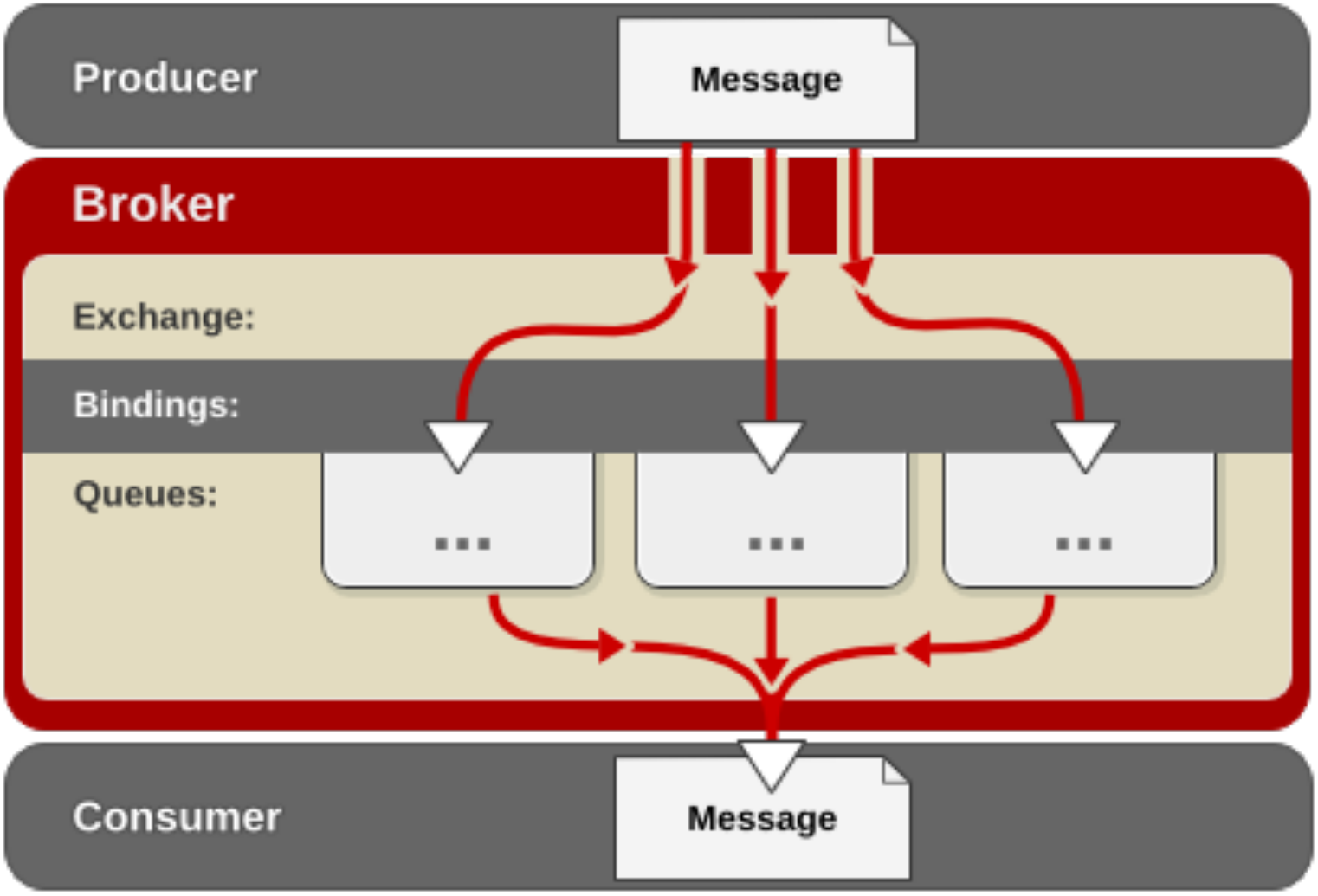
Configuration & Flow

HP

AMQP concepts

Producer Consumer



HP

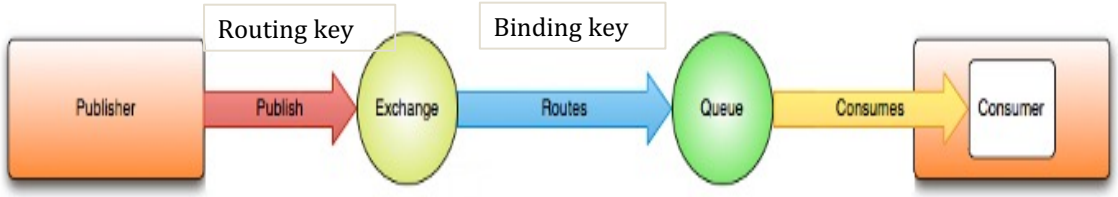
AMQP stack: exchanges, bindings, and queues

- The exchanges are where producers publish their messages.
- queues are where the messages end up and are received by consumers,
