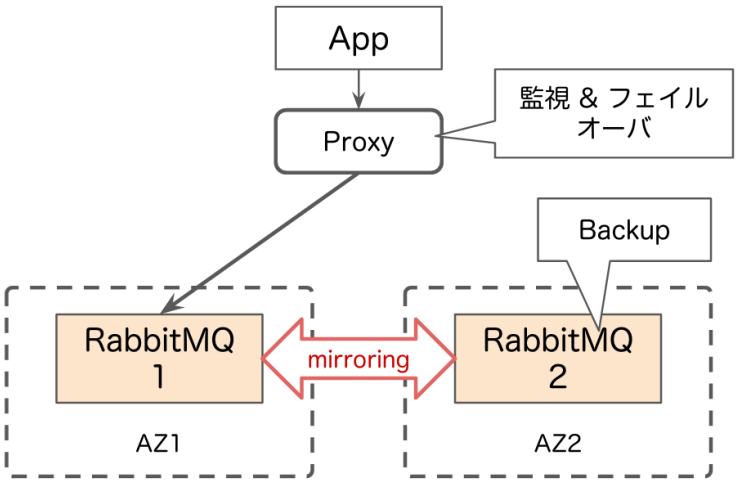
N	msg/sec	N = 1 に対する比
1	14877	1.00
2	4956	0.33
3	3210	0.22

今回は動作確認ということで 1 台のマシンに複数のノードを起動して計測しています．実際に判断を下すためには複数のマシンを用意して計測する必要はありますが，ノード数が多い場合は `exactly` や `nodes` への変更を検討した方が良さそうです．

構成の検討

以上を踏まえ，いくつかのパターンを考えてみます．

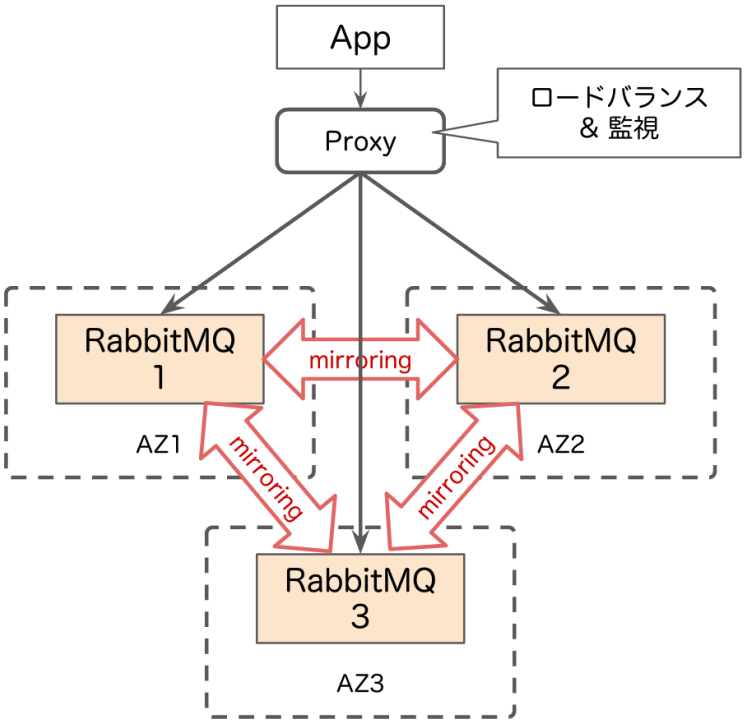
2 台構成



- `{"ha-mode": "all"}`
- `cluster_patition_handling` は設定しない
- ロードバランスしない
- 利用していない方のノードはバックアップノード
- `ha-sync-mode` や `ha-sync-batch-size` は適宜

普段は 1 側のみに接続し，そちらに障害が発生したら 2 側に切り替えます．切り替わった後は 1 側を回復して，今度はこちらがバックアップになります．ロードバランスはしません．network partitions が発生したら，基本的にバックアップ側を切り捨ててリカバリ作業を行います．

3 台構成



- `{"ha-mode": "all"}`
- `{cluster_patition_handling, pause_minority}`
- ロードバランスする
- `ha-sync-mode` や `ha-sync-batch-size` は適宜

network partitions が発生したら，停止したパーティションのノードを切り捨ててリカバリ作業を行います．

もっと多い場合

単純にノードを追加するか，クラスタのトポロジーを変更するか，これはクラスタに担わせる仕事によって変わってくるのだと思います．

単純にノードを追加するにしても構成ノード数は奇数を保って，`ha-mode` は `exactly` の検討をした方が良さそうです．

単一のクラスタにする必要が無ければ，少数ノードによるクラスタを「クラスタグループ」としてこれを複数用意し，シャーディングするのもありかもしれません．

今のところこの規模のクラスタを構築する予定は無いため，必要なタイミングで検討しようと思います．

*1: clustering や mirrored queue については以前調べたのですが，今回改めて調べ直しました

*2: 以前確認したとき AZ 間のラウンドトリップは 2 - 3msec 程度だった (AZ 内だと 0.3 - 0.8msec 程度). WAN というほどではないが LAN よりも遅い感じ.

*3: 各ノードで ERROR が出力され続けて一部のノードが stop ではなくダウンしてしまった