



Introduction to queues

# Queues

In computer science, queue is a **collection of entities** maintained in a **sequence**. Sequence can be **modified by**

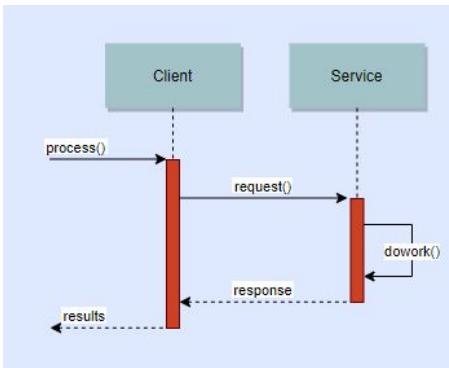
- **adding** entities to the **beginning** or to the **end**
- **removing** from the **beginning** or from the **end**



# Queues - asynchronous communication

## → Synchronous (i.e. HTTP)

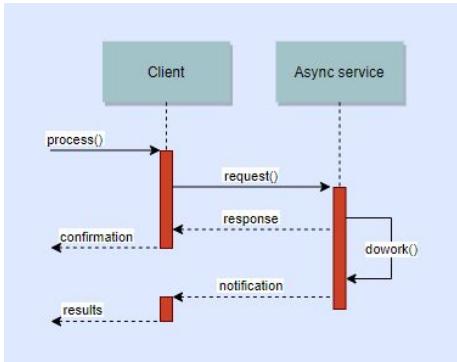
RESTful API, WebServices, RPC etc.



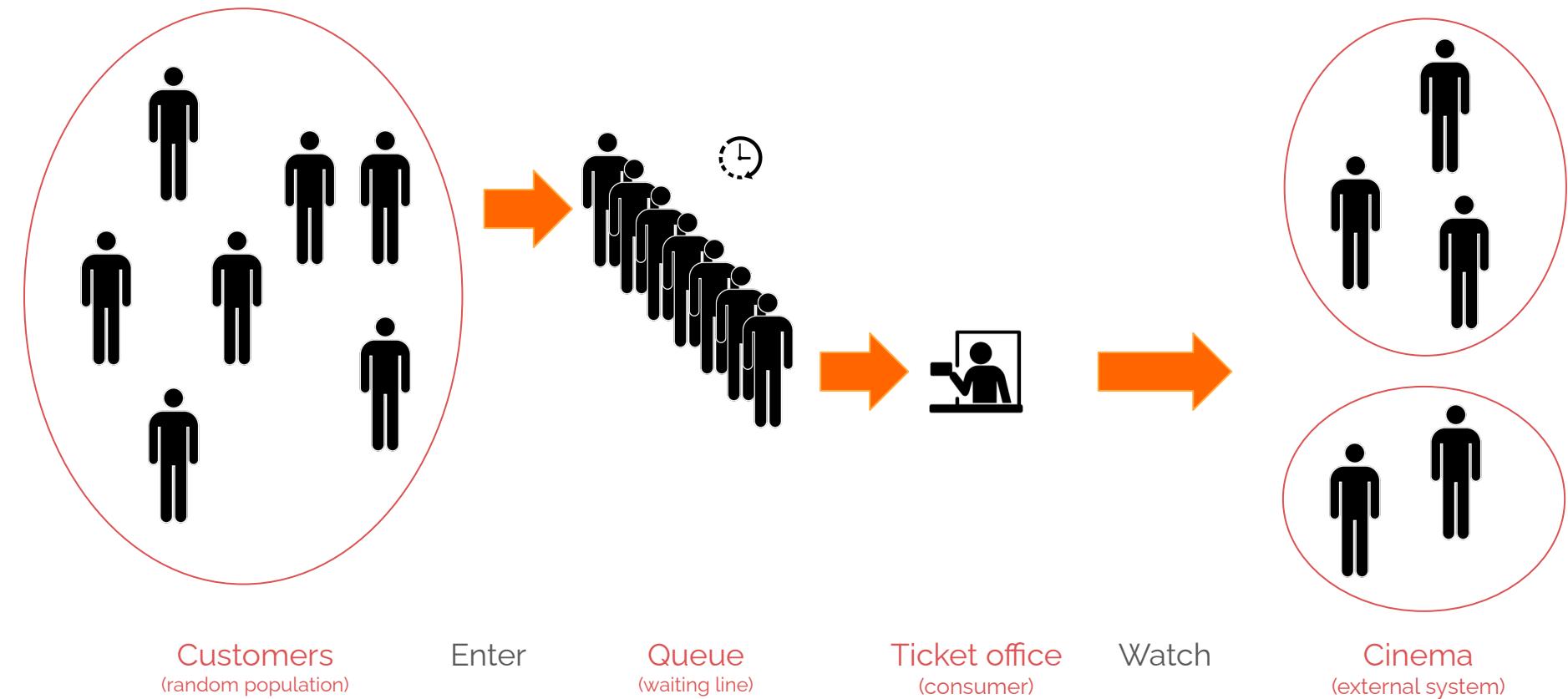
Best for requests with really short execution time

## → Asynchronous (i.e. AMQP)

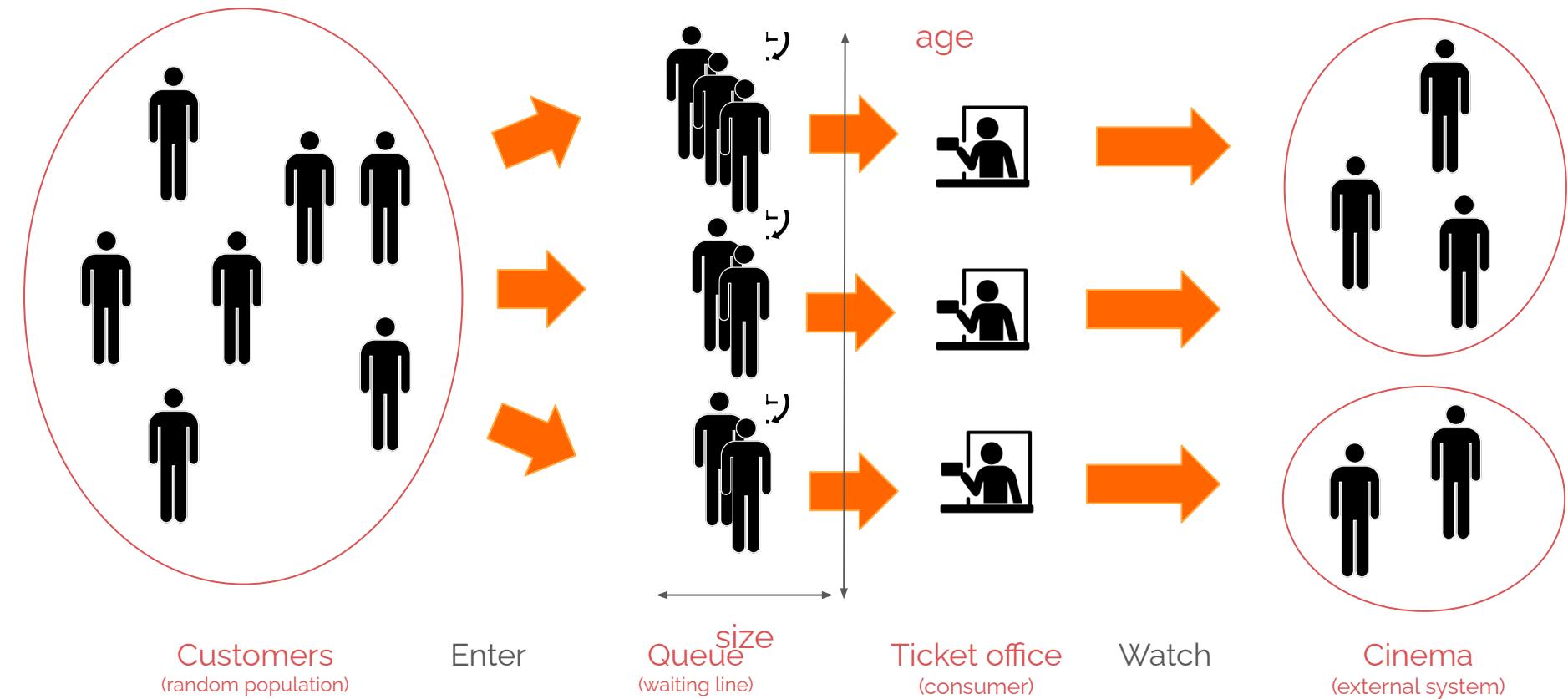
Queues



# Queues - attributes



# Queues - attributes



# Queue attributes

Most important queue attributes for monitoring

## 1. Queue size

Number of messages in the queue

## 2. Queue age (time)

Age of the oldest message in the queue



I see queue with 1 million messages

Should I take any action?

No - if age is 5s (produce and consumption rate is 200 000 messages /s)

I see queue with 10 messages

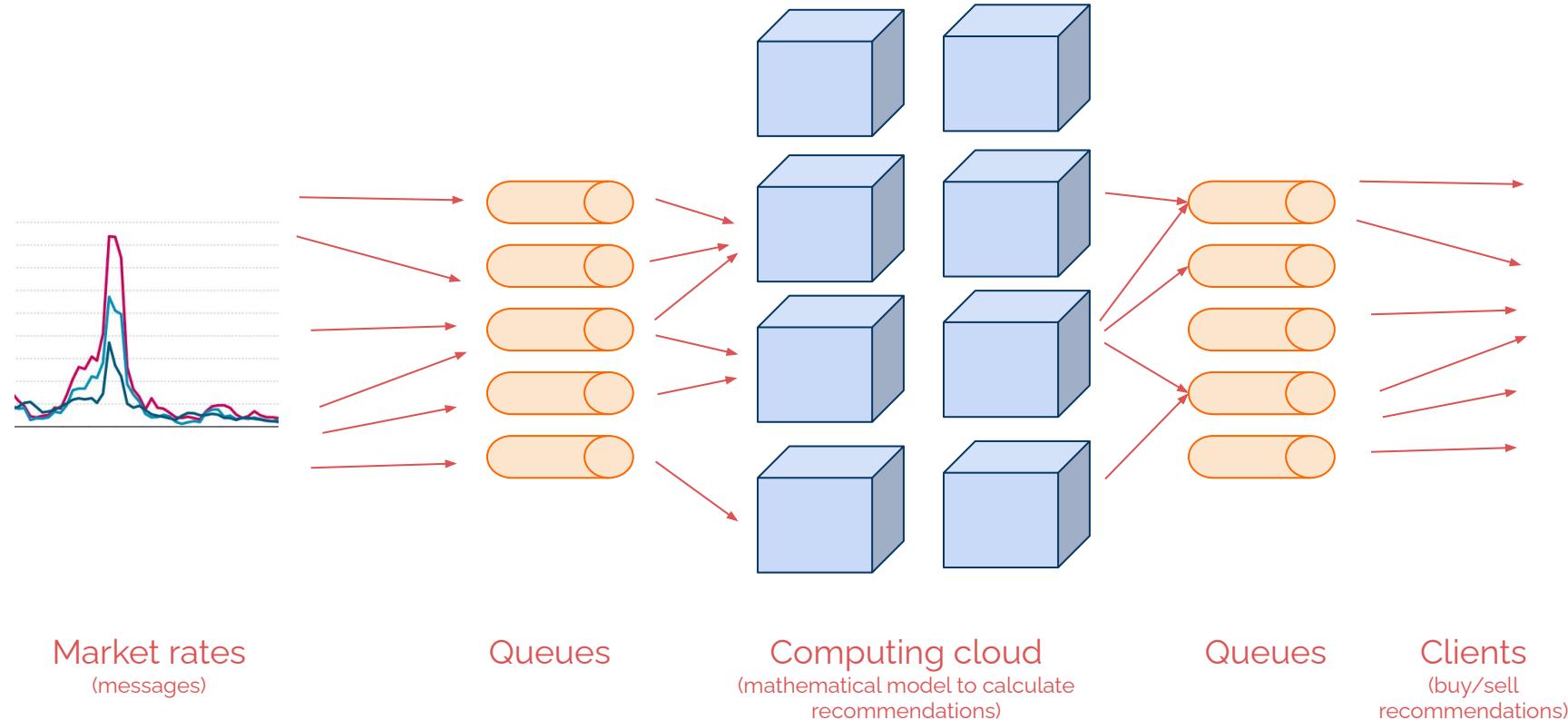
Should I take any action?

Yes - if age is 5 minutes (messages not consumed fast enough or at all)



RabbitMQ doesn't support queue age by default - feature must be implemented separately by publishers and consumers

# Queues - system architecture



# Queues - system architecture

Queues in such architecture play three important roles:

## 1. Send market rates asynchronously

The system that sends market rates doesn't care about calculation results, it only requires confirmation that message been accepted and queued for further calculation.

## 2. System autoscaling

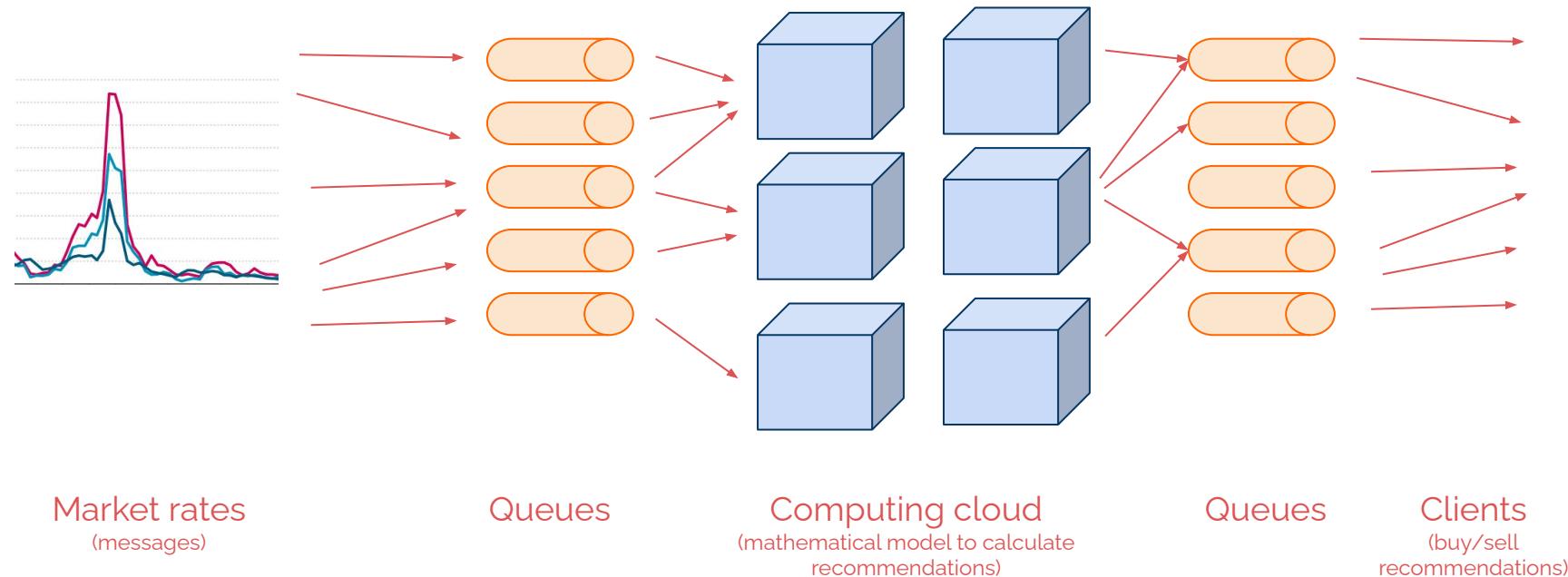
Market rates volume can be unpredictable. Sometimes number of incoming rates per second is small, but sometimes it suddenly grow and more servers are needed to calculate recommendations and meet the **SLA**. Market rates are stored in queues = system has the ability to autoscale itself based on queues **size and age**.

## 3. Reliability

When computing cloud went down, queues will continue to accumulate incoming traffic, and any failure of computing cloud is fully transparent for publishers. When computing cloud is up and running again, message rates accumulated in the queues allow calculations to continue from where they were interrupted and send recommendations to clients without losing any single message. Reliable system should meet the **Single Data Responsibility principle**, commonly known as a **System Component Reliability** to guarantee that we won't lose any single message in case of any component failure.

# System reliability with Queues

Single Data Responsibility Principle (System Component Reliability)



# Use Cases for Queues

Among others, queues are mostly used to:

(not necessarily in order below)

## 1. Decompose two systems

Integrate systems written in different technologies



## 2. Estimate desired system performance,

Throttle or speed up data flows



## 3. Increase fault tolerance level (not the same as high availability),

Persistent queues, "packet lost" issues, single data responsibility principle



## 4. Increases loose coupling and scalability (horizontal scaling, autoscaling)

Many consumers for the same publisher, components don't know each-other (scale-in and scale-out components)



## 5. Increase system's inbound traffic

Asynchronous communication, "fire-and-forget" scenarios



## 6. Buffer system's outbound bytes

Video streaming, ppv data streaming, notifications



# Queues

What is your use-case?



- How do you use queues in your system?
- Do you autoscale horizontally your system?
- Do you face any data loss issues?
- Is your system prepared for any component crash?
- Do you monitor your queues in both dimensions (time & size)?



Introduction to RabbitMQ

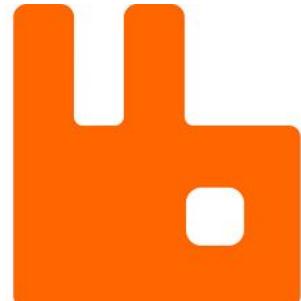
# RabbitMQ

RabbitMQ is an open-source **message-broker software** (sometimes just called queuing software) that originally implemented the **Advanced Message Queuing Protocol (AMQP)**

RabbitMQ supports plugins - standard distribution contains plugins to support protocols like Streaming Text Oriented Messaging Protocol (**STOMP**), Message Queuing Telemetry Transport (**MQTT**), and others.



In this course we focus on AMQP only



# RabbitMQ

RabbitMQ 1.0.0 was released on **1 July 2007**.

Written in Erlang Programming Language. Erlang is a functional language, designed to build massively scalable real-time systems with high availability requirements. Mainly used in telecommunication, but also in banking, e-commerce, computer telephony and instant messaging. Erlang's runtime system has built-in support for concurrency distribution and fault tolerance.



# RabbitMQ - Erlang

Do you think Erlang is an uncommon solution? - all depends from the industry sector.