- And bindings are how the messages get routed from the exchange to particular queues

HP

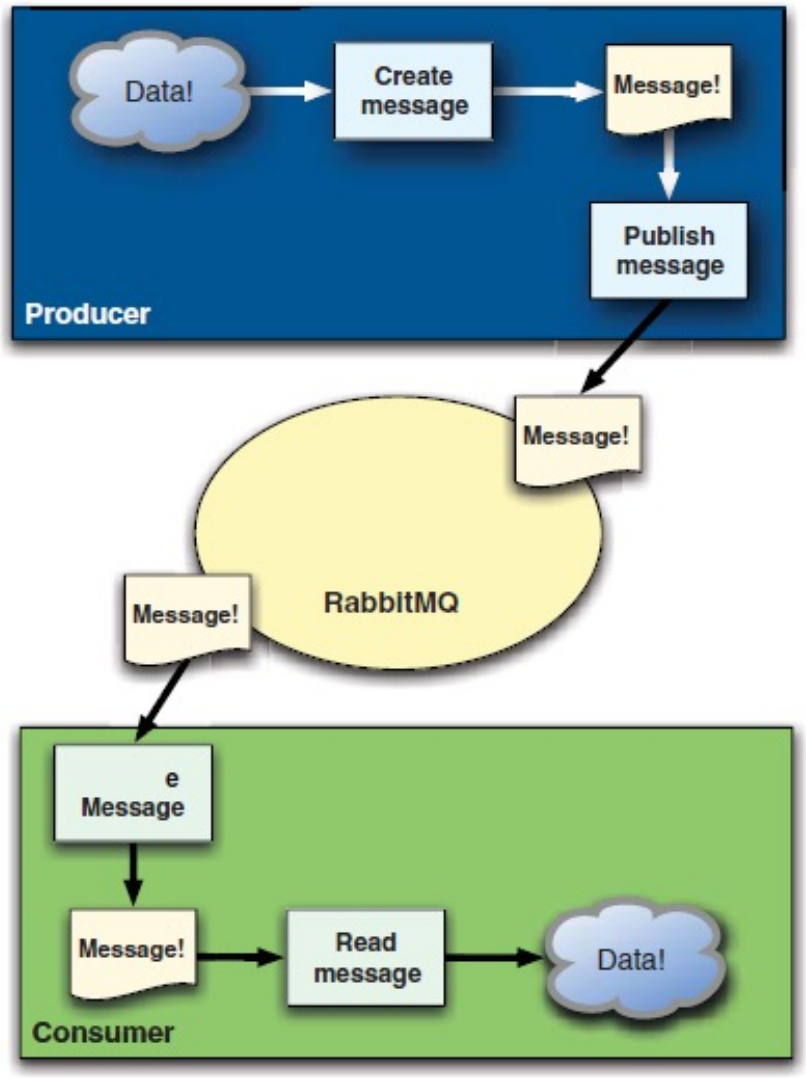
AMQP Models - To get there …

- Queues
  - Messages are consumed from queues
  - Stores messages in memory or disk
  - Deliver in sequence



HP

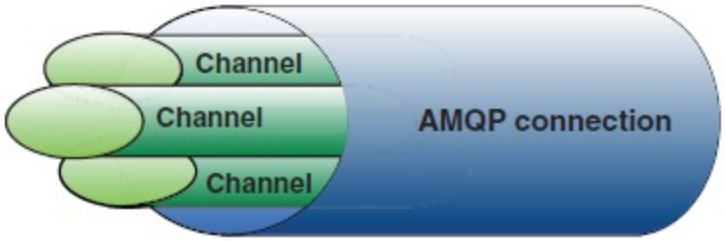
Consumers and producers



HP

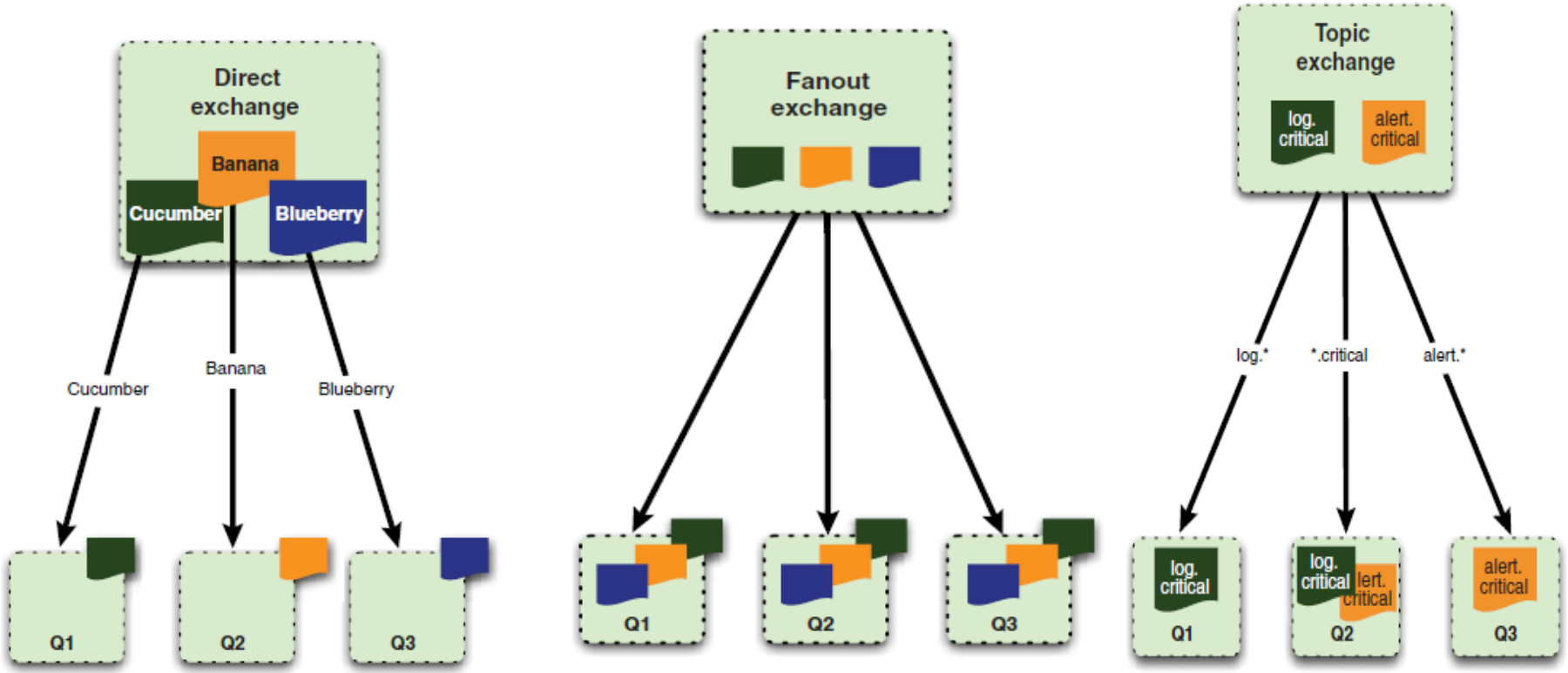
AMQP channels

- a virtual connection inside the “real” TCP connection,
- issue AMQP commands
- a unique ID assigned to it



HP

Types of exchanges



HP

Different types of exchanges

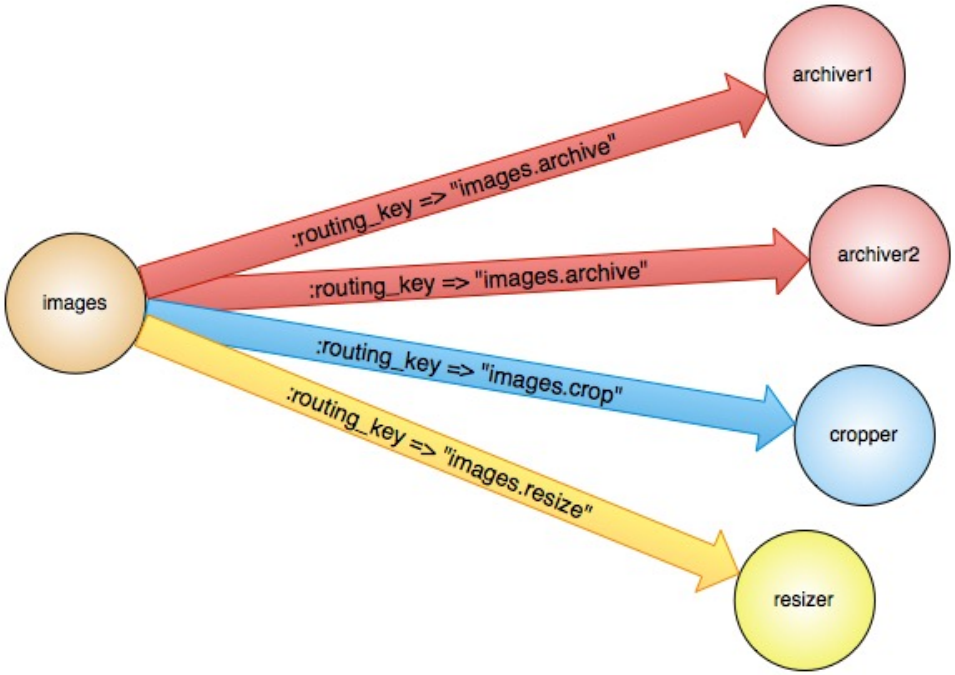
- There are four:
  - direct, fanout, topic, and headers.
- Each implements a different routing algorithm

Exchanges

- Attributes:
  - Name
  - Durability (exchanges survive broker restart)
  - Auto-delete (exchange is deleted when all queues have finished using it)
  - Arguments ->these are broker-dependent

Direct Exchange

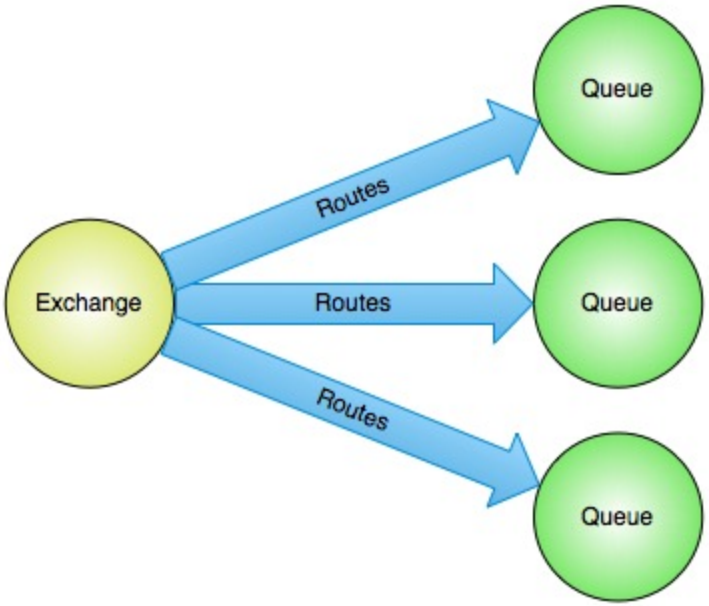
- ideal for the unicast routing of messages
- A queue binds to the exchange with a routing key K
- When a new message with routing key R arrives at the direct exchange, the exchange routes it to the queue if K = R



HP

Fanout Exchange

- routes messages to all of the queues that are bound to it and the routing key is ignored.
- ideal for the broadcast routing of messages.



HP

Topic Exchange

- route 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
- used for the multicast routing of messages.

HP

Headers Exchange

- routing on multiple attributes that are more easily expressed as message headers than a routing key.
- Headers exchanges ignore the routing key attribute.
- Instead, the attributes used for routing are taken from the headers attribute.
- A message is considered matching if the value of the header equals the value specified upon binding.

Queues

- are very similar to queues in other message- and task-queueing systems
- they store messages that are consumed by applications
- Additional properties :
  - Name
  - Durable (the queue will survive a broker restart)
  - Exclusive (used by only one connection and the queue will be deleted when that connection closes)
  - Auto-delete (queue is deleted when last consumer unsubscribes)
  - Arguments (some brokers use it to implement additional features like message TTL)

HP

Queues

- Durable queues are persisted to disk and thus survive broker restarts.
- Queues that are not durable are called transient.
- Not all scenarios and use cases mandate queues to be durable.
- Only *persistent* messages will be recovered.

Bindings

- Bindings are rules that exchanges use (among other things) to route messages to queues.
- To instruct an exchange E to route messages to a queue Q, Q has to be *bound* to E.
- Bindings may have an optional *routing key* attribute used by some exchange types.
- The purpose of the routing key is to select certain messages published to an exchange to be routed to the bound queue.
- In other words, the routing key acts like a filter

HP

Bindings

- Mapping an exchange to a Queue

Name	Default pre-declared names
Direct exchange	(Empty string) and amq.direct
Fanout exchange	amq.fanout
Topic exchange	amq.topic
Headers exchange	amq.match (and amq.headers in RabbitMQ)

Message

- Every message that's received by a consumer is required to be acknowledged
- Ways:
  - basic.ack
  - auto_ack parameter

Message

- Messages in the AMQP model have *attributes*.
  - Content type
  - Content encoding
  - Routing key
  - Delivery mode (persistent or not)
  - Message priority
  - Message publishing timestamp
  - Expiration period
  - Publisher application id

Message

- Messages may be published as persistent, which makes the AMQP broker persist them to disk.
- Publishing messages as persistent affects performance (just like with data stores, durability comes at a certain cost in performance).

HP

Virtual Hosts

- To make it possible for a single broker to host multiple isolated "environments“
  - (groups of users, exchanges, queues and so on),
- AMQP clients specify what vhosts they want to use during AMQP connection negotiation.

HP

Configuration

- RabbitMQ comes with default built-in settings which will most likely be sufficient for running your RabbitMQ server effectively.
- probably won't need any configuration at all.
- provides three general ways to customize the server:
  - **environment variables** **:** define ports, file locations and names (taken from the shell, or set in the rabbitmq-env.conf file)
  - **a configuration file** **:** defines server component settings for permissions, limits and clusters, and also plugin settings.
  - **runtime parameters and policies****:** defines cluster-wide settings which can change at run time

HP

Customise RabbitMQ Environment

- The active configuration can be verified in the broker log, e.g. the active configuration file:

```
config file(s) : /etc/rabbitmq/rabbitmq.config
```

- **Unix**
  - create/edit rabbitmq-env.conf to define environment variables.

```
#example rabbitmq-env.conf file entries
#Rename the node
NODENAME=bunny@myhost
#Config file location and new filename bunnies.config
CONFIG_FILE=/etc/rabbitmq/testdir/bunnies
```

HP

Customise RabbitMQ Environment

- **Windows**
  - Windows dialogue:
    Start > Settings > Control Panel > System > Advanced > Environment Variables

- For environment changes to take effect on Windows, the service must be re-installed. It is not sufficient to restart the service.

RabbitMQ Environment Variables

- Variable names have the prefix RABBITMQ_.
- A typical variable called RABBITMQ_*var_name* is set as follows:
  - a shell environment variable called RABBITMQ_*var_name* is used if this is defined;
  - *otherwise*, a variable called *var_name* is used if this is set in the rabbitmq-env.conf file;
  - *otherwise*, a system-specified default value is used.

HP

RabbitMQ Environment Variables

Name	Default	Description
RABBITMQ_NODE_IP_ADDRESS	the empty string - meaning bind to all network interfaces.	Change this if you only want to bind to one network interface. To bind to two or more interfaces, use the `tcp_listeners` key in `rabbitmq.config`.
RABBITMQ_NODE_PORT	5672	
RABBITMQ_DIST_PORT	RABBITMQ_NODE_PORT + 20000	Port to use for clustering. Ignored if your config file sets `inet_dist_listen_min` or `inet_dist_listen_max`
RABBITMQ_NODENAME	•Unix*: `rabbit@$HOSTNAME` •Windows: `rabbit@%COMPUTERNAME%`	The node name should be unique per erlang-node-and-machine combination.
RABBITMQ_USE_LONGNAME		When set to `true` this will cause RabbitMQ to use fully qualified names to identify nodes. This may prove useful on EC2. Note that it is not possible to switch between using short and long names without resetting the node.
RABBITMQ_SERVICENAME	Windows Service: RabbitMQ	The name of the installed service. This will appear in `services.msc`.

HP

RabbitMQ Environment Variables

- Other variables upon which RabbitMQ depends are:

Name	Default	Description
HOSTNAME	•Unix, Linux: `env hostname` •MacOSX: `env hostname -s`	The name of the current machine
COMPUTERNAME	Windows: localhost	The name of the current machine
ERLANG_SERVICE_MANAGER_PATH	Windows Service:`%ERLANG_HOME%\erts-x.x.x\bin`	This path is the location of`erlsrv.exe`, the Erlang service wrapper script.

Configuration File

- **The rabbitmq.config File**
  - allows the RabbitMQ core application, Erlang services and RabbitMQ plugins to be configured.
  - It is a standard Erlang configuration file.

```
[
  {mnesia, [{dump_log_write_threshold, 1000}]},
  {rabbit, [{tcp_listeners, [5673]}]}
].
```

- **Location of rabbitmq.config and rabbitmq-env.conf**

- Generic UNIX - `$RABBITMQ_HOME/etc/rabbitmq/`
- Debian - `/etc/rabbitmq/`
- RPM - `/etc/rabbitmq/`
- Mac OS X (Macports) - `${install_prefix}/etc/rabbitmq/`, the Macports prefix is usually `/opt/local`
- Windows - `%APPDATA%\RabbitMQ\`

By default, they are not created,

HP

Variables Configurable

- rabbitmq.config

Key	Documentation
`tcp_listeners`	List of ports on which to listen for AMQP connections (without SSL). Can contain integers (meaning "listen on all interfaces") or tuples such as `{"127.0.0.1", 5672}` to listen on one or more interfaces.Default: `[5672]`
`handshake_timeout`	Maximum time for AMQP 0-8/0-9/0-9-1 handshake (after socket connection and SSL handshake), in milliseconds.Default: `10000`
`ssl_listeners`	As above, for SSL connections.Default: `[]`
`ssl_handshake_timeout`	SSL handshake timeout, in milliseconds.Default: `5000`
`vm_memory_high_watermark`	Memory threshold at which the flow control is triggered.Default: `0.4`

rabbitmq.config

Key	Documentation
`disk_free_limit`	Disk free space limit of the partition on which RabbitMQ is storing data. The value may be set relative to the total amount of RAM (e.g.`{mem_relative, 1.0}`). By default free disk space must exceed 50MB. Default: `50000000`
`log_levels`	Controls the granularity of logging. The categories are: •`connection` - for all events relating to network connections •`mirroring` - for all events relating to **mirrored queues** •`federation` - for all events relating to **federation** Default: `[{connection, info}]`
`frame_max`	Maximum permissible size of a frame (in bytes) to negotiate with clients. Setting to 0 means "unlimited“ Default: `131072`

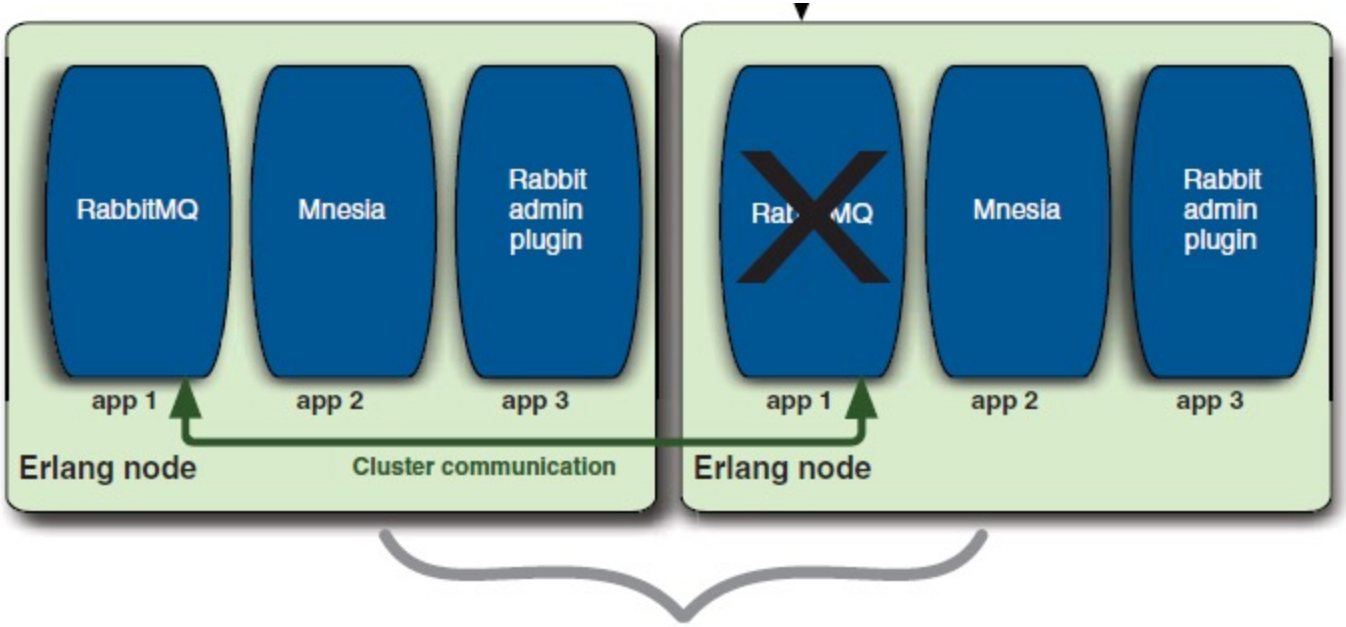
HP

rabbitmq.config

Key	Documentation
`channel_max`	Maximum permissible number of channels to negotiate with clients. Setting to 0 means "unlimited". Using more channels increases memory footprint of the broker. Default: `0`
`heartbeat`	Value representing the heartbeat delay, in seconds, that the server sends in the `connection.tune` frame. If set to 0, heartbeats are disabled. Default: `580`
`default_vhost`	Virtual host to create when RabbitMQ creates a new database from scratch. The exchange `amq.rabbitmq.log`will exist in this virtual host. Default: `<<"/">>`
`default_user`	User name to create when RabbitMQ creates a new database from scratch. Default: `<<"guest">>`

Node - RabbitMq

- ***Starting nodes :***
  - ./rabbitmq-server
  - ./rabbitmq -server -detached.



HP Node - RabbitMq

- ***Stopping nodes:***
  - CTRL-C

```
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
```

  - ./sbin/rabbitmqctl stop

HP

Logging with the firehose tracer

- **Firehose Tracer**
  - administrator can enable (on a per-node, per-vhost basis) an exchange to which publish- and delivery-notifications should be CCed.
  - When the feature is switched off, it has no effect on performance;
  - when it is switched on, performance will drop somewhat due to additional messages being generated and routed.

Firehose

- **Enabling the firehose**
  - Decide which node, and which vhost, you want to enable it for
  - Within your chosen vhost create queues, bind them to the topic exchange amq.rabbitmq.trace, and begin consuming.
  - Run ***rabbitmqctl trace_on***.
- **Disabling the firehose**
  - Run ***rabbitmqctl trace_off.***
  - Clean up the queues used by the firehose.

HP

Firehose : Notification format

- publishes messages to the topic exchange amq.rabbitmq.trace with:
  - routing key either:
    - "publish.exchangename", for messages entering the broker,
    - or "deliver.queuename", for messages leaving the broker;
  - headers containing metadata about the original message:

Header	Type	Description
exchange_name	longstr	name of the exchange to which the message was published
routing_keys	array	routing key plus contents of CC **and** BCC **headers**
properties	table	**content properties**
node	longstr	Erlang node on which the trace message was generated
redelivered	signedint	whether the message has its redelivered flag set (messages leaving the broker only)

HP

Firehose notification format

- body corresponding to the body of the original message
- The rabbitmq_tracing **plugin** **:**
  - builds on top of the tracer
  - provides a GUI to capture traced messages
  - log them in text or JSON format files.

HP

RabbitMQ's logs

- LOG_BASE=/var/log/rabbitmq - rabbitmq-server script
- RabbitMQ will create two log files:
  - RABBITMQ_NODENAME-sasl.log
  - RABBITMQ_NODENAME.log,

HP

rabbit.log file

```
=INFO REPORT==== 10-Sep-2010::13:50:58 ===
accepted TCP connection on 0.0.0.0:5672 from 192.168.1.253:44550

=INFO REPORT==== 10-Sep-2010::13:50:58 ===
starting TCP connection <0.29749.52> from 192.168.1.253:44550

=INFO REPORT==== 10-Sep-2010::13:50:58 ===
closing TCP connection <0.29749.52> from 192.168.1.253:44550

=INFO REPORT==== 10-Sep-2010::13:51:08 ===
Rolling persister log to
"/var/lib/rabbitmq/mnesia/rabbit/rabbit_persister.LOG.previous"
```

HP

ROTATING THE LOGS

- ./rabbitmqctl rotate_logs suffix
- ./rabbitmqctl rotate_logs .1

```
$ ls /var/log/rabbitmq
    rabbit@mrhyde-sasl.log
    rabbit@mrhyde-sasl.log.1
    rabbit@mrhyde.log
    rabbit@mrhyde.log.1
```

Lab : Configuration