HP

Parameters and Policies

HP

Paramaters Overview

- Some attributes do not mesh well with the use of a configuration file :
  - same across all nodes in a cluster.
  - change at run time
- Parameters

HP

Parameters

- Set by
  - rabbitmqctl
  - HTTP API
- Scoped per vhost.
- Usage → policies
  - optional arguments for queues and exchanges
  - plugins such as Federation and Shovel

HP

Parameter Management

rabbitmqctl

```
rabbitmqctl set_policy federate-me "^amq\." '{"federation-upstream-set":"all"}' --priority 1 --apply-to exchanges
```

```
PUT /api/policies/%2f/federate-me
{"pattern": "^amq\.",
 "definition": {"federation-upstream-set":"all"},
 "priority": 1,
 "apply-to": "exchanges"}
```

HTTP API

```
rabbitmqctl set_parameter shovel my-shovel \
          '{"src-uri": "amqp://", "src-queue": "my-queue", \
           "dest-uri": "amqp://remote-server", "dest-queue": "another-queue"}'
```

HP

Parameter Management

- Reside in the database used by RabbitMQ
- Exported by the management plugin's export feature.
- Used by the federation and shovel plugins.

Policies

- match against exchanges and queues
- determine how EX and Q behave
- at most one policy matching
- applied automatically without the involvement of the client application

Policies

- features controlled by policy are different from arguments.
- matched every time an exchange or queue is created

HP

Policies

- configure
  1. federation plugin,
  2. mirrored queues,
  3. alternate exchanges,
  4. dead lettering,
  5. per-queue TTLs,
  6. and maximum queue length.

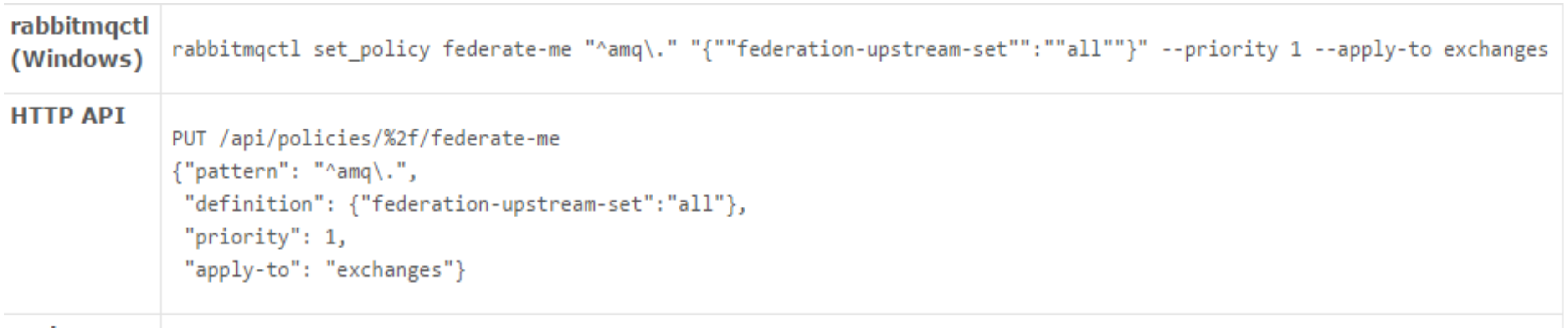
Messages from a queue can be "dead-lettered"; that is, republished to an exchange when any of the following events occur:

- The message is negatively acknowledged by a consumer using `basic.reject` or basic.nack with requeue parameter set to .
- The message expires due to per-message TTL; or
- The message is dropped because its queue exceeded a length limit

HP

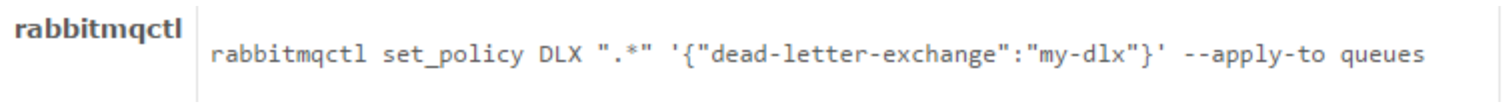
Policies

federation plugin



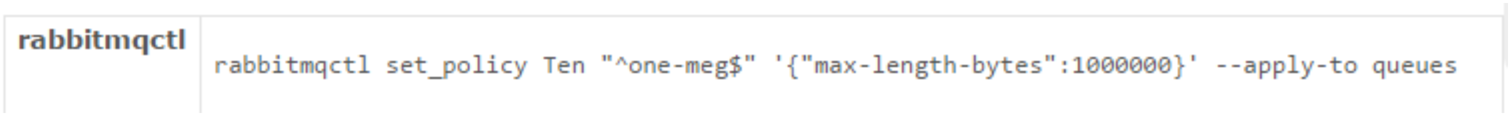
rabbitmqctl (Windows)	`rabbitmqctl set_policy federate-me "^amq\." "{""federation-upstream-set"":""all""}" --priority 1 --apply-to exchanges`
HTTP API	`PUT /api/policies/%2f/federate-me` `{"pattern": "^amq\.",` `"definition": {"federation-upstream-set":"all"},` `"priority": 1,` `"apply-to": "exchanges"}`

Dead Letter Exchanges



rabbitmqctl	`rabbitmqctl set_policy DLX ".*" '{"dead-letter-exchange":"my-dlx"}' --apply-to queues`

Queue Length Limit



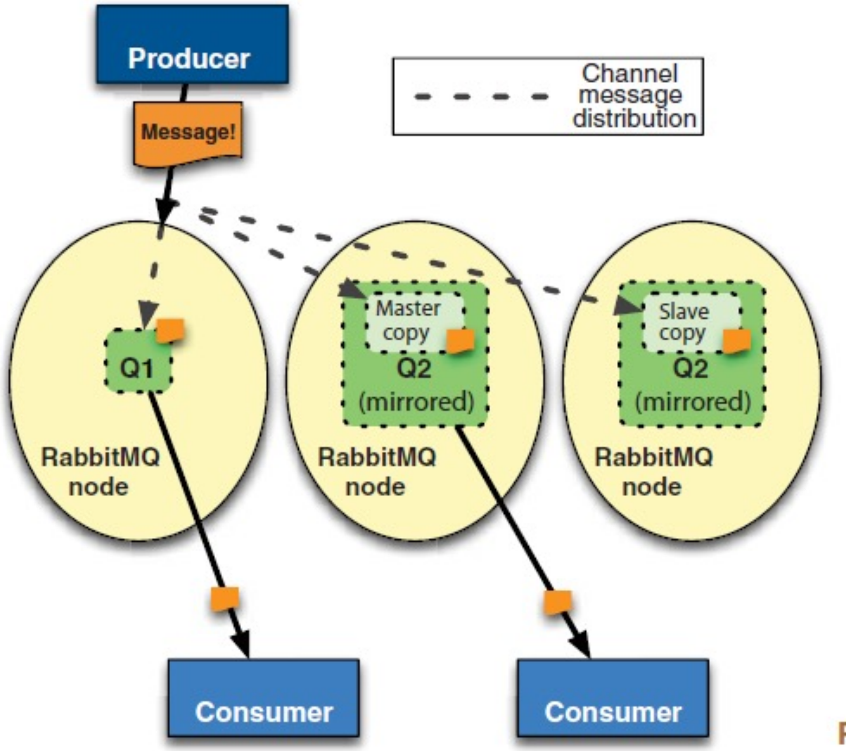
rabbitmqctl	`rabbitmqctl set_policy Ten "^one-meg$" '{"max-length-bytes":1000000}' --apply-to queues`

Mirroring

HP

Highly Available Queues

- Queue's in a RabbitMQ cluster are located on a single node
- Queue can optionally be made *mirrored* across multiple nodes.
- Each mirrored queue consists of one *master* and one or more *slaves*



Highly Available Queues

- Messages are replicated to all slaves.
- Consumers are connected to the master regardless of which node they connect to
- Slaves dropped messages that have been acknowledged at the master.
- Queue mirroring enhances availability, but does not distribute load across nodes (all participating nodes each do all the work).
- This solution requires a RabbitMQ cluster, not recommended for use across a WAN

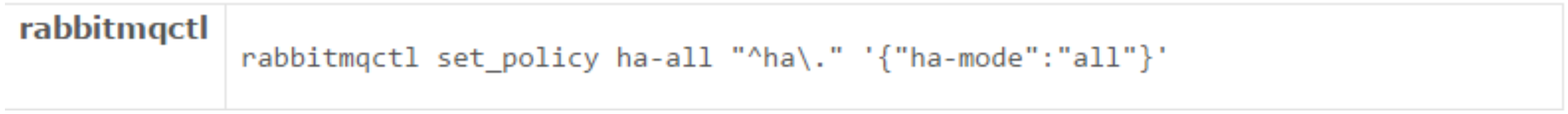
Configuring Mirroring

- Queues have mirroring enabled via policy
- Policies can change at any time
- Steps :
  - create a policy which matches Queue
  - sets policy keys *ha-mode* and (optionally) *ha-params*.

HP

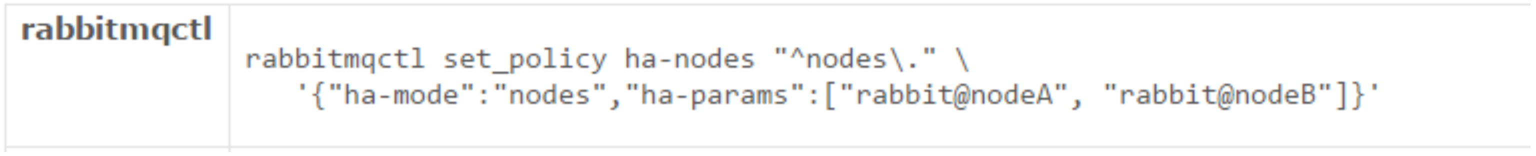
Configuring Mirroring

ha-mode	ha-params	Result
all	(absent)	Queue is mirrored across all nodes in the cluster.
exactly	count	Queue is mirrored to count nodes in the cluster.
nodes	node names	Queue is mirrored to the nodes listed in node names..



rabbitmqctl

```
rabbitmqctl set_policy ha-all "^ha\." '{"ha-mode":"all"}'
```



rabbitmqctl

```
rabbitmqctl set_policy ha-nodes "^nodes\." \
   '{"ha-mode":"nodes","ha-params":["rabbit@nodeA", "rabbit@nodeB"]}'
```

Mirroring - Features

- Queue Master Location
  - All queue operations go through the master first and then are replicated to mirrors.
- Queue masters strategies:
  - using the *x-queue-master-locator* queue declare argument
  - setting the *queue-master-locator* policy key or
  - by defining the *queue_master_locator* key in config

HP

possible strategies

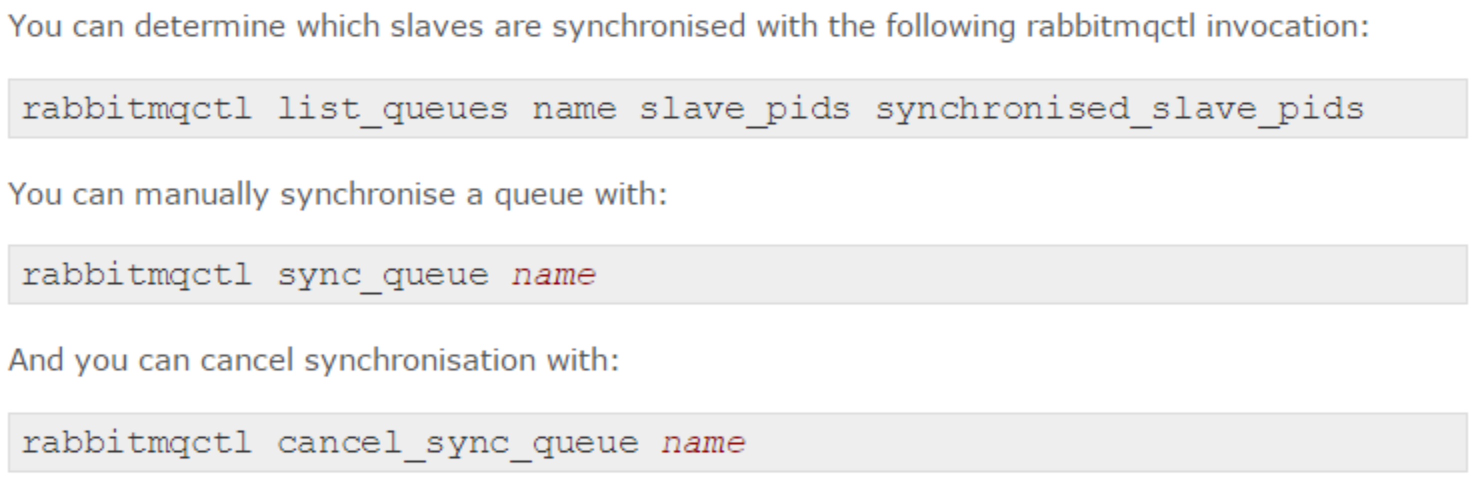
- min-masters
- client-local
- random

```
./rabbitmqctl set_policy qml-policy "^MinMasterQueue\." '{"queue-
master-locator":"min-masters"}' --apply-to queues
```

HP

Unsynchronised Slaves

- A newly added slave provides no additional form of redundancy or availability of the queue's contents that existed before the slave was added, unless the queue has been explicitly synchronised



You can determine which slaves are synchronised with the following rabbitmqctl invocation:

```
rabbitmqctl list_queues name slave_pids synchronised_slave_pids
```

You can manually synchronise a queue with:

```
rabbitmqctl sync_queue name
```

And you can cancel synchronisation with:

```
rabbitmqctl cancel_sync_queue name
```

HP

Lab :Highly Available Queues - Mirror