**WhatsApp** - supports over 450 million users with only 32 engineers



live monitoring of the real-time bidding system



**Pinterest** - more than 30K events per-second



20 million+ users,  
100,000s of concurrent users at peak times



3.2 million active daily users

# RabbitMQ

The concurrency in Erlang is performed by the process, but Erlang processes are slightly different than most people used to. Erlang process are rather an activity or task run in parallel and executed independently from the other processes.

Erlang processes:

- are very light (memory footprint)
- start very fast
- operate in isolation from other processes
- are scheduled by Erlang's Virtual Machine (default limit of Erlang processes for RabbitMQ is 1 million)

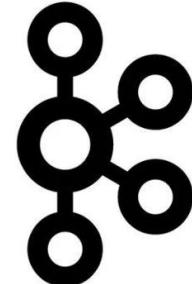


# Similar products



IBM MQ  
Amazon MQ/Kinesis  
Google Cloud Pub/Sub  
Kafka  
Apache ActiveMQ  
Tibco EMS/Messaging/Rendezvous  
Alibaba Message Queue

And many, many, more ....



# RabbitMQ, Kafka, AWS Kinesis?

Data we manage influences the architecture. Any decision must take into account:

- Data throughput
- Data volume
- Team skills
- Cost \$\$\$

	RabbitMQ	Kafka	AWS Kinesis
Released	June 2007	January 2011	November 2013
General purpose	Messages broker (queue)	Messages bus (stream processing - log)	Messages bus (stream processing - log)
Messages replay	No	Yes	Yes
Consumer	Dummy consumer / Push model	Smart consumer / Pull model	Smart consumer / Pull & Push models
Data throughput (relative to single shard/node)	High- (No quoted throughput)	Highest (No quoted throughput)	High+ (one shard can support 1 MB/s input, 2 MB/s output or 1000 records per second)
Data consistency	Highest (ACKs)	High-	High+
Persistence period	No limit	No limit	Up to 7 days
Maintenance effort	High-	Highest	Very Low
Cost	Relatively small	Relatively normal	Highest
Open Source	Yes	Yes	No

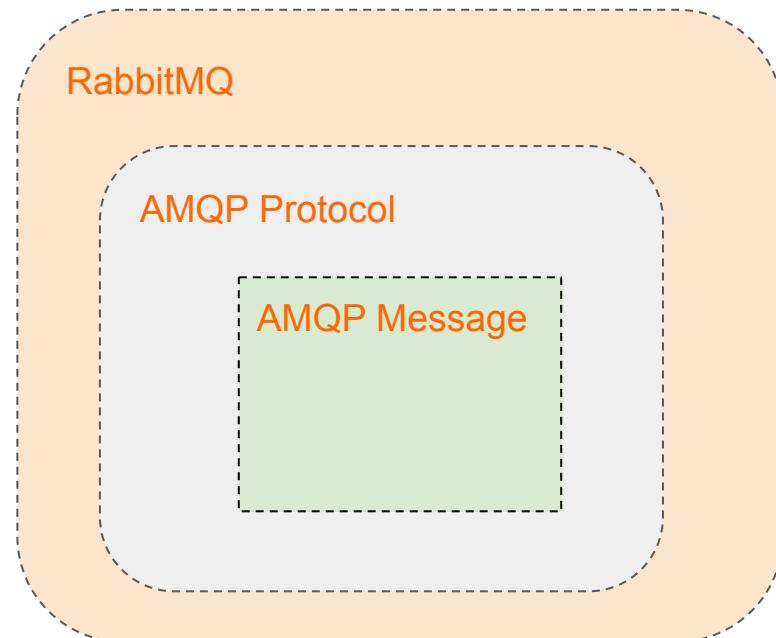
# RabbitMQ vs Kafka - deep dive

	<b>RabbitMQ</b>	<b>Kafka</b>
Distribution	many consumers per queue, but message is consumed only once	consumers distributed by topic partitions, message can be consumed by many consumers
Availability	highly available	highly available, but Zookeeper is needed to manage cluster state
Performance	high	very high - higher throughput
Replication	queues are not replicated by design	by design
Protocols	standard queue protocols like AMQP, STOMP, HTTP, and MQTT	binary serialized data
Storage type	queue	log
Acknowledgments	sophisticated	basic
Routing	very flexible (exchange, binding keys)	message is sent to the topic by a key



RabbitMQ AMQP

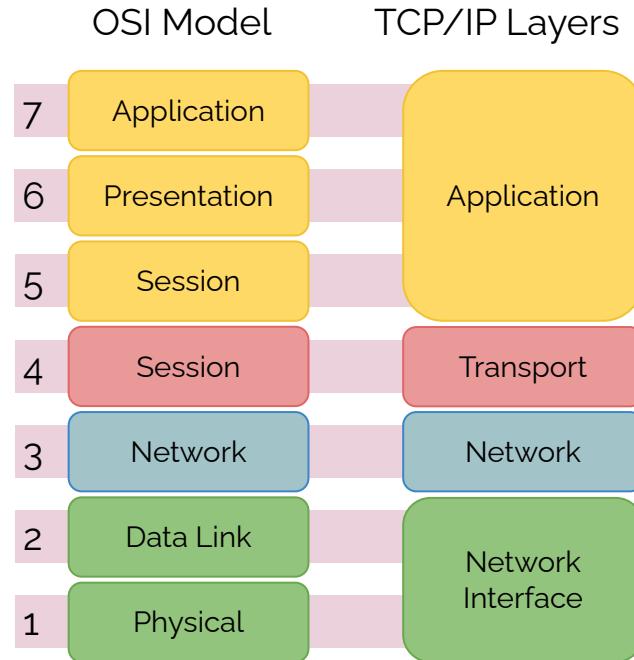
# RabbitMQ deep dive



# AMQP Protocol

AMQP is an open standard application layer protocol.

- Designed for asynchronous communication
- AMQP standardizes the behavior of the messaging publisher and consumer
- Platform independent
- Technology independent (many SDKs)



# AMQP Message

Message is package of information:

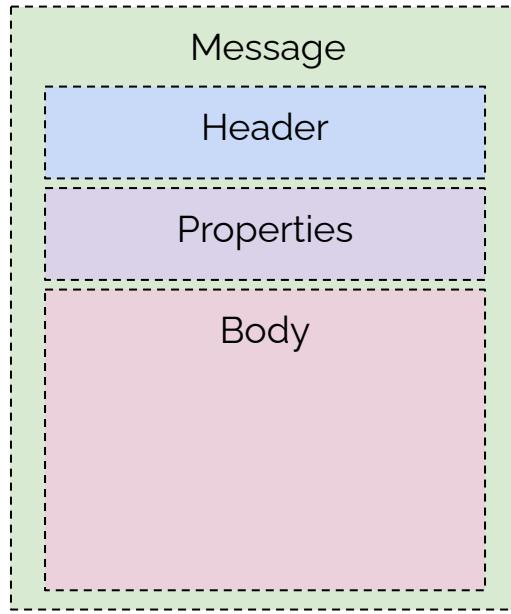
- **Header (key-value pair)**
- **Properties**
- **Body (payload, actual message)**



# AMQP Message

## Properties:

Metadata. Key/Value pairs.  
Application-specific  
information holder



## Header:

Metadata. Key/Value pairs.  
Defined by AMQP specification

## Body:

Payload. byte[].



Message limit is 2GB, but it's better to avoid such a big messages if possible. Message is send per frames; 131KB by default)

# Queuing models

Any process written  
in any language



The Message

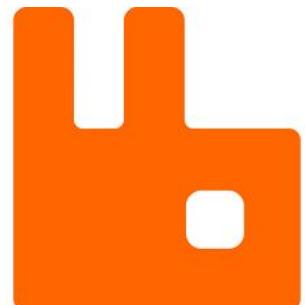
Any process written in  
any language

# Introduction to RabbitMQ

Now we know:

- Basic concepts and terms of queuing systems like RabbitMQ
- We know about the existence of products similar to RabbitMQ
- We know what AMQP protocol is

Let's go to the first Hands On lecture ! :-)





## Installation

# Installation - prerequisites

## Hardware

- Laptop or Virtual machine with Windows 10

## Software

- Text editor, like Notepad++ or similar
- Web browser, like Chrome or similar
- (optionally) IntelliJ to run Java code examples

## Skills

- Using command line tools
- Basic Windows administration knowledge

# Installation

For better understanding let's follow manual installation steps

**1**

Download RabbitMQ

<https://github.com/rabbitmq/rabbitmq-server/releases/download/v3.8.5/rabbitmq-server-windows-3.8.5.zip>

**2**

Download Erlang/OTP

[http://erlang.org/download/otp\\_win64\\_23.0.3.exe](http://erlang.org/download/otp_win64_23.0.3.exe)

**3**

Run the Rabbit!

```
> cd %RABBITMQ%\rabbitmq_server-3.8.5\sbin  
> set ERLANG_HOME=c:\Program Files\erl-23.0.3  
> rabbitmq-server.bat
```

```
## ##      RabbitMQ 3.8.2  
## ##  
##### Copyright (c) 2007-2019 Pivotal Software, Inc.  
##### ##  
##### Licensed under the MPL 1.1. Website: https://rabbitmq.com  
  
Doc guides: https://rabbitmq.com/documentation.html  
Support: https://rabbitmq.com/contact.html  
Tutorials: https://rabbitmq.com/getstarted.html  
Monitoring: https://rabbitmq.com/monitoring.html  
  
Logs: C:/Users/  
      C:/Users/          'RabbitMQ/log/rabbit@U8019795-TPL  
                           'RabbitMQ/log/rabbit@U8019795-TPL  
  
Config file(s): (none)  
  
Starting broker... completed with 3 plugins.
```

# Installation - plugins

RabbitMQ supports plugins. Plugins extend core broker functionality in a many ways

- more protocols
- monitoring
- additional AMQP 0-9-1 exchange types
- node federations
- and many many more

Default RabbitMQ distribution includes lot of plugins disabled by default.

Plugins can be enabled and disabled when node is stopped as well at runtime using command line tools.



# Installation - plugins (management)

Stop the RabbitMQ, then install web management plugin (alternatively open new console, set environment variables and enable plugin using the same command without stopping the node)

```
> cd %RABBITMQ%\rabbitmq_server-3.8.5\sbin  
> set ERLANG_HOME=c:\Program Files\erl-23.0.3  
> rabbitmq-plugins.bat enable rabbitmq_management
```



```
c:\Moje\RabbitMQszkolenie\rabbitmq-server-windows-3.8.5_node1\sbin>rabbitmq-server.bat  
## ##      RabbitMQ 3.8.5  
## ##  
##### Copyright (c) 2007-2020 VMware, Inc. or its affiliates.  
##### #  
##### Licensed under the MPL 1.1. Website: https://rabbitmq.com  
  
Doc guides: https://rabbitmq.com/documentation.html  
Support: https://rabbitmq.com/contact.html  
Tutorials: https://rabbitmq.com/getstarted.html  
Monitoring: https://rabbitmq.com/monitoring.html  
  
Logs: c:/tmp/rabbit1/logs/rabbit1@localhost.log  
      c:/tmp/rabbit1/logs/rabbit1@localhost_upgrade.log  
  
Config file(s): c:\Moje\RabbitMQszkolenie\rabbitmq-server-windows-3.8.5_node1\config\rabbitmq.conf  
  
Starting broker... completed with 3 plugins.
```





Configuration

# Configuration

RabbitMQ provides three general ways to customise the node:

- **environment variables** - define ports, file locations and flags (taken from the shell, or set in the environment configuration file,rabbitmq-env.conf/rabbitmq-env-conf.bat)
- **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

```
cd c:\RabbitMQ\rabbitmq-server-windows-3.8.5_node1\sbin  
set HOMEDRIVE=C:  
set HOMEPATH=\Users%\USERNAME%\br/>set ERLANG_HOME=c:\Program Files\erl-23.0.3  
set RABBITMQ_NODE_PORT=5672  
set RABBITMQ_DIST_PORT=25672  
set RABBITMQ_NODENAME=rabbit1@localhost  
set RABBITMQ_MNESIA_BASE=C:\data\rabbit1  
set RABBITMQ_MNESIA_DIR=C:\data\rabbit1\data  
set RABBITMQ_LOG_BASE=C:\data\rabbit1\logs  
REM Change rabbit1.conf, management.tcp.port = 15672  
REM The Erlang runtime automatically appends the .conf extension to the value of this variable.  
set RABBITMQ_CONFIG_FILE=C:\Users\Me\AppData\Roaming\RabbitMQ\rabbitmq1  
set RABBITMQ_ENABLED_PLUGINS_FILE=C:\data\rabbit1\enabled_plugins  
rabbitmq-server.bat
```



Erlang is a general-purpose programming language and runtime environment

# Configuration

Most important configuration variables:

- **RABBITMQ\_NODE\_PORT** (default 5672) - used by AMQP 0-9-1 and 1.0 clients (without TLS)
- **RABBITMQ\_DIST\_PORT** (default 20000 + RABBITMQ\_NODE\_PORT) - used by Erlang distribution for inter-node and CLI tools communication,
- **RABBITMQ\_NODENAME** (Unix default: rabbit@\$HOSTNAME; Windows default: rabbit@%COMPUTERNAME%) - unique node name
- **RABBITMQ\_MNESIA\_BASE** - RabbitMQ server's node database, message store and cluster state files, one for each node
- **RABBITMQ\_MNESIA\_DIR** - This is **RABBITMQ\_MNESIA\_BASE** subfolder. Includes a schema database, message stores, cluster member information and other persistent node state
- **RABBITMQ\_LOG\_BASE** - just logs
- **RABBITMQ\_CONFIG\_FILE** - points rabbitmq.config file (note the ".config" extension is added)
- **RABBITMQ\_ENABLED\_PLUGINS\_FILE** - file to track enabled plugins

# Configuration

```
# A new style format snippet. This format is used by rabbitmq.conf files.  
ssl_options.cacertfile      = /path/to/ca_certificate.pem  
ssl_options.certfile        = /path/to/server_certificate.pem  
ssl_options.keyfile         = /path/to/server_key.pem  
ssl_options.verify          = verify_peer  
ssl_options.fail_if_no_peer_cert = true
```

```
%% A classic format snippet, now used by advanced.config files.  
[  
  {rabbit, [{ssl_options, [{cacertfile,           "/path/to/ca_certificate.pem"},  
                        {certfile,             "/path/to/server_certificate.pem"},  
                        {keyfile,              "/path/to/server_key.pem"},  
                        {verify,               verify_peer},  
                        {fail_if_no_peer_cert, true}]}]}  
].
```

# Configuration

Most important config file entries

- **cluster\_name** (default "") - used for automatic clustering
- **listeners.tcp.default** - same as **RABBITMQ\_NODE\_PORT**
- **heartbeat** (default 60 seconds) - after 60s connection should be considered unreachable by RabbitMQ and client libraries. Server suggest this value to client libraries while establishing TCP connection at AMQP protocol level. Clients might not follow server's suggestion,
- **frame\_max** - (default 131072) - maximum permissible size of a frame (in bytes) to negotiate with clients (AMQP protocol level). Larger value improves throughput, smaller value improves latency (not related with max message size which is **2GB**)
- **channel\_max** - (default 0 - unlimited) - number of channels to negotiate with clients. Using more channels increases broker's memory footprint,
- **management.tcp.port** (default 15672) - the web-management and RESTful service



RabbitMQ Web Admin



Check the "Backup & Restore" section for more information

# Tab: Overview

RabbitMQ 3.8.2 Erlang 22.2

Refreshed 2020-02-04 18:09:26 Refresh every 5 seconds ▾

Virtual host All ▾

Cluster dev3.eng.megacorp.local

User guest Log out

**Overview** Connections Channels Exchanges Queues Admin

## Overview

▼ Totals

Queued messages last minute ?

Currently idle

Message rates last minute ?

Currently idle

Global counts ?

Connections: 0 Channels: 0 Exchanges: 7 Queues: 0 Consumers: 0

▼ Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	+/-
rabbit1@localhost	0 65536 available	0 58893 available	448 1048576 available	99MiB 6.4GiB high watermark	10GiB 48MiB low watermark	59m 24s	basic disc 1 rss	This node All nodes	
rabbit2@localhost	0 65536 available	0 58893 available	444 1048576 available	95MiB 6.4GiB high watermark	10GiB 48MiB low watermark	58m 38s	basic disc 1 rss	This node All nodes	

► Churn statistics

► Ports and contexts

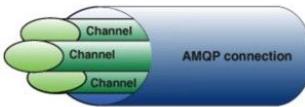
► Export definitions

► Import definitions

---

[HTTP API](#) [Server Docs](#) [Tutorials](#) [Community Support](#) [Community Slack](#) [Commercial Support](#) [Plugins](#) [GitHub](#) [Changelog](#)

# Tab: Connections

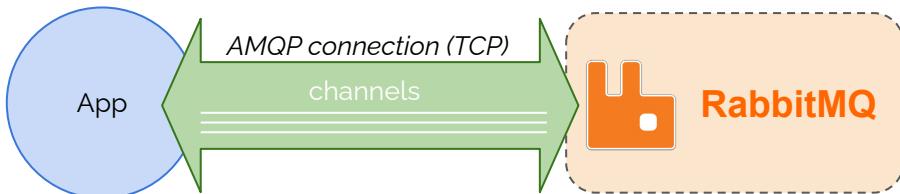


## → Connection

TCP/IP connection between the client and the broker

## → Channel

Virtual connection inside the physical TCP/IP connection (single TCP/IP connection with many channels vs many TCP/IP connections)



## Connections

All connections (26)

Pagination

Page 1 of 1 - Filter:   Regex ?

Overview		Details			Network		
Name	User name	State	SSL / TLS	Protocol	Channels	From client	To client
127.0.0.1:62629	guest	flow	o	AMQP 0-9-1	1	307kiB/s	0iB/s
perf-test-producer-0							
127.0.0.1:62630	guest	flow	o	AMQP 0-9-1	1	320kiB/s	0iB/s
perf-test-producer-1							
127.0.0.1:62632	guest	flow	o	AMQP 0-9-1	1	320kiB/s	0iB/s
perf-test-producer-3							
127.0.0.1:62635	guest	flow	o	AMQP 0-9-1	1	307kiB/s	0iB/s
perf-test-producer-6							
127.0.0.1:62636	guest	flow	o	AMQP 0-9-1	1	346kiB/s	0iB/s
perf-test-producer-7							
127.0.0.1:62637	guest	flow	o	AMQP 0-9-1	1	333kiB/s	0iB/s
perf-test-producer-8							
127.0.0.1:62638	guest	flow	o	AMQP 0-9-1	1	333kiB/s	0iB/s
perf-test-producer-9							
127.0.0.1:62640	guest	flow	o	AMQP 0-9-1	1	320kiB/s	0iB/s
perf-test-producer-11							
127.0.0.1:62641	guest	flow	o	AMQP 0-9-1	1	294kiB/s	0iB/s
perf-test-producer-12							
127.0.0.1:62643	guest	running	o	AMQP 0-9-1	1	346kiB/s	0iB/s
perf-test-producer-14							
127.0.0.1:62644	guest	flow	o	AMQP 0-9-1	1	333kiB/s	0iB/s
perf-test-producer-15							
127.0.0.1:62645	guest	flow	o	AMQP 0-9-1	1	320kiB/s	0iB/s
perf-test-producer-16							
127.0.0.1:62646	guest	flow	o	AMQP 0-9-1	1	294kiB/s	0iB/s
perf-test-producer-17							
127.0.0.1:62647	guest	flow	o	AMQP 0-9-1	1	320kiB/s	0iB/s
perf-test-producer-18							
127.0.0.1:62648	guest	flow	o	AMQP 0-9-1	1	320kiB/s	0iB/s
perf-test-producer-19							
127.0.0.1:62651	guest	flow	o	AMQP 0-9-1	1	294kiB/s	0iB/s
perf-test-producer-22							
127.0.0.1:62652	guest	flow	o	AMQP 0-9-1	1	320kiB/s	0iB/s
perf-test-producer-23							
127.0.0.1:62653	guest	flow	o	AMQP 0-9-1	1	320kiB/s	0iB/s
perf-test-producer-24							
[::1]:62628	guest	running	o	AMQP 0-9-1	1	0iB/s	8.9MiB/s
perf-test-consumer-0							
1..11..62621	object	flow	o	AMQP 0-9-1	1	204kiB/s	0iB/s

# Tab: Connections

- Running
- Flow
- Idle
- Blocking/Blocked

## Connections

▼ All connections (26)

### Pagination

Page **1** of 1 - Filter:   Regex ?

Overview			Details			Network	
Name	User name	State	SSL / TLS	Protocol	Channels	From client	To client
<a href="#">127.0.0.1:62629</a> perf-test-producer-0	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	307kib/s	0ib/s
<a href="#">127.0.0.1:62630</a> perf-test-producer-1	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	320kib/s	0ib/s
<a href="#">127.0.0.1:62632</a> perf-test-producer-3	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	320kib/s	0ib/s
<a href="#">127.0.0.1:62635</a> perf-test-producer-6	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	307kib/s	0ib/s
<a href="#">127.0.0.1:62636</a> perf-test-producer-7	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	346kib/s	0ib/s
<a href="#">127.0.0.1:62637</a> perf-test-producer-8	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	333kib/s	0ib/s
<a href="#">127.0.0.1:62638</a> perf-test-producer-9	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	333kib/s	0ib/s
<a href="#">127.0.0.1:62640</a> perf-test-producer-11	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	320kib/s	0ib/s
<a href="#">127.0.0.1:62641</a> perf-test-producer-12	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	294kib/s	0ib/s
<a href="#">127.0.0.1:62643</a> perf-test-producer-14	guest	<span style="background-color: #6aa84f; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> running	○	AMQP 0-9-1	1	346kib/s	0ib/s
<a href="#">127.0.0.1:62644</a> perf-test-producer-15	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	333kib/s	0ib/s
<a href="#">127.0.0.1:62645</a> perf-test-producer-16	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	320kib/s	0ib/s
<a href="#">127.0.0.1:62646</a> perf-test-producer-17	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	294kib/s	0ib/s
<a href="#">127.0.0.1:62647</a> perf-test-producer-18	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	320kib/s	0ib/s
<a href="#">127.0.0.1:62648</a> perf-test-producer-19	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	320kib/s	0ib/s
<a href="#">127.0.0.1:62651</a> perf-test-producer-22	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	294kib/s	0ib/s
<a href="#">127.0.0.1:62652</a> perf-test-producer-23	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	320kib/s	0ib/s
<a href="#">127.0.0.1:62653</a> perf-test-producer-24	guest	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	320kib/s	0ib/s
<a href="#">[::1]:62628</a> perf-test-consumer-0	guest	<span style="background-color: #6aa84f; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> running	○	AMQP 0-9-1	1	0ib/s	8.9MiB/s
<a href="#">1..1..62621</a>	object	<span style="background-color: #ffff00; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span> flow	○	AMQP 0-9-1	1	204kib/s	0ib/s

# Tab: Channels

RabbitMQ™ RabbitMQ 3.8.5 Erlang 23.0.3

Refreshed 2020-07-30 22:47:28 Refresh every 5 seconds

Virtual host All Cluster rabbit1@U8019795-TPL-Q.ten.thomsonreuters.com User guest Log out

Overview Connections **Channels** Exchanges Queues Admin

All channels (26)

Pagination

Page 1 of 1 - Filter:   Regex ? Displaying 26 items, page size up to: 100

Overview		Details			Message rates						+/-	
Channel	User name	Mode	State	Unconfirmed	Prefetch	Unacked	publish	confirm	unroutable (drop)	deliver / get	ack	
127.0.0.1:62629 (1)	guest	yellow	flow	0	0	0	3,200/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62630 (1)	guest	yellow	flow	0	0	0	3,120/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62632 (1)	guest	yellow	flow	0	0	0	3,273/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62635 (1)	guest	yellow	flow	0	0	0	3,280/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62636 (1)	guest	yellow	flow	0	0	0	3,120/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62637 (1)	guest	yellow	flow	0	0	0	3,160/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62638 (1)	guest	yellow	flow	0	0	0	3,080/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62640 (1)	guest	yellow	flow	0	0	0	3,000/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62641 (1)	guest	yellow	flow	0	0	0	3,200/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62643 (1)	guest	yellow	flow	0	0	0	3,000/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62644 (1)	guest	yellow	flow	0	0	0	3,120/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62645 (1)	guest	yellow	flow	0	0	0	2,960/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62646 (1)	guest	yellow	flow	0	0	0	3,156/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62647 (1)	guest	yellow	flow	0	0	0	3,074/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62648 (1)	guest	yellow	flow	0	0	0	3,160/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62651 (1)	guest	yellow	flow	0	0	0	3,200/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62652 (1)	guest	yellow	flow	0	0	0	3,120/s	0.00/s	0.00/s	0.00/s	0.00/s	
127.0.0.1:62653 (1)	guest	yellow	flow	0	0	0	3,280/s	0.00/s	0.00/s	0.00/s	0.00/s	
[::1]:62628 (1)	guest	green	running	0	0	0	77,890/s	0.00/s	0.00/s	0.00/s	0.00/s	
[::1]:62631 (1)	guest	yellow	flow	0	0	0	3,240/s	0.00/s	0.00/s	0.00/s	0.00/s	
[::1]:62633 (1)	guest	yellow	flow	0	0	0	3,280/s	0.00/s	0.00/s	0.00/s	0.00/s	
[::1]:62634 (1)	guest	yellow	flow	0	0	0	3,240/s	0.00/s	0.00/s	0.00/s	0.00/s	
[::1]:62639 (1)	guest	yellow	flow	0	0	0	3,120/s	0.00/s	0.00/s	0.00/s	0.00/s	
[::1]:62642 (1)	guest	yellow	flow	0	0	0	3,240/s	0.00/s	0.00/s	0.00/s	0.00/s	
[::1]:62649 (1)	guest	yellow	flow	0	0	0	3,200/s	0.00/s	0.00/s	0.00/s	0.00/s	
[::1]:62650 (1)	guest	yellow	flow	0	0	0	3,040/s	0.00/s	0.00/s	0.00/s	0.00/s	

# Tab: Exchanges

RabbitMQ™ RabbitMQ 3.8.5 Erlang 23.0.3 Refreshed 2020-07-30 22:49:06 Refresh every 5 seconds Virtual host All Cluster rabbit1@U8019795-TPL-Q.ten.thomsonreuters.com User guest Log out

Overview Connections Channels Exchanges Queues Admin

## Exchanges

All exchanges (8)

Pagination

Page 1 of 1 - Filter:   Regex ? Displaying 8 items , page size up to: 100

Name	Type	Features	Message rate in	Message rate out
(AMQP default)	direct	D		
amq.direct	direct	D		
amq.fanout	fanout	D		
amq.headers	headers	D		
amq.match	headers	D		
amq.rabbitmq.trace	topic	D I		
amq.topic	topic	D		
direct	direct		0.00/s	66,484/s

Add a new exchange

Name:  \*

Type: direct

Durability: Durable

Auto delete: No

Internal: No

Arguments:  =  String

Add Alternate exchange ?

Add exchange

# Tab: Queues

RabbitMQ™ RabbitMQ 3.8.5 Erlang 23.0.3 Refreshed 2020-07-30 22:50:13 Refresh every 5 seconds Virtual host All Cluster rabbit1@U8019795-TPL-Q.ten.thomsonreuters.com User guest Log out

Overview Connections Channels Exchanges **Queues** Admin

Queues ▾ All queues (2013)

Pagination Page 1 of 21 - Filter:   Regex ? Displaying 100 items , page size up to: 100

Name	Type	Features	State	Messages			Message rates		
				Ready	Unacked	Total	incoming	deliver / get	ack
perf_queue-name	classic	D AD	running	0	0	0	146,508/s	80,081/s	0.00/s
not-lazy-queue	classic	D Args	idle	0	0	0			
lazy-queue	classic	D Args	policy-4-lazy-queues	idle	0	0			
'perf-test-9'	classic	D AD	idle	0	0	0			
'perf-test-8'	classic	D AD	idle	0	0	0			
'perf-test-7'	classic	D AD	idle	0	0	0			
'perf-test-6'	classic	D AD	idle	0	0	0			
'perf-test-5'	classic	D AD	idle	0	0	0			
'perf-test-4'	classic	D AD	idle	0	0	0			
'perf-test-3'	classic	D AD	idle	0	0	0			
'perf-test-2'	classic	D AD	idle	0	0	0			
'perf-test-10'	classic	D AD	idle	0	0	0			
'perf-test-1'	classic	D AD	idle	0	0	0			
'perf-test-02000'	classic	D AD	idle	0	0	0			
'perf-test-01999'	classic	D AD	idle	0	0	0			
'perf-test-01998'	classic	D AD	idle	0	0	0			
'perf-test-01997'	classic	D AD	idle	0	0	0			
'perf-test-01996'	classic	D AD	idle	0	0	0			
'perf-test-01995'	classic	D AD	idle	0	0	0			
'perf-test-01994'	classic	D AD	idle	0	0	0			
'perf-test-01993'	classic	D AD	idle	0	0	0			
'perf-test-01992'	classic	D AD	idle	0	0	0			

# Tab: Admin

RabbitMQ™ RabbitMQ 3.8.5 Erlang 23.0.3 Refreshed 2020-07-30 22:51:01 Refresh every 5 seconds ▾ Virtual host All Cluster rabbit1@U8019795-TPL-Q.ten.thomsonreuters.com User guest Log out

Overview Connections Channels Exchanges Queues Admin

Users

All users

Filter:   Regex ? 1 item, page size up to 100 Feature Flags

Name	Tags	Can access virtual hosts	Has password
guest	administrator	/	*

? Virtual Hosts 

Add a user

Username:  \*  
Password:  \*  \* (confirm)  
Tags:   
Set Admin | Monitoring | Policymaker  
Management | Impersonator | None  
?

Add user

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

# RabbitMQ web admin

Let's assume scenario:

- You don't like web management UI
- You need to integrate external systems with RabbitMQ
- You need to create your own monitoring UI for RabbitMQ

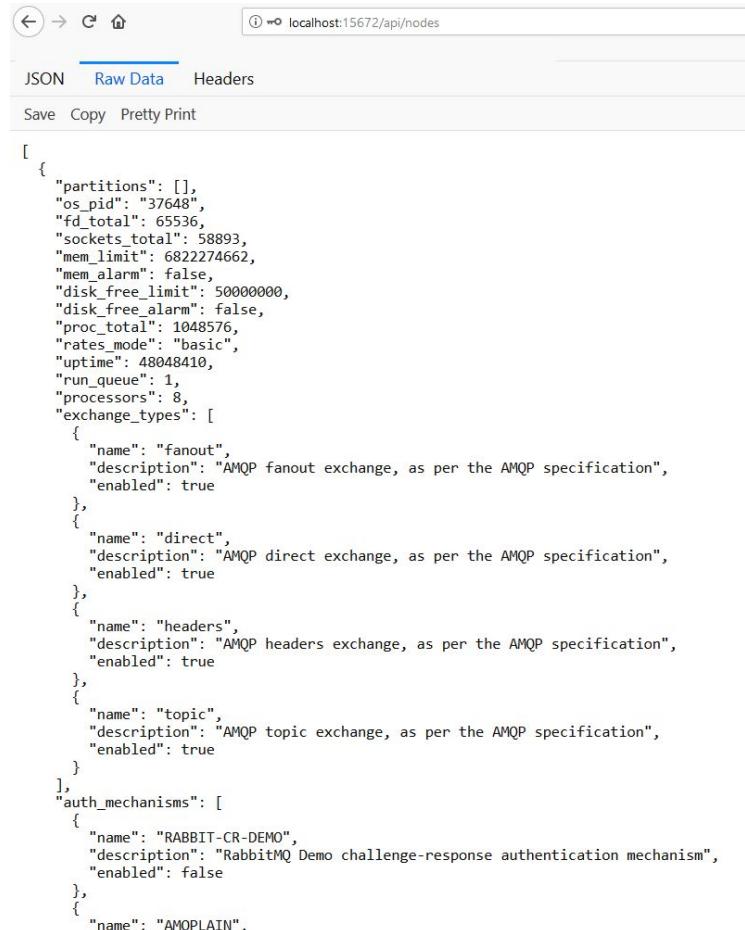
Use RabbitMQ RESTful API instead!

# RabbitMQ web admin

Web Admin is coming with RESTful service

<http://localhost:15672/api/nodes>

```
> cd %RABBITMQ%\rabbitmq_server-3.8.5\sbin  
> set ERLANG_HOME=c:\Program Files\erl-23.0.3  
> rabbitmq-plugins.bat enable rabbitmq_management
```



```
[  
  {  
    "partitions": [],  
    "os_pid": "37648",  
    "fd_total": 65536,  
    "sockets_total": 58893,  
    "mem_limit": 6822274662,  
    "mem_alarm": false,  
    "disk_free_limit": 50000000,  
    "disk_free_alarm": false,  
    "proc_total": 1048576,  
    "rates_mode": "basic",  
    "uptime": 48048410,  
    "run_queue": 1,  
    "processors": 8,  
    "exchange_types": [  
      {  
        "name": "fanout",  
        "description": "AMQP fanout exchange, as per the AMQP specification",  
        "enabled": true  
      },  
      {  
        "name": "direct",  
        "description": "AMQP direct exchange, as per the AMQP specification",  
        "enabled": true  
      },  
      {  
        "name": "headers",  
        "description": "AMQP headers exchange, as per the AMQP specification",  
        "enabled": true  
      },  
      {  
        "name": "topic",  
        "description": "AMQP topic exchange, as per the AMQP specification",  
        "enabled": true  
      }  
    ],  
    "auth_mechanisms": [  
      {  
        "name": "RABBIT-CR-DEMO",  
        "description": "RabbitMQ Demo challenge-response authentication mechanism",  
        "enabled": false  
      },  
      {  
        "name": "AMQPLAIN",  
        "description": "AMQP plain text authentication mechanism",  
        "enabled": true  
      }  
    ]  
  }  
]
```

# RabbitMQ ports

Make sure the following ports are open:

**4369**: epmd, a peer discovery service used by RabbitMQ nodes and CLI tools

**5672, 5671**: used by AMQP 0-9-1 and 1.0 clients without and with TLS

**25672**: used by Erlang distribution for inter-node and CLI tools communication and is allocated from a dynamic range (limited to a single port by default, computed as AMQP port + 20000).

**15672**: HTTP API clients and rabbitmqadmin (only if the management plugin is enabled)

Other protocols:

**61613, 61614**: STOMP clients without and with TLS (only if the STOMP plugin is enabled)

**1883, 8883**: MQTT clients without and with TLS, (only if the MQTT plugin is enabled)

**15674**: STOMP-over-WebSockets clients (only if the Web STOMP plugin is enabled)

**15675**: MQTT-over-WebSockets clients (only if the Web MQTT plugin is enabled)



Core concepts

# Core concepts

Any process written  
in any language



The Message

Any process written in  
any language

# Core concepts

- **Producer** emits messages to exchange
- **Consumer** receives messages from the queue
- **Queue** keeps/stores messages
- **Exchange**
  - ◆ **Binding** connects an exchange with a queue using **binding key**
  - ◆ Exchange compares **routing key** with binding key (on-to-one or using pattern)

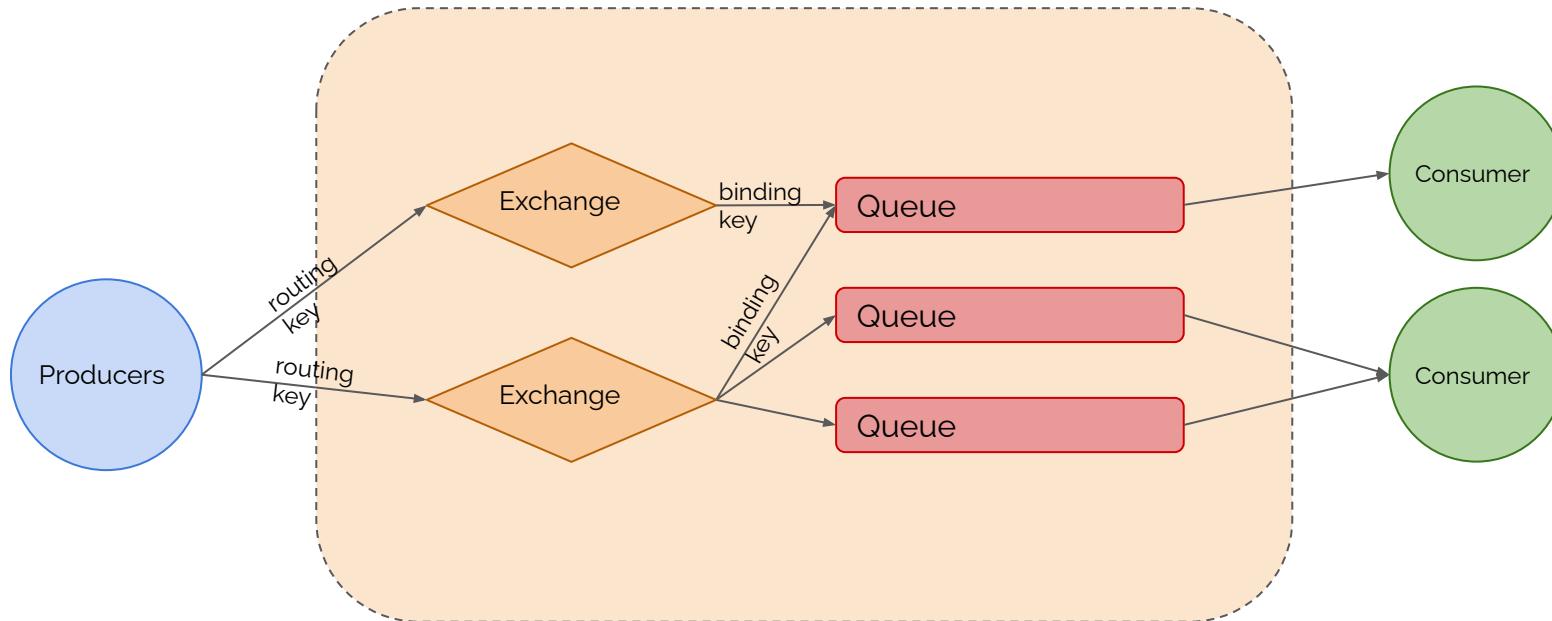


From the day one take care about naming conventions!



Producer never send messages directly to the queue

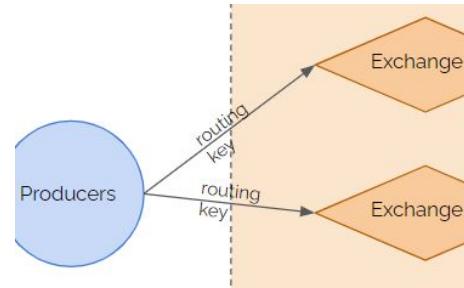
# RabbitMQ



**RabbitMQ**  
broker

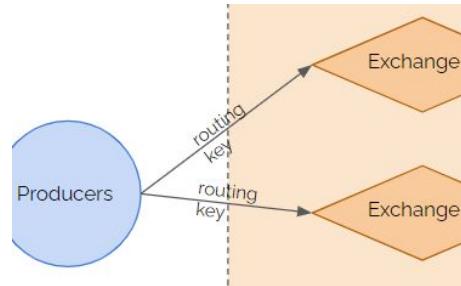
# Exchange

- Exchange type determines distribution model
- **Exchange types: nameless (empty string - default one), fanout, direct, topic , headers**
- Nameless exchange (aka “default exchange”, aka “AMQP default”)
  - ◆ Special one created by RabbitMQ
  - ◆ Compares routing key with queue name
  - ◆ Allows sending messages “directly” to the queue (from the publisher perspective exchange is transparent)



```
> rabbitmqctl list_exchanges
```

# Exchange



## → **fanout**

Simply routes a received message to all queues that are bound to it. Ignores routing key

## → **direct**

Routes a received message to the queue that match "rounting-key". Nameless exchange is a "direct" exchange.

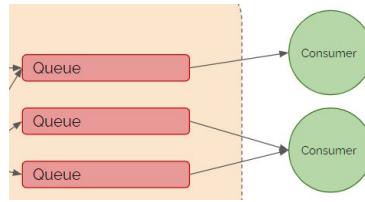
## → **topic**

Routes a received message to queues, where binding key (defined as a pattern) matches to the routing key. Example, binding-key: ".logs.error", routing-key: "application1.logs.error"

## → **headers**

Same as topic, but binding-key is compared against "any" or "all" message headers (header x-match determines the behavior)

# Queue concept



For resilience and availability, queue can be **Federated or Mirrored**.

Queue max size is set by policies

- Located on single node where it was declared and referenced by unique name

In contrary to Exchanges and Bindings, which exist on all nodes. Name can be provided or auto-generated by RabbitMQ. Both: Producer and Consumer **can create a queue**.

- Performance

1 queue = 1 Erlang process

When node starts, up to **16384** messages are loaded into RAM from the queue

- Queue is ordered collection of messages

Messages are published and consumed in the FIFO manner (except prioritized queues)

- Queue has many properties

Depends form the use case we can set queue behavior

- Queue can be federated or mirrored

To increase reliability and availability

- Internal queues amq.

Queues prefixed by "amq." are used for RabbitMQ internal purposes only

```
> rabbitmq-diagnostics memory_breakdown  
> rabbitmqctl status  
> rabbitmqctl list_queues
```

# Queue properties

Among others RabbitMQ implements AMQP protocol. So, attributes (properties) and queue behavior can be changed in many ways to support generic RabbitMQ architecture.

## → By common Queue definition

Name, Durable, Auto-Delete, Exclusive

```
channel.queueDeclare(QUEUE_NAME, /*durable*/true, /*exclusive*/false, /*autoDelete*/false, /*arguments*/args);
```

## → By protocol specific settings

Priority etc.

```
Map<String, Object> args = new HashMap<String, Object>();  
args.put("x-max-priority", 10);  
channel.queueDeclare(..... /*arguments*/args);
```

## → By policies

TTL, Federation etc. (web management, REST API, command line tool)

```
> rabbitmqctl set_policy TTL "." "{\"message-ttl\":60000} --apply-to queues
```

# Queue properties (most popular)

## → **Name**

Name can be provided or auto-generated by RabbitMQ



There are more attributes to set-up queue behavior.

## → **Durable or not**

Not durable queues won't survive broker restart

## → **Auto Delete feature**

Queue deletes itself when all consumers disconnect

## → **Classic or Quorum**

Quorum and Mirroring (policy) increases availability

## → **Exclusive**

Used by only one connection and the queue will be deleted when that connection closes

## → **Priority**

Additional CPU cost and increased latency; no guarantee of exact order (just strong suggestion)

## → **Expiration time (TTL)**

Both messages (expiration property) and queues (x-message-ttl) can have TTL (policy settings); minimum value from both is used

## → **Lazy Queues, Dead Letter Queues and many more....**

Overview				
Name	Node	Type	Features	State
test-ha2	rabbit1@localhost +1	classic	D Args policy-test-ha	idle
test-ha3	rabbit1@localhost +1	classic	D AD Args	

# Queue concept - order

Queues are FIFO manner - **in terms of producer** (messages are always held in the queue in publication order), but **not in terms of consumer**.

Section 4.7 of the AMQP 0-9-1 core specification explains the conditions under which consuming order is guaranteed (received in the same order that they were sent):

- messages published in one channel,
- passing through one exchange,
- stored in one queue
- consumed by exactly one outgoing channel



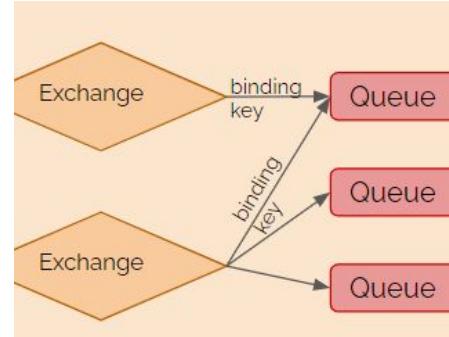
For example, prioritized queue or queue where consumer rejects with requeue=true.

# Binding

- **Connects Exchange with Queue**  
Using binding key (aka. Routing Key)

- **Decide about message processing**  
Defines whether message posted to an exchange should be send to the queue or not

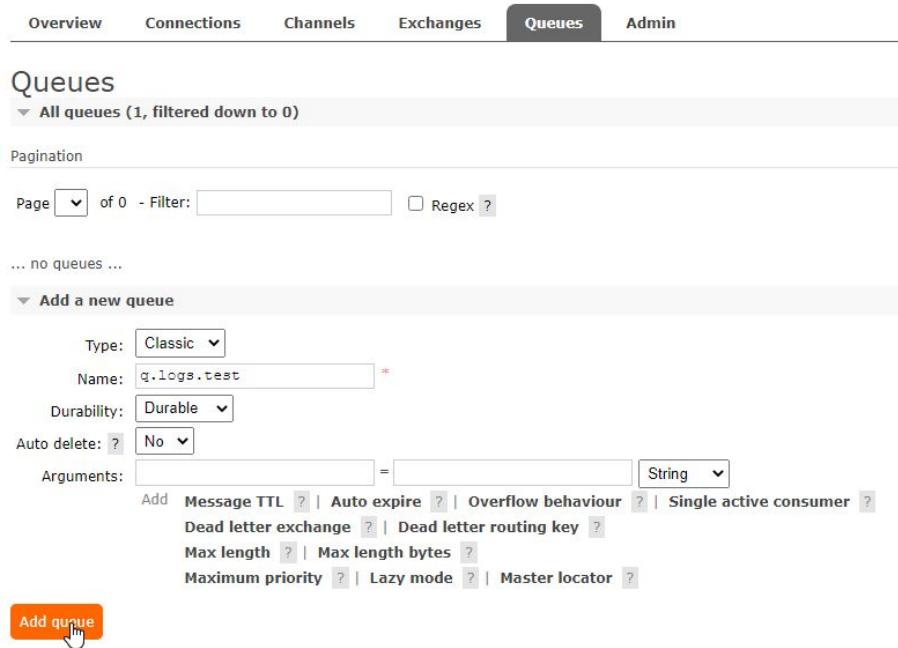
- **Routing key behaviour depends from Exchange type**  
fanout - just ignores routing key,  
topic - routing key has to be valid topic separated by dots,  
direct - exact string match



```
> rabbitmqctl list_bindings
```

# First Queue Hands-On

Create first queue from the web management UI.



The screenshot shows the RabbitMQ Web Management UI with the 'Queues' tab selected. The page title is 'Queues' and there is a dropdown menu showing 'All queues (1, filtered down to 0)'. Below this is a 'Pagination' section with a dropdown for 'Page' set to '1' of 0, a 'Filter' input field, and a 'Regex?' checkbox. A message '... no queues ...' is displayed. At the bottom, there is a form for 'Add a new queue' with fields for 'Type' (set to 'Classic'), 'Name' (set to 'q.logs.test'), 'Durability' (set to 'Durable'), 'Auto delete' (set to 'No'), and an 'Arguments' section. The 'Arguments' section includes options like 'Message TTL', 'Auto expire', 'Overflow behaviour', 'Single active consumer', 'Dead letter exchange', 'Dead letter routing key', 'Max length', 'Max length bytes', 'Maximum priority', 'Lazy mode', and 'Master locator'. A large orange button labeled 'Add queue' is at the bottom left of the form.

EXERCISE

# First Queue Hands-On

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
my_first_queue	classic	D Args	idle	0	0	0				

→ **Name**

Unique queue name in entire RabbitMQ cluster



Naming convention!

→ **Type**

Classic - not replicated queue

→ **Durability**

Durable - queue will survive RabbitMQ restart

→ **Auto-Delete**

Yes - queue will be deleted after at least one consumer has connected, and then all consumers have disconnected.



# First Message

Create first message and send it into queue from the web management UI.

The screenshot shows the RabbitMQ Management UI interface. The top navigation bar includes tabs for Overview, Connections, Channels, Exchanges, Queues (which is selected), and Admin. Below the navigation is a timeline showing recent activity: 14:33:00, 14:33:10, 14:33:20, 14:33:30, 14:33:40, and 14:33:50. A message rates section displays 'last minute' data, indicating the system is currently idle. The main area is titled 'Details' for the queue 'my\_first\_queue'. It shows the following configuration:

Features	arguments: x-queue-type: classic durable: true	State	idle	Consumers	0	Messages	0	Total	0	Ready	0	Unacked	0	In memory	0	Persistent	0	Transient	0	Paged Out	0
Policy				Consumer utilisation	0%	Message body bytes	0B	0B	0B	0B	0B	0B	Process memory	11kB	0B	0B	0B	0B	0B	0B	

On the left, there are sections for Consumers, Bindings, and Publish message. The Publish message section contains the following fields:

- Delivery mode: 1 - Non-persistent
- Headers:  =  String
- Properties:  =
- Payload:

A red arrow icon with the word "EXERCISE" is overlaid on the bottom left of the screenshot.

# First Message

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
my_first_queue	classic	D Args	idle	1	0	1	0.00/s			

## → Delivery mode

Persistent - message will survive RabbitMQ restart

## → Headers

AMQP level settings

## → Properties

RabbitMQ level settings



# First consumption

Overview    Connections    Channels    Exchanges    **Queues**    Admin

## Queue my\_first\_queue

▶ Overview

▶ Consumers

▶ Bindings

▶ Publish message

▼ Get messages

Warning: getting messages from a queue is a destructive action. [?](#)

Ack Mode:

Encoding:  [?](#)

Messages:

**Get Message(s)** 

▶ Move messages

▶ Delete

▶ Purge

▶ Runtime Metrics (Advanced)

[HTTP API](#)    [Server Docs](#)    [Tutorials](#)    [Community Support](#)    [Community Slack](#)    [Commercial Support](#)    [Plugins](#)    [GitHub](#)    [Changelog](#)



**EXERCISE** 

# First consumption

## → Ack mode

Reject, Nack(negative ack - reject extension), Ack; requeue  
refers to Dead Letter Exchange (DLX)

## → Encoding

How to handle binary payload

## → Messages

How many messages to be read

### ▼ Get messages

Warning: getting messages from a queue is a destructive action. [?](#)

Ack Mode:  [?](#)

Encoding:  [?](#)

Messages:





Requested message is placed to its original position **if possible**. If not message will be requeued closer to the queue head (concurrent deliveries and acknowledgements from consumers).

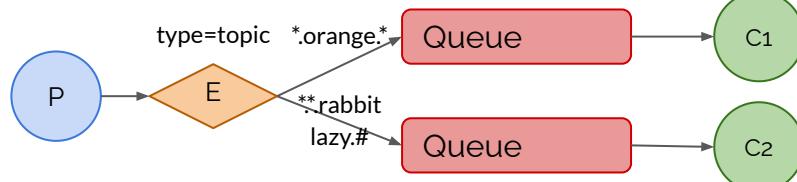
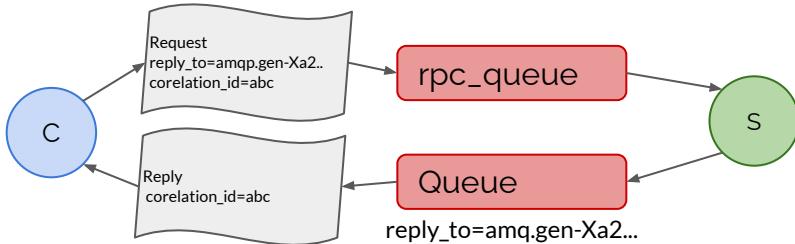
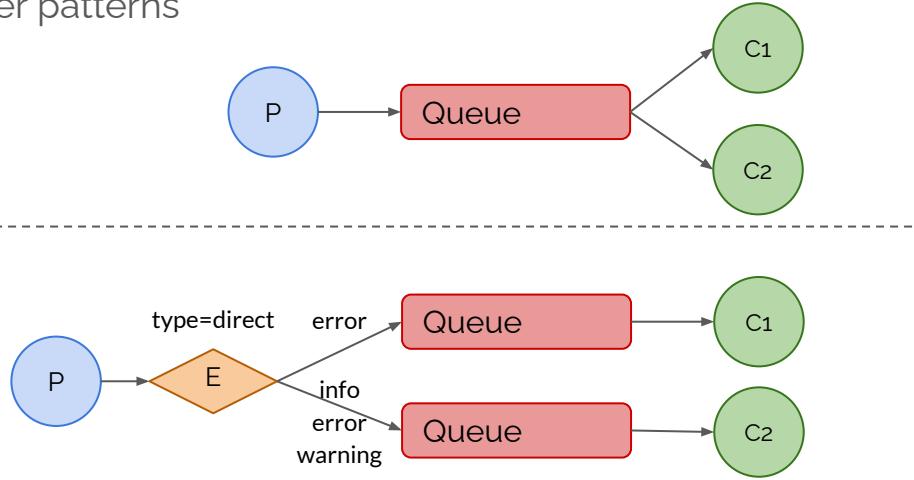
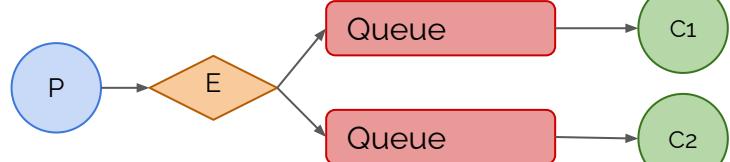
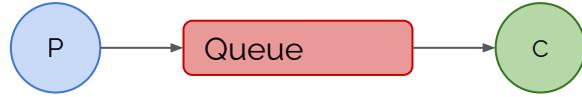




Workshop - sample use cases

# RabbitMQ common scenarios

RabbitMQ allows to implement many consumer patterns

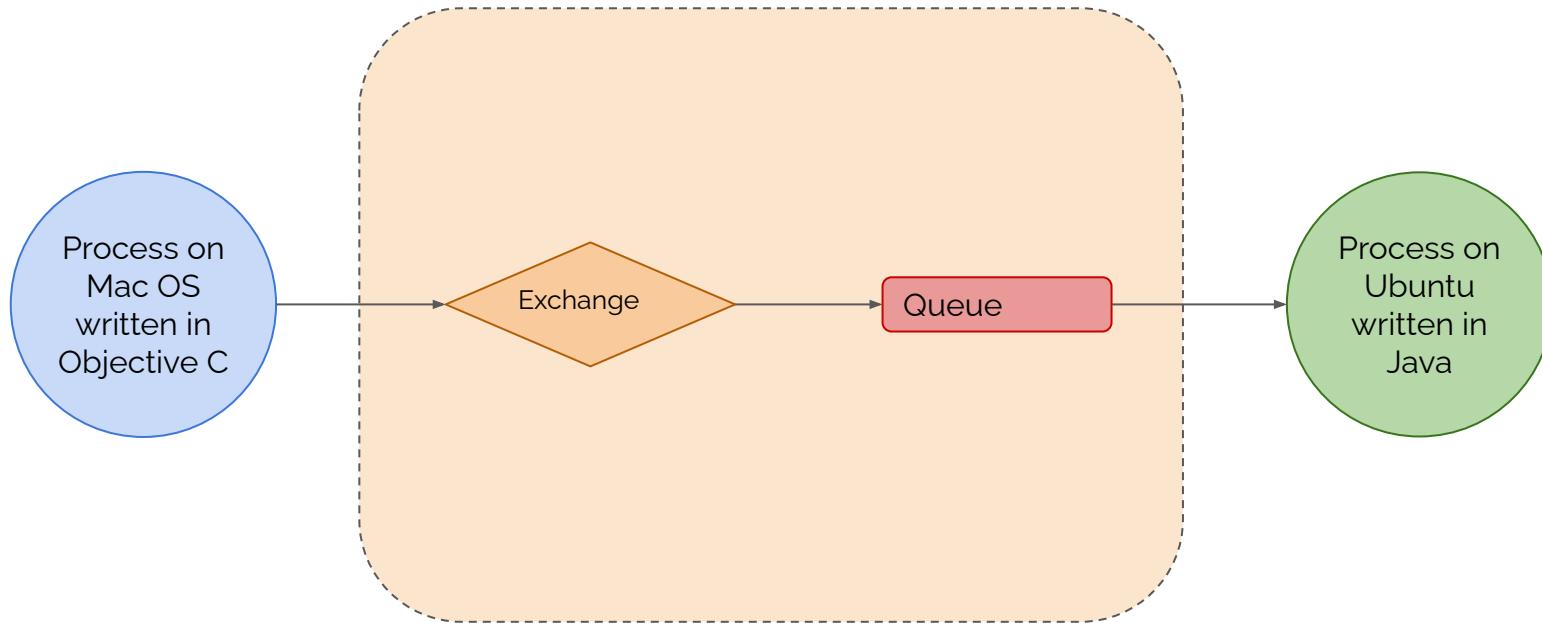




Simple Queue

Sample usecases:

- Easy code split without multithreading
- Connect two processes written in different technologies



RabbitMQ



# Simple Queue

▼ Add a new queue

Type: Classic

Name: helloqueue \*

Durability: Transient

Auto delete: ? No

Arguments:  =

Add Message TTL  | Auto ex<sup>?</sup> | Dead letter exchange  | Lazy mode  Master loca

▼ Publish message

Message will be published to the default exchange with routing key: test

Delivery mode: 1 - Non-persistent

Headers:  =

Properties:  =

Payload: test

We publish to the queue directly  
(indirectly via empty exchange "")

helloqueue	classic	Args	idle	1	0	1	0.00/s
------------	---------	------	------	---	---	---	--------

EXERCISE 

# Publisher/Producer



HelloExample.java

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);

String message = "Hello World";
channel.basicPublish(/*exchange*/"", /*routingKey*/QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));

//---- Alternative with TTL ---
Map<String, Object> args = new HashMap<String, Object>();
args.put("x-message-ttl", 60000); // 60s
channel.queueDeclare(QUEUE_NAME, false, false, false, args);

//---- Alternative with TTL ---
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
    .expiration("60000")
    .build();
channel.basicPublish(/*exchange*/"", /*routingKey*/QUEUE_NAME, properties, message.getBytes(StandardCharsets.UTF_8));
```

# Consumer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);
System.out.println(" [*] Waiting for messages....");

DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Received '" + message + "'");
};

channel.basicConsume(QUEUE_NAME, /*autoAck*/true, deliverCallback, consumerTag -> { });
```

# Queues - Persistence and Durability

## → **Durability**

AMQP property for queues and exchanges. Messages in durable entities **can survive** server restarts, by being automatically recreated when server gets up.

## → **Persistence**

Messages property. Stored on disk in special persistency log file, allowing them to **be restored** once server gets up. Persistence has no effect a non-durable queues.

Persistent messages are removed from a durable queue once they are consumed (and acknowledged).

# Queues - Persistence and Durability

As default messages won't survive RabbitMQ restart or entire server restarts. To ensure messages survive, make sure to:

- Send message as **persistent message**

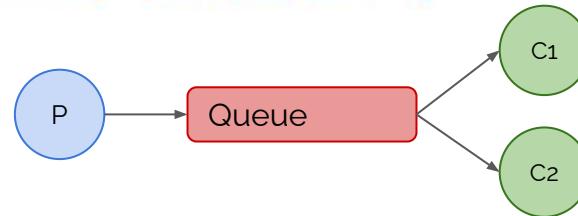
```
channel.basicPublish("", WORK_QUEUE_NAME,  
    /*props*/ MessageProperties.PERSISTENT_TEXT_PLAIN,  
    message.getBytes("UTF-8"));
```

- Publish into **durable exchange**

```
channel.exchangeDeclare(EXCHANGE_NAME, "topic", /*durable*/true);
```

- Message to be stored in **durable queue**

```
channel.queueDeclare(QUEUE_NAME, /*durable*/true, /*exclusive*/false, /*autoDelete*/false,  
    /*arguments*/null);
```



## Work Queues / Task Queues

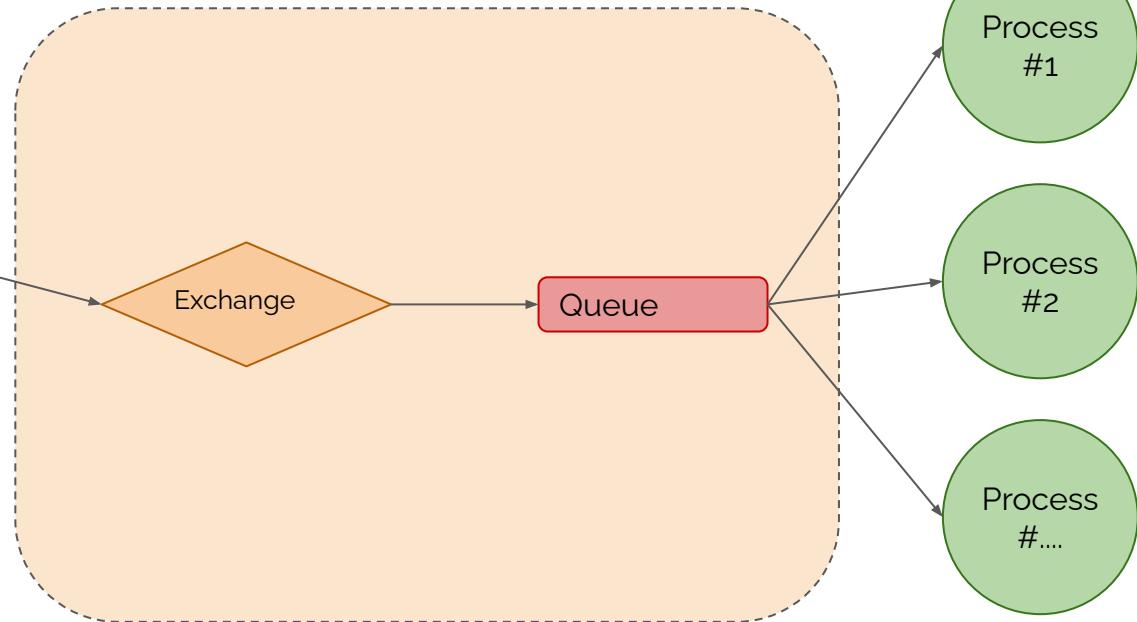
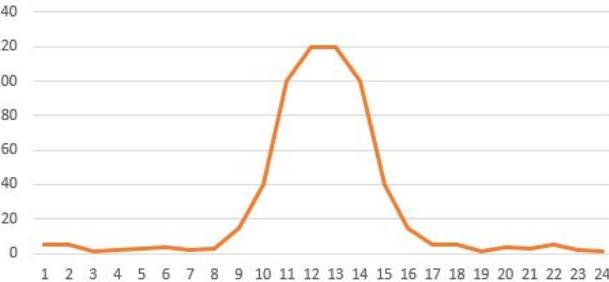
Sample usecases:

- ➔ Distribute time-consuming tasks among multiple clients (workers)
- ➔ Introduce asynchronous HTTP calls to RESTful services



events /  
messages

Daily load



 RabbitMQ

The RabbitMQ logo consists of a stylized orange icon followed by the word "RabbitMQ" in a grey sans-serif font.

  
EXERCISE

A red arrow pointing to the right, with the word "EXERCISE" written in white capital letters across it.

# Publisher / Producer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);

String message = "Hello World";
channel.basicPublish(/*exchange*/"", /*routingKey*/QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));
```



Can be the same as in first example  
(WorkQueueExample.java)

# Consumer / Worker



Note **positive** and **negative** acknowledgments.  
Nack is a Reject extension (Reject has no  
"multiple" field - batch acknowledgment for all  
past unacknowledged messages when = true)

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);
System.out.println(" [*] Waiting for messages....");

channel.basicQos(/*prefetchCount; basic.qos*/1);

DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Received '" + message + "'");

    channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);

    // channel.basicNack(delivery.getEnvelope().getDeliveryTag(), false, /*requeue*/true);
    // channel.basicReject(delivery.getEnvelope().getDeliveryTag(), true);
};

channel.basicConsume(QUEUE_NAME, /*autoAck*/false, deliverCallback, consumerTag -> { });
```

# Simple Queue

Exchange: (AMQP default)

▶ Overview

▶ Bindings

▼ Publish message

Routing key: helloqueue

Headers: ? =

Properties: ? =

Payload: test

**Publish message**

We publish to the queue directly via empty exchange ""



helloqueue	classic	Args	idle	2	0	2	0.00/s
------------	---------	------	------	---	---	---	--------

# Work Queues - Summary

- **Round-robin dispatching**

Sends each message to the next consumer in a sequence

- **Message acknowledgment**

As default, message delivered into consumer is marked for deletion

- **No message timeouts**

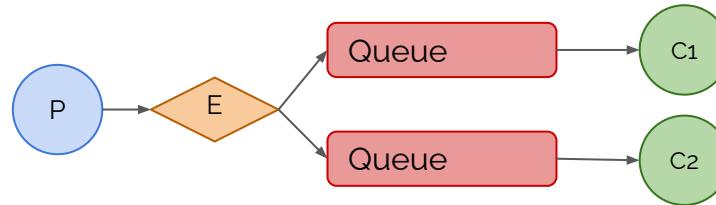
RabbitMQ redeliver message when consumer dies

- **Prefetch**

RabbitMQ dispatches a message when it enters the queue, here we want consumer to handle one message at time



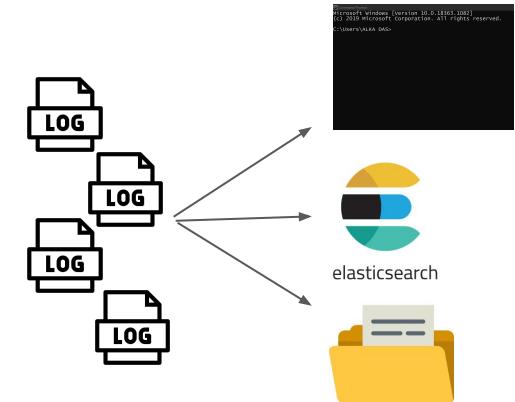
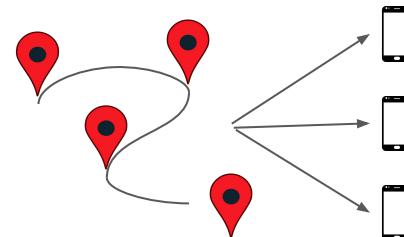
Monitor queue size! If all workers are busy, queue can easily fill-up.

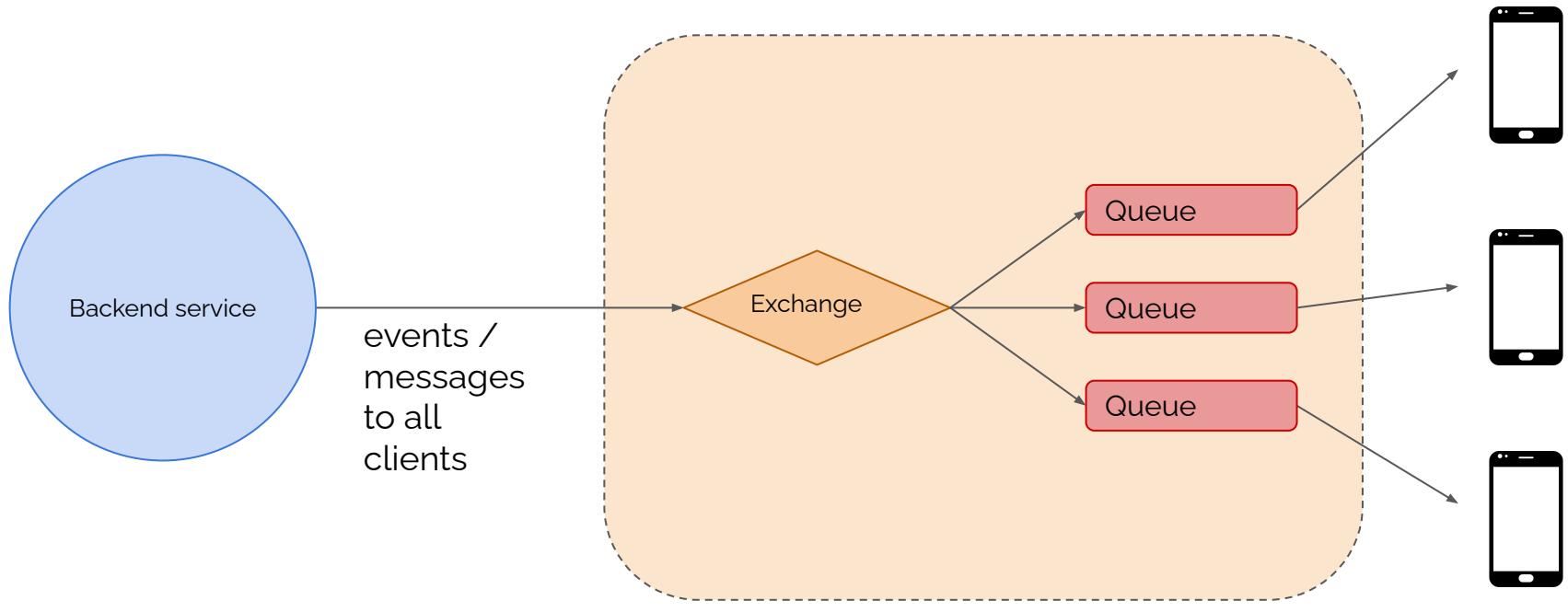


## Publish / Subscribe (fanout)

Sample usecases:

- Notifications
- Feeds





TTL or MAX queue size  
to be considered

 **RabbitMQ**

**EXERCISE**

# Producer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);

String message = "Hello World";
channel.basicPublish(/*exchange*/EXCHANGE_NAME, /*routingKey*/"", null, message.getBytes(StandardCharsets.UTF_8));
```



Exchange type can be one of: fanout, direct, topic , headers  
(PublishSubscribeExample.java)

# Consumer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

String queueName = channel.queueDeclare().getQueue();
channel.queueBind(queueName, EXCHANGE_NAME, /*routingKey*/"");

System.out.println(" [*] Waiting for messages....");

DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Received '" + message + "'");
};

channel.basicConsume(QUEUE_NAME, /*autoAck*/true, deliverCallback, consumerTag -> { });
```

# Publish/Subscribe

Bind both queues  
with "logs" exchange  
with empty routing  
key

Add a new exchange

Name: logs \*Type: fanout  
Durability: Durable  
Auto delete: No  
Internal: No  
Arguments: [ ]

Add Alternate exchange ?

Add exchange

Exchange: logs

Overview

Bindings

Publish message

Routing key: [ ]

Headers: [ ]

Properties: [ ]

Payload: test

Add a new queue

Type: Classic \*Name: logs\_queue\_1  
Durability: Durable  
Auto delete: No  
Arguments: [ ]

Add Message TTL ? | Auto expire  
Dead letter exchange ? | Dead letter mode ?  
Lazy mode ? | Master locator ?

Add queue

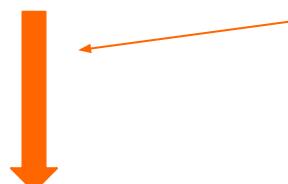
Add a new queue

Type: Classic \*Name: logs\_queue\_2  
Durability: Durable  
Auto delete: No  
Arguments: [ ]

Add Message TTL ? | Auto expire  
Dead letter exchange ? | Dead letter mode ?  
Lazy mode ? | Master locator ?

Add queue

Bind both queues  
with "logs" exchange  
with empty routing  
key



Add queue

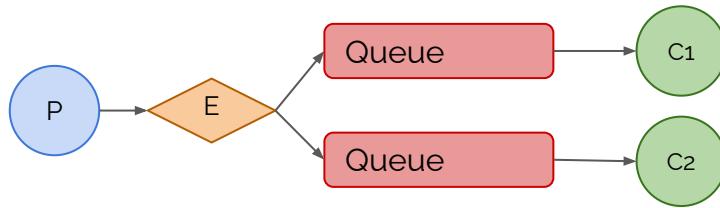
Add queue

Publish into "logs"  
exchange

logs_queue_1	classic	D Args	idle	1	0	1	0.00/s		
logs_queue_2	classic	D Args	idle	1	0	1	0.00/s		

EXERCISE

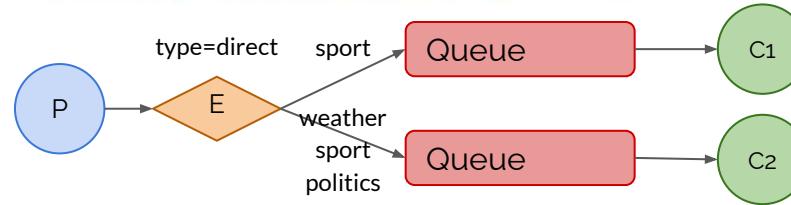
# Publish / Subscribe - Summary



```
> rabbitmqctl list_exchanges  
> rabbitmqctl list_bindings
```

just broadcasts all the messages it receives to all the queues it knows - ignores routing key

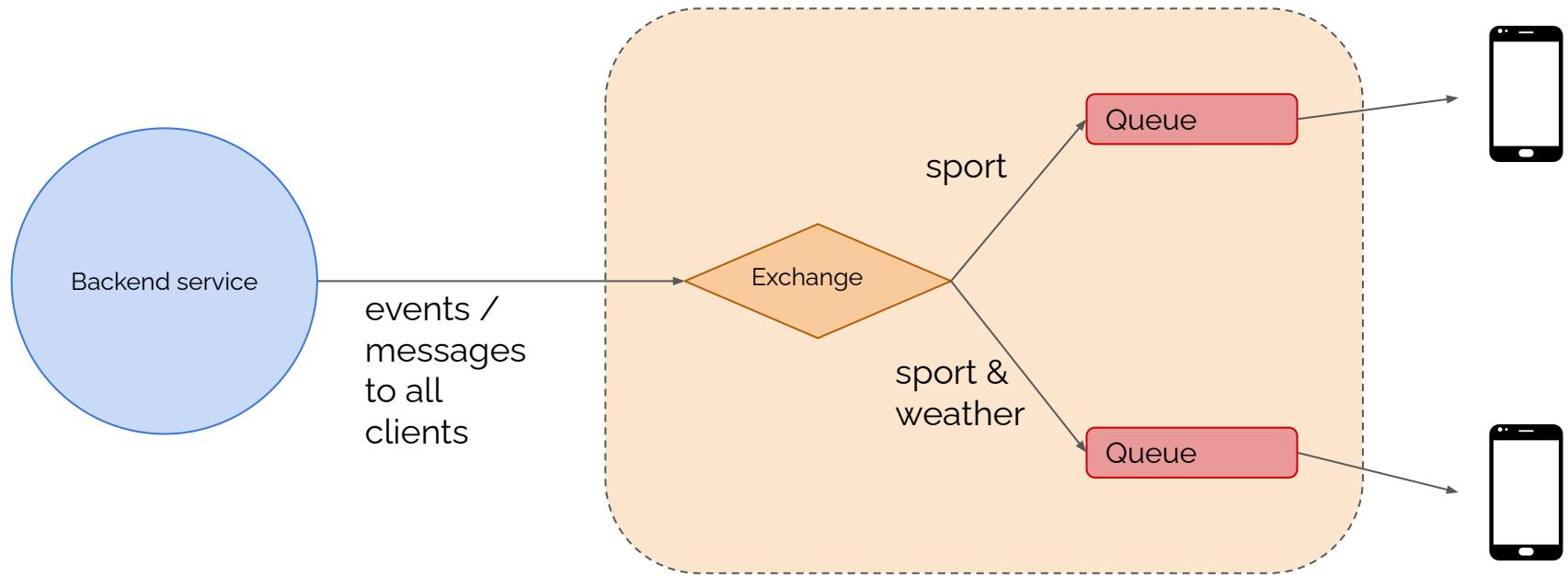
We created non-durable, exclusive, autodelete queue with a generated name like:  
`amq.gen-JzTY20BRgKO-HjmUJj0wLg`



## Publish / Subscribe based on Routing

Sample usecases:

- Notifications based on key
- Feeds based on key



 RabbitMQ



# Producer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.DIRECT);
channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);

channel.basicPublish(EXCHANGE_NAME, "info", null, "Information message".getBytes("UTF-8"));
channel.basicPublish(EXCHANGE_NAME, "warning", null, "Warning message".getBytes("UTF-8"));
channel.basicPublish(EXCHANGE_NAME, "error", null, "Error message".getBytes("UTF-8"));
channel.basicPublish(EXCHANGE_NAME, "critical", null, "Critical message".getBytes("UTF-8"));
```



Exchange type can be one of: fanout, direct, topic , headers

# Consumer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.DIRECT);

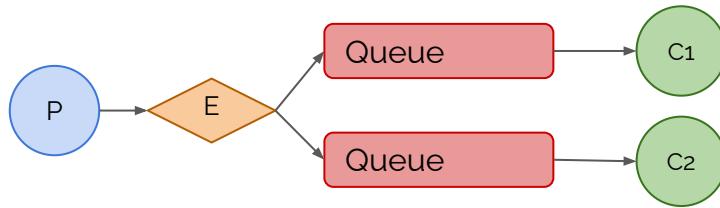
String queueName = channel.queueDeclare().getQueue();
channel.queueBind(queueName, EXCHANGE_NAME, /*routingKey*/"info");
channel.queueBind(queueName, EXCHANGE_NAME, /*routingKey*/"warning");

System.out.println(" [*] Waiting for messages....");

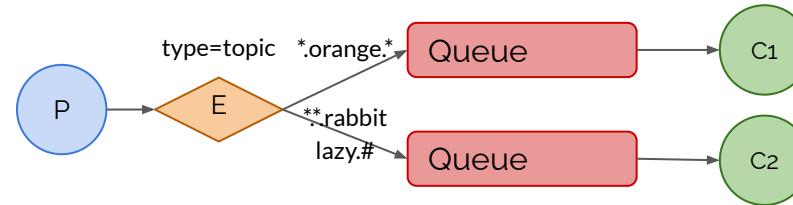
DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Received '" + message + "'");
};

channel.basicConsume(QUEUE_NAME, /*autoAck*/true, deliverCallback, consumerTag -> { });
```

# Publish / Subscribe - Summary



- Direct exchange routes to specific queue
- Routing key is matched with binding key to route **subset of messages** to bound queues
- Many categories of messages cause lot of bindings - it complicates administration of RabbitMQ



## Publish / Subscribe based on Topics

Sample usecases:

- Notifications based on topic
- Feeds based on topic

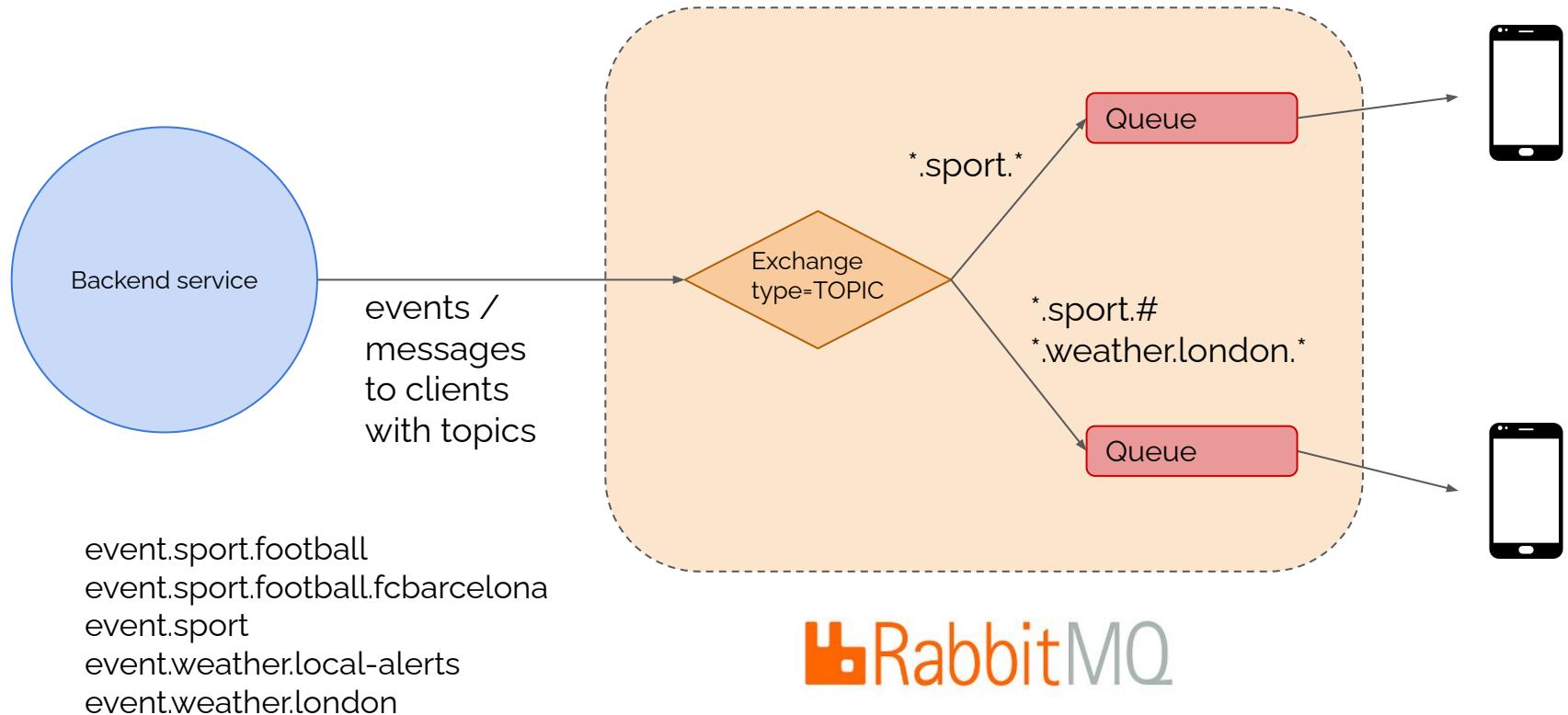


Topic is a kind of routing key defined as list of words, delimited by dots



Binding is a simple regular expression where:  
\* (star) can substitute exactly one word  
# (hash) can substitute zero or more words

# EXERCISE



# Producer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.TOPIC);

channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);

channel.basicPublish(EXCHANGE_NAME, "application1.logs.error", null, "Sample Error Message from App1".getBytes("UTF-8"));
channel.basicPublish(EXCHANGE_NAME, "application1.logs.info", null, "Sample Info Message from App1".getBytes("UTF-8"));
channel.basicPublish(EXCHANGE_NAME, "application1.metrics.memory", null, "Sample Info Message from App1".getBytes("UTF-8"));
channel.basicPublish(EXCHANGE_NAME, "application1.metrics.cpu", null, "Sample Info Message from App1".getBytes("UTF-8"));

channel.basicPublish(EXCHANGE_NAME, "application2.logs.error", null, "Sample Error Message from App2".getBytes("UTF-8"));
channel.basicPublish(EXCHANGE_NAME, "application2.logs.debug", null, "Sample Debug Message from App2".getBytes("UTF-8"));
```



Exchange type can be one of: fanout, direct, topic , headers  
(TopicsExample.java)

# Consumer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.TOPIC);

String queueName = channel.queueDeclare().getQueue();
channel.queueBind(queueName, EXCHANGE_NAME, /*routingKey*/".logs.error");
channel.queueBind(queueName, EXCHANGE_NAME, /*routingKey*/"application1.#");

System.out.println(" [*] Waiting for messages....");

DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Received '" + message + "'");
};

channel.basicConsume(QUEUE_NAME, /*autoAck*/true, deliverCallback, consumerTag -> { });
```

# Publish/Subscribe

Add a new exchange

Name: topic\_logs \*Type: topic  
Durability: Durable  
Auto delete: No  
Internal: No  
Arguments:

Add Alternate exchange ?

Add exchange

Add a new queue

Type: Classic  
Name: logs\_topic\_queue \*  
Durability: Durable  
Auto delete: No  
Arguments:  
Add Message TTL ? | Auto expire  
Dead letter exchange ? | De  
Lazy mode ? | Master locator

Add queue

Add binding to this queue

From exchange: topic\_logs  
Routing key: \*.logs.error  
Arguments:

Bind

Exchange: topic\_logs

Overview

Bindings

Publish message

Routing key: application1.logs.error  
Headers: ?  
Properties: ?  
Payload: Sample Error Message

Publish message

Bind using topic: ".logs.error"

Publis 2 messages with routing keys (second one won't be routed - note the DLX can be useful feature):  
application1.logs.error  
application1.logs.info

application1.metrics.cpu

logs_topic_queue	classic	D	Args	idle	1	0	1	0.00/s	

EXERCISE

# Publish / Subscribe based on Topics

## - Summary

- **Topic limit is 255 characters**

List of words, delimited by dots

- **No words limit**

A word can be any string; should specify features hierarchy

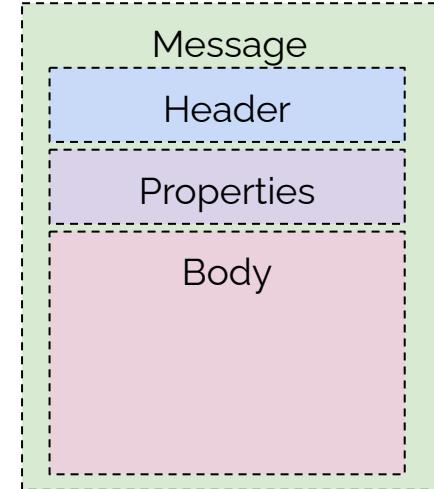
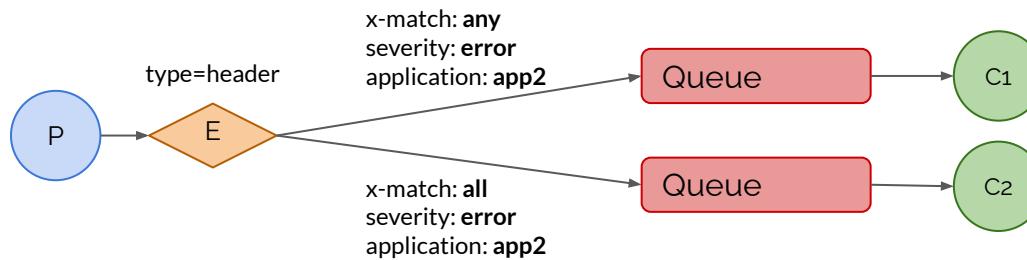
- **Substitutions**

\* (star) exactly one word

# (hash) zero or more words



Define topic names wisely! - Topic must represent a hierarchy, i.e.:  
`event.weather.south-east.london`



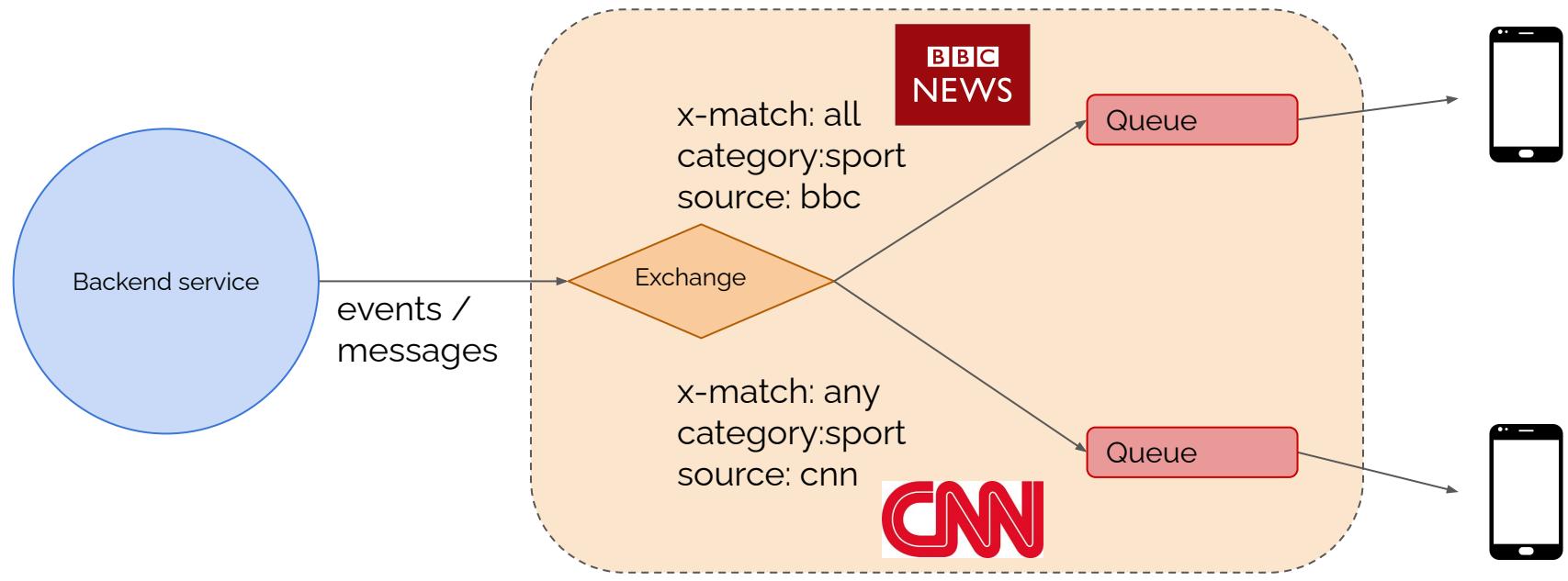
## Publish / Subscribe based on Headers

Sample usecases:

- Notifications based on header
- Feeds based on header



headers beginning with the string **x-** are not used to evaluate matches



RabbitMQ



# Publish/Subscribe

To	Routing key	Arguments	
q.messages.consumer1		severity: error user: john x-match: all	<button>Unbind</button>
q.messages.consumer2		severity: error user: john x-match: any	<button>Unbind</button>

Add binding from this exchange

To queue :  \*

Routing key:

Arguments:  =  String

Bind

**▼ Publish message**

Routing key:

Headers: ? severity = error String  
user = john String  
= String

Properties: ?  =

Payload: test



# Producer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.HEADERS);

channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);

HashMap<String, Object> map = new HashMap<>();
map.put("my-header-severity", "error");
map.put("my-custom-header", "hello");

AMQP.BasicProperties props = new AMQP.BasicProperties
    .Builder()
    .headers(map)
    .build();

channel.basicPublish(EXCHANGE_NAME, /*routingKey*/"", props, "Sample Error Message".getBytes("UTF-8"));
```

# Consumer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.HEADERS);

String queueName = channel.queueDeclare().getQueue();

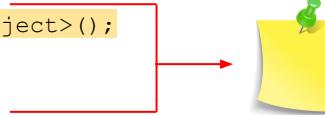
HashMap<String, Object> map = new HashMap<String, Object>();
map.put("x-match", "any"); //all or any
map.put("my-header-severity", header);
map.put("my-custom2", "bambo");

channel.queueBind(queueName, EXCHANGE_NAME, /*routingKey*/"", map);

System.out.println(" [*] Waiting for messages....");

DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Received '" + message + "'");
};

channel.basicConsume(QUEUE_NAME, /*autoAck*/true, deliverCallback, consumerTag -> { });
```



headers for routing messages from exchange to the queue. **All** must match vs **Any (at least one)** must match

# Publish / Subscribe based on Headers

## - Summary

- **Uses headers from AMQP message structure**

Key -> value pairs

- **No substitutions**

Header must exactly match to the list of headers defined in the binding

- **Ignores routing key**

Like fanout exchange, headerds exchange ignores routing key

- **More flexible than direct exchange**

But sometimes harder to maintain



headers beginning with the string **x-** are not used to evaluate matches

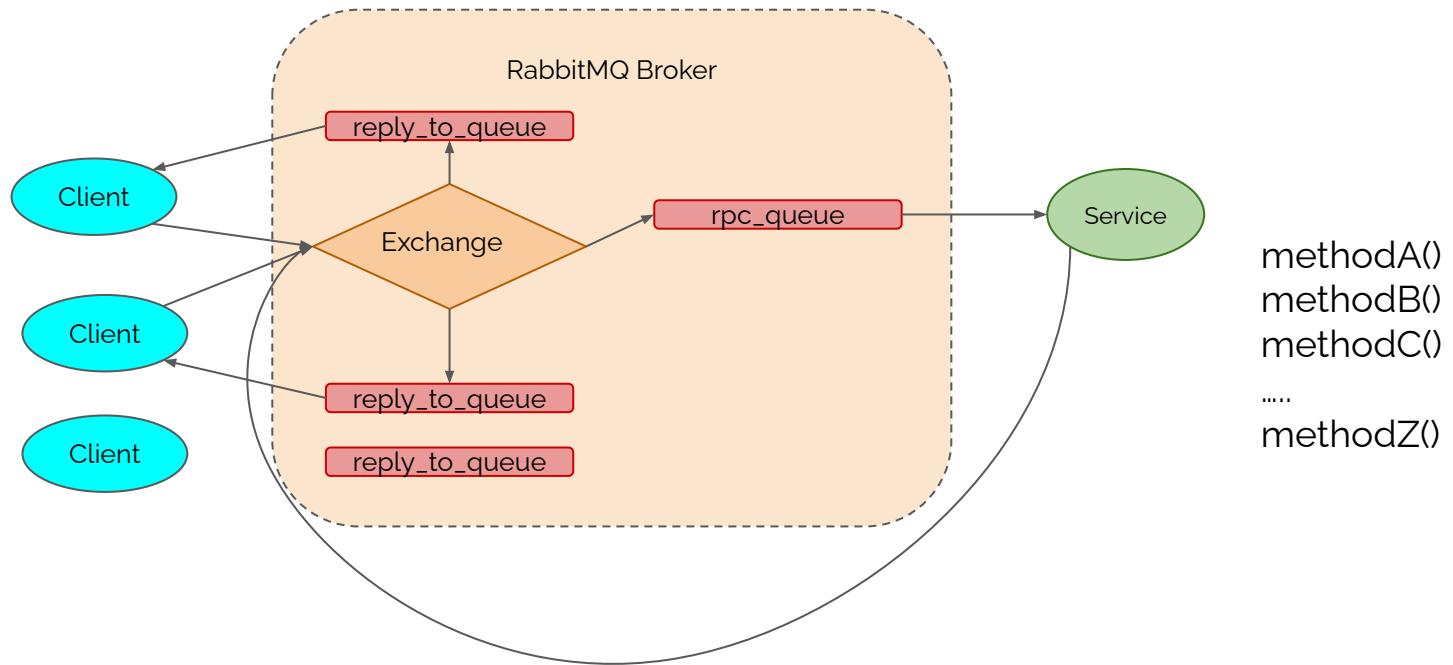


## RPC - Remote Procedure Call

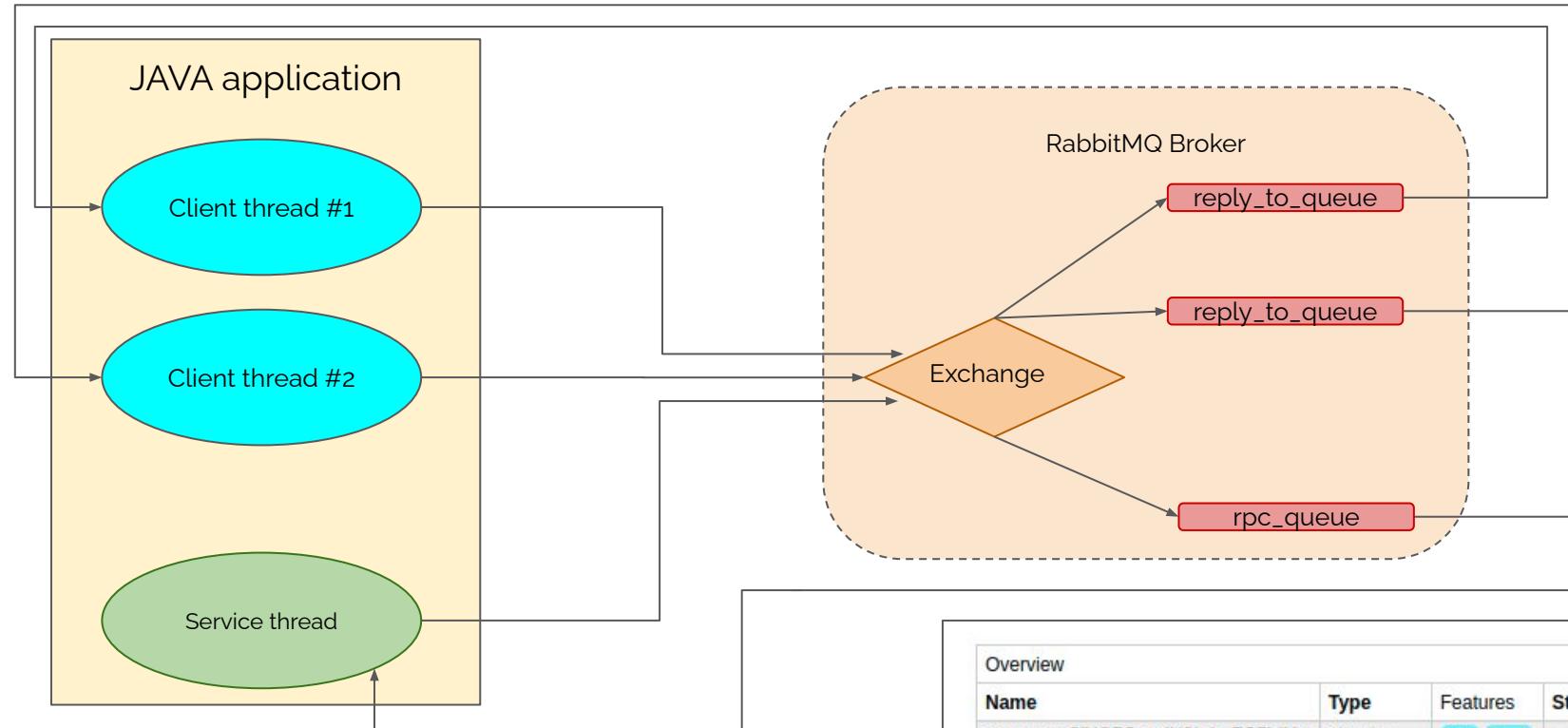
Sample usecases:

- Out of the box RPC implementation
- Distribute tasks among multiple clients (workers) - and wait for result

# RPC - Remote Procedure Call



# RPC - Remote Procedure Call



Overview					Message
Name	Type	Features	State		
amq.gen-8iYQ5CsgdV2iqIm7C5LjVw	classic	AD Excl	idle		
rpc_queue	classic		idle		

▶ Add a new queue

# RPC Client (producer)

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

final String corrId = UUID.randomUUID().toString();

String replyQueueName = channel.queueDeclare().getQueue();
AMQP.BasicProperties props = new AMQP.BasicProperties
    .Builder()
    .correlationId(corrId)
    .replyTo(replyQueueName)
    .build();

channel.basicPublish(/*exchange*/ "", RPC_QUEUE_NAME, props, message.getBytes("UTF-8"));

channel.basicConsume(replyQueueName, true, (consumerTag, delivery) -> {
    if (delivery.getProperties().getCorrelationId().equals(corrId)) {
        System.out.println(" [*] Received response !");
    }
}, consumerTag -> { });
```

# RPC Server (consumer)

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(RPC_QUEUE_NAME, false, false, false, null);
channel.queuePurge(RPC_QUEUE_NAME);

channel.basicQos(1);

System.out.println(" [*] Waiting for messages....");

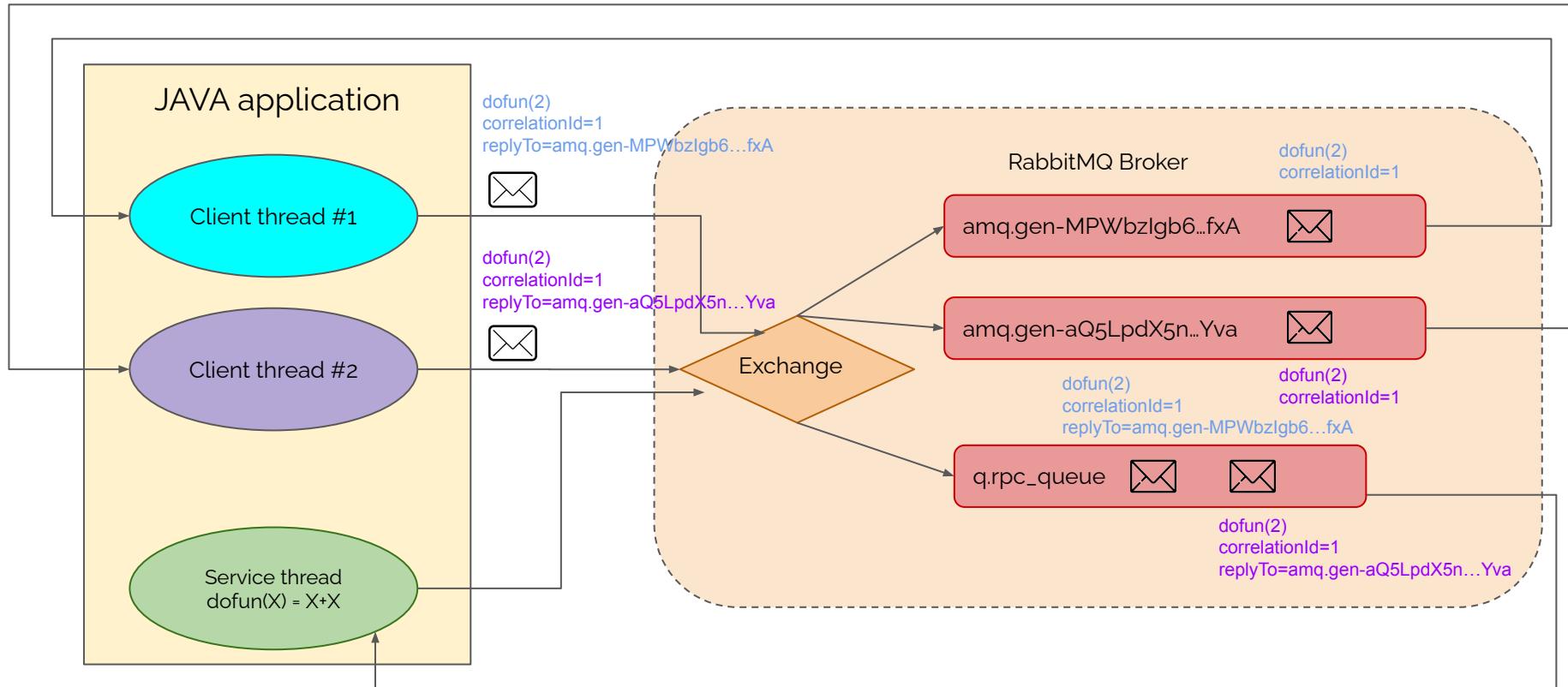
DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    AMQP.BasicProperties replyProps = new AMQP.BasicProperties
        .Builder()
        .correlationId(delivery.getProperties().getCorrelationId())
        .build();

    System.out.println(" [x] Doing the Math ...");

    channel.basicPublish(/*exchange*/ "", delivery.getProperties().getReplyTo(), replyProps, response.getBytes("UTF-8"));
    channel.basicAck(delivery.getEnvelope().getDeliveryTag(), /*multiple*/false);
};

channel.basicConsume(RPC_QUEUE_NAME, /*autoAck*/false, deliverCallback, consumerTag -> { });
```

# RPC - Remote Procedure Call



# RPC - Remote Procedure Call

## → RabbitMQ can be used a nice RPC wrapper

Any RPC implementation based on queues is fine, here we have an API out of the box

## → Single queue

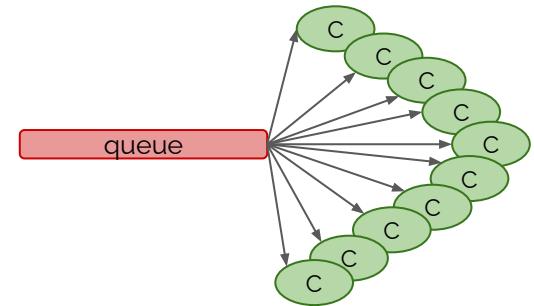
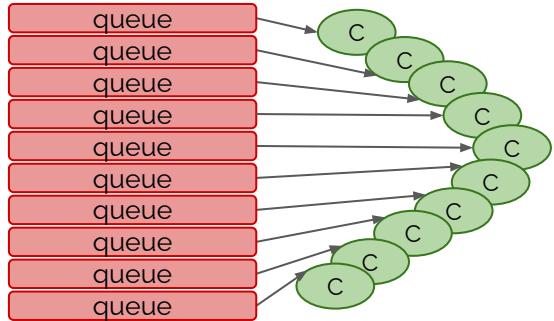
Requests and responses goes through single queue;  
both matched by **correlationId** property,



CorrelationID is one of 14 messages properties defined in AMQP 0-9-1 protocol (like deliveryMode, replyTo, contentType etc.)

## → General RPC good habits

- ◆ Make a clear comment in the code that function call is local or remote,
- ◆ Keep up-to-date documentation about dependencies between components,
- ◆ Handle communication issues, i.e. when RPC server is down;  
Scale-out RPC service by combining RPC pattern with Work Queues or more sophisticated exchanges like consistent hash,  
and so on



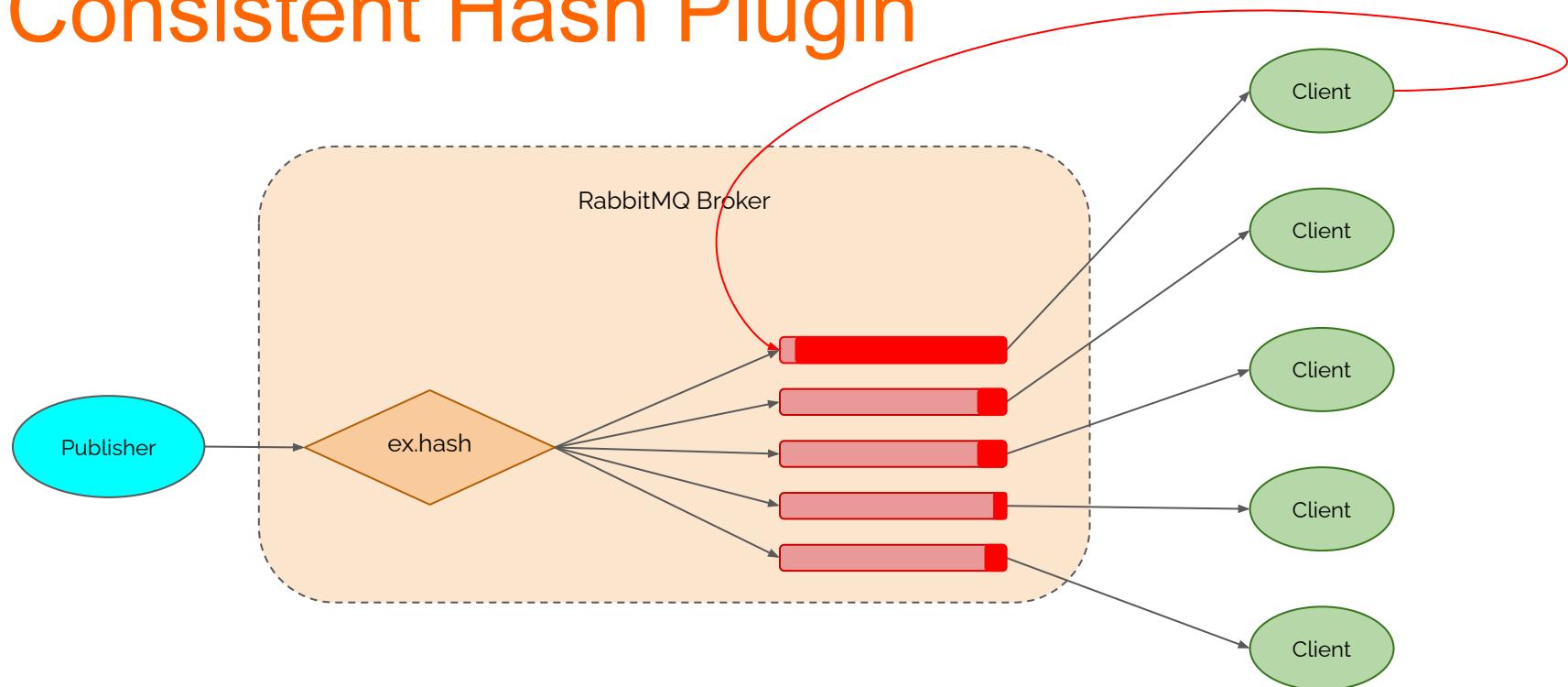
## Custom exchanges - consistent hash

Sample usecases:

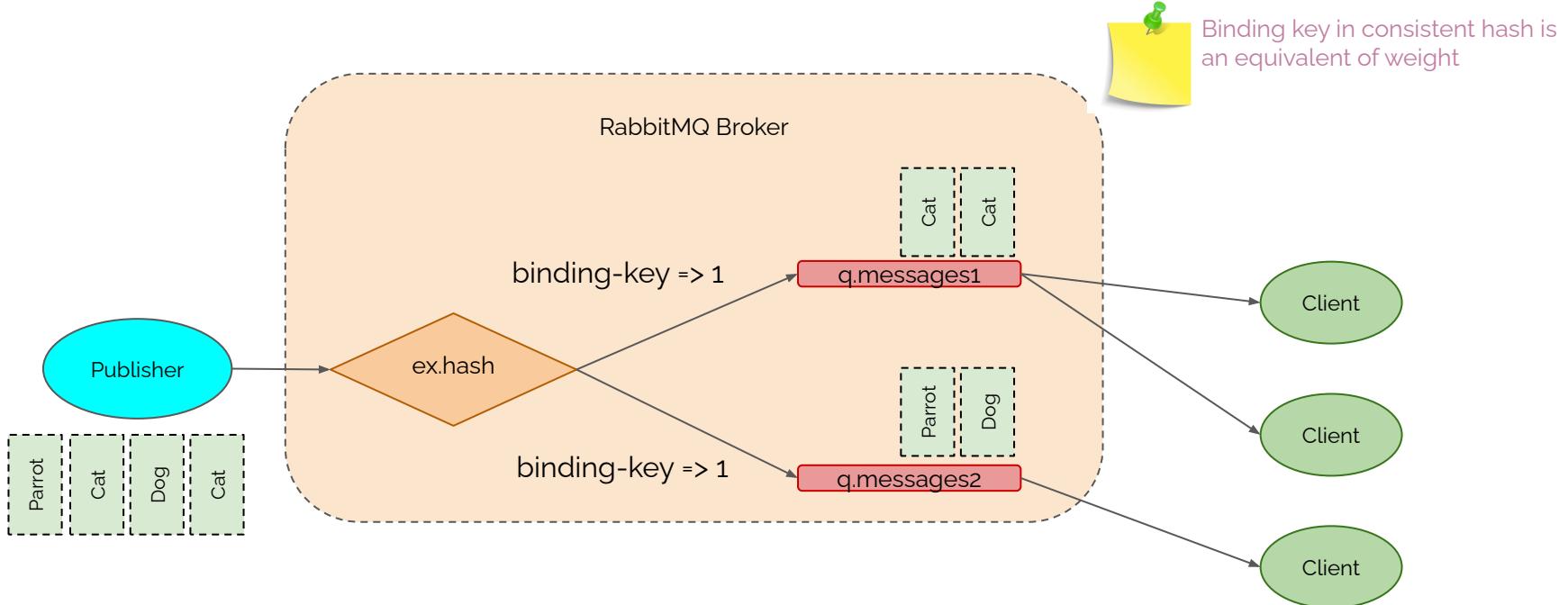
- Split long queue into smaller ones
- Distribute logically related messages across many queues (to reduce latency from the consumer's perspective and increase reliability)

# Consistent Hash Plugin

Reject with requeue=true



# Consistent Hash Plugin



# Consistent Hash Plugin

```
> cd %RABBITMQ%\rabbitmq_server-3.8.5\sbin  
> set ERLANG_HOME=c:\Program Files\erl-23.0.3  
> rabbitmq-plugins.bat enable rabbitmq_consistent_hash_exchange
```

```
Enabling plugins on node rabbit1@localhost:  
rabbitmq_consistent_hash_exchange  
The following plugins have been configured:  
rabbitmq_consistent_hash_exchange  
rabbitmq_federation  
rabbitmq_federation_management  
rabbitmq_management  
rabbitmq_management_agent  
rabbitmq_shovel  
rabbitmq_shovel_management  
rabbitmq_web_dispatch  
Applying plugin configuration to rabbit1@localhost...  
The following plugins have been enabled:  
rabbitmq_consistent_hash_exchange  
  
started 1 plugins.
```

▼ Add a new exchange

Virtual host:	/
Name:	<input type="text"/>
Type:	direct
Durability:	direct fanout headers topic
Auto delete:	<input type="checkbox"/>
Internal:	x-consistent-hash
Arguments:	<input type="text"/> = <input type="text"/> String
Add Alternate exchange <input type="button"/>	
<input type="button" value="Add exchange"/>	



Dead Letter Exchange (DLX)

# Dead Letter Exchange (DLX)

Messages from the queue can be dead-lettered when

- **Message is negatively (ack)nowledged**

When Nack or Reject with requeue=false is called

- **TTL expired**

When message expires due to per-message TTL,

- **Message is dropped**

Queue exceeded length limit

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

# Delay retry/schedule with DLX - Producer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(DLX_EXCHANGE_NAME, "direct");    // dead letter exchange
channel.queueDeclare(DLX_QUEUE_NAME, /*durable*/true, /*exclusive*/false, /*autoDelete*/false, /*arguments*/null);
channel.queueBind(DLX_QUEUE_NAME, DLX_EXCHANGE_NAME, "some-routing-key", null);

Map<String, Object> args = new HashMap<String, Object>();
args.put("x-dead-letter-exchange", DLX_EXCHANGE_NAME);
args.put("x-message-ttl", RETRY_DELAY );
args.put("x-dead-letter-routing-key", "some-routing-key");

channel.exchangeDeclare(RETRY_EXCHANGE, "direct");
channel.queueDeclare(RETRY_QUEUE, true, false, false, args);
channel.queueBind(RETRY_QUEUE, RETRY_EXCHANGE, "", null);

String message = "Hello World " + i;//System.currentTimeMillis();
channel.basicPublish(/*exchange*/RETRY_EXCHANGE, /*routingKey*/"", null, message.getBytes(StandardCharsets.UTF_8));
```

# Delay retry/schedule with DLX - Consumer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

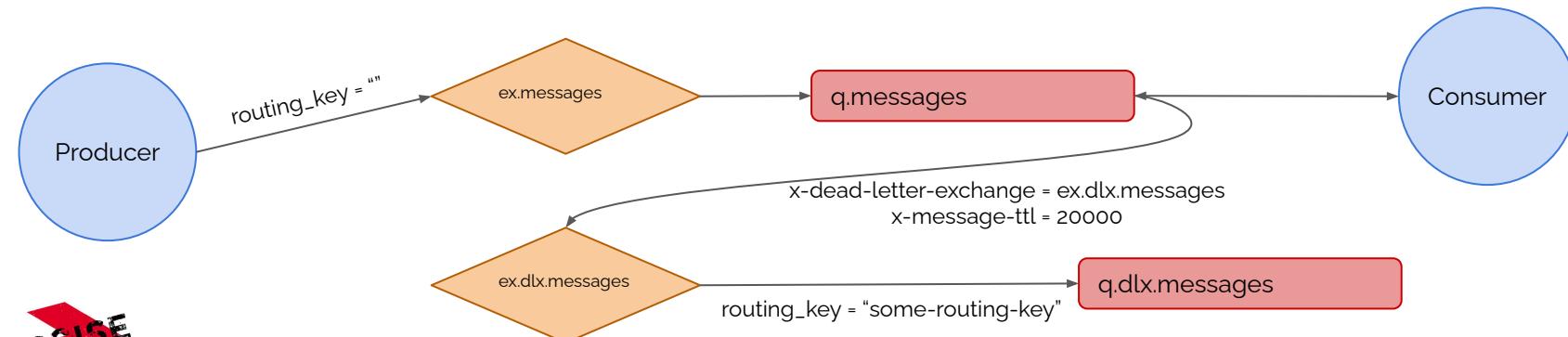
DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Rejecting '" + message + "' with routingKey=" + delivery.getEnvelope().getRoutingKey());
    channel.basicNack(delivery.getEnvelope().getDeliveryTag(), false, /*requeue*/false);
};

channel.basicConsume(RETRY_QUEUE, /*autoAck*/false, deliverCallback, consumerTag -> { });
```

# DLX example: Hands-On

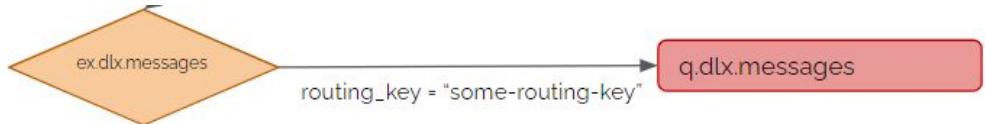
1. Create a DLX exchange
2. Create a DLX queue
3. Bind DLX exchange with DLX queue
  
4. Create an exchange to be used by publisher
5. Create queue with x-dead-letter-exchange attribute
6. Bind them

Consumer to read with nACK and requeue=false



**EXERCISE**

# DLX example



▼ Add a new queue

Type: Classic ▾  
Name: dlxqueue2 \*  
Durability: Durable ▾  
Auto delete: ? No ▾  
Arguments:  =  
Add Message TTL ? | Auto ex  
Dead letter exchange ? |  
Lazy mode ? Master loc

Add queue

Add binding to this queue

From exchange: some.exchange.name \*  
Routing key: some-routing-key  
Arguments:  =

Bind

▼ Add a new exchange

Name: some.exchange.name \*  
Type: direct ▾  
Durability: Durable ▾  
Auto delete: ? No ▾  
Internal: ? No ▾  
Arguments:  =  
Add Alternate exchange ?

Add exchange

EXERCISE

# DLX example

▼ Add a new exchange

Name:	RetryExchange
Type:	direct
Durability:	Durable
Auto delete:	No
Internal:	No
Arguments:	

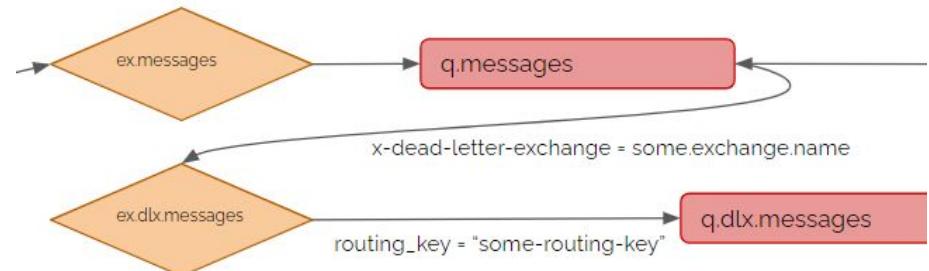
Add Alternate exchange ?

**Add exchange**

Add binding to this queue

From exchange:	RetryExchange
Routing key:	
Arguments:	

**Bind**



▼ Add a new queue

Type:	Classic
Name:	RetryQueue
Durability:	Durable
Auto delete:	No
Arguments:	x-dead-letter-exchange = some.exchange.name x-message-ttl = 10000 x-dead-letter-routing-key = some-routing-key =

Add Message TTL ? | Auto expire ? | Max length ? | Max length bytes  
Dead letter exchange ? | Dead letter routing key ? | Single active cc  
Lazy mode ? | Master locator ?

**Add queue**

EXERCISE →

# DLX example

## Exchange: RetryExchange

Overview

Message rates last minute ?

Currently idle

Details

Type: direct  
Features: durable: true

Policy

Bindings

Publish message

Routing key:

Headers:  =

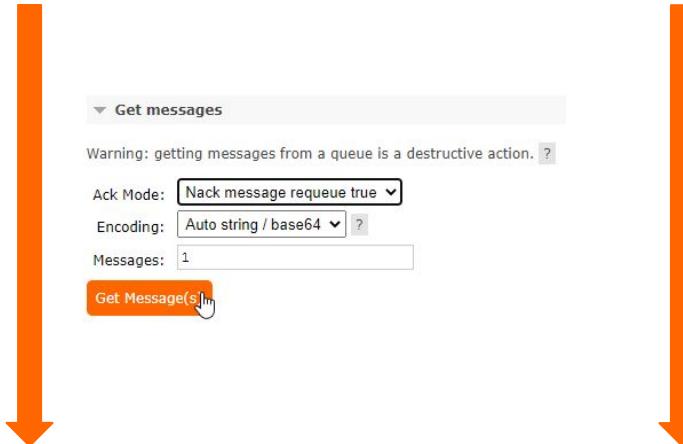
Properties:  =

Payload: test

**Publish message**



Overview					Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
RetryQueue	classic	D TTL DLX DLK Args	idle	1	0	1	0.00/s			
dlxqueue2	classic	D Args	idle	0	0	0				



Alternatively after 10s without consumption

Overview					Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
RetryQueue	classic	D TTL DLX DLK Args	idle	0	0	0	0.00/s	0.00/s	0.00/s	
dlxqueue2	classic	D Args	idle	1	0	1				

... - - -

EXERCISE

# Dead Letter Exchange (DLX)

Dead letter exchange

- **Manage rejected messages**

When Nack or Reject with requeue=false is called

- **Manage queue capacity**

When message expires due to per-message TTL, or queue size exceeded length limit

- **Delay publication**

Intentional latency increase between publisher and consumer.

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

# Delay retry / delay schedule with DLX

When to use

- **No need or forbidden to consume messages immediately**

Publish messages, but allow consumers to consume them after 3:00 PM Friday after conference press

- **Automatic retrying**

Rejected message to be resend again in about 10 minutes

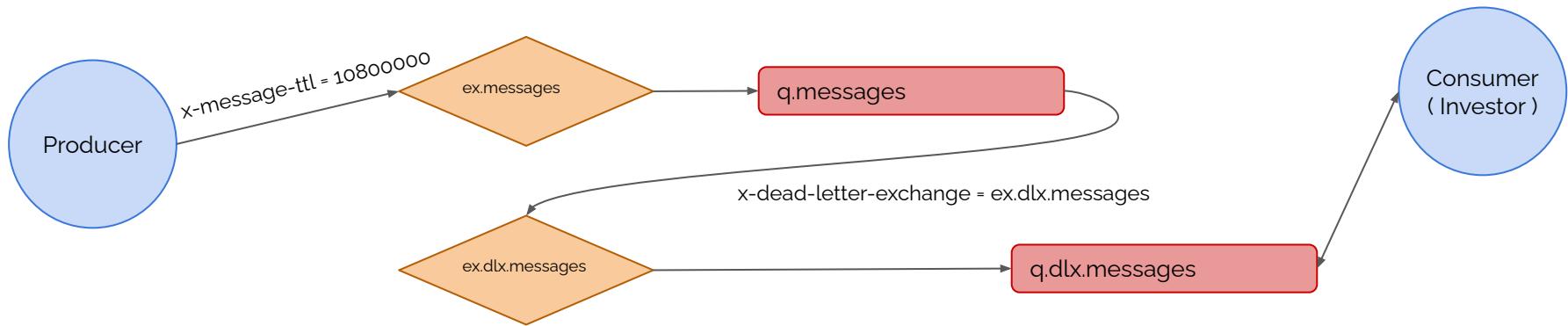
- **Delay and “batch” publication**

Intentional latency increase between publisher and consumer or group messages into bigger batches



Consider simple solutions over installing additional plugins like rabbitmq-delayed-message-exchange.  
Such plugins store messages in mnesia table, which is not designed to store high volume of the data

# Delay schedule with DLX



Publication at - 1:00 PM

Press conference starts - 3:00 PM

Press conference ends - 4:00 PM

TTL = 10 800 seconds

# Delay retry/schedule with DLX - Retry consumer

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    String message = new String(delivery.getBody(), "UTF-8");
    String key = delivery.getEnvelope().getRoutingKey();
    System.out.println(" [x] Received rejected message: '" + message + "' with routingKey=" + key);
};

channel.basicConsume(DLX_QUEUE_NAME, /*autoAck*/true, deliverCallback, consumerTag -> { });
```



- RabbitMQ has no support for delayed/scheduled messages. Feature can be used when:
- there is no need to read messages immediately.
  - message I read can't be processed or I want to requeue it and try again in about 10 minutes
  - group messages into batches or reduce consumer's load

# Delay retry/schedule with DLX - Retry consumer

```
--> Running producer
[x] Sent 'Hello World 0'
--> Running consumer for rejected messages
[*] Waiting for rejected messages....
[x] Sent 'Hello World 1'
[x] Sent 'Hello World 2'
[x] Sent 'Hello World 3'
--> Running consumer
[*] Waiting for messages....
[x] Rejecting 'Hello World 0' with routingKey=
[x] Rejecting 'Hello World 1' with routingKey=
[x] Rejecting 'Hello World 2' with routingKey=
[x] Rejecting 'Hello World 3' with routingKey=
[x] Received rejected message: 'Hello World 0' with routingKey=some-routing-key (reason=rejected)
[x] Received rejected message: 'Hello World 1' with routingKey=some-routing-key (reason=rejected)
[x] Received rejected message: 'Hello World 2' with routingKey=some-routing-key (reason=rejected)
[x] Received rejected message: 'Hello World 3' with routingKey=some-routing-key (reason=rejected)
[x] Sent 'Hello World 4'
[x] Rejecting 'Hello World 4' with routingKey=
[x] Received rejected message: 'Hello World 4' with routingKey=some-routing-key (reason=rejected)
--> Closing consumer
--> Consumer closed
[x] Sent 'Hello World 5'
[x] Sent 'Hello World 6'
[x] Received rejected message: 'Hello World 5' with routingKey=some-routing-key (reason=expired)
[x] Received rejected message: 'Hello World 6' with routingKey=some-routing-key (reason=expired)
```



Transactions & Publisher Confirms

# Data safety

Acknowledgements on both consumer and publisher side are important for data safety

*Defaults*

→ **Consumer Acknowledgment**

One of **ACK**, **nACK** (RabbitMQ extension for Reject to reject multiple messages at once),  
**Reject** (defined in AMQP protocol)

→ Auto Ack

→ **Transactions (tx)**

Safe batching feature (remember: batching Acks)

→ Disabled

→ **Publisher Confirms**

Broker to send confirmation once message is safely stored (RabbitMQ feature)

→ Disabled

**ACK**

- Positive acknowledgment

→ remove message from the queue

**nACK or Reject**

- Negative acknowledgment with **requeue=true**

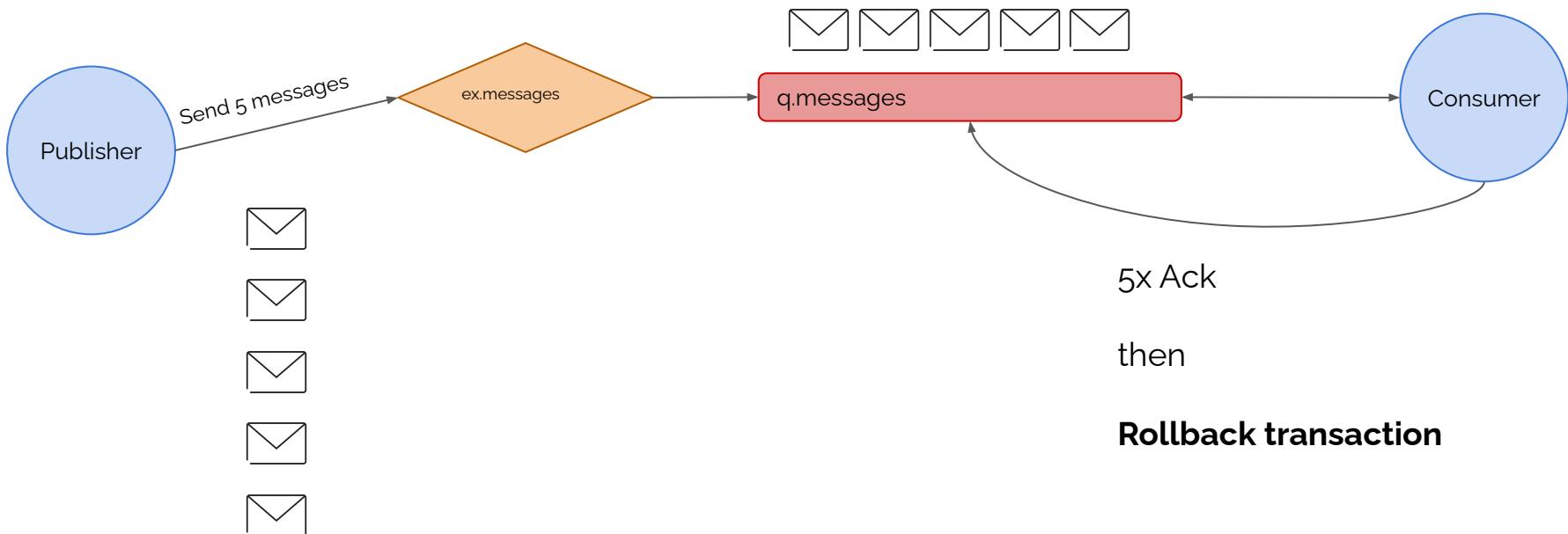
→ keep message in the queue

**nACK or Reject**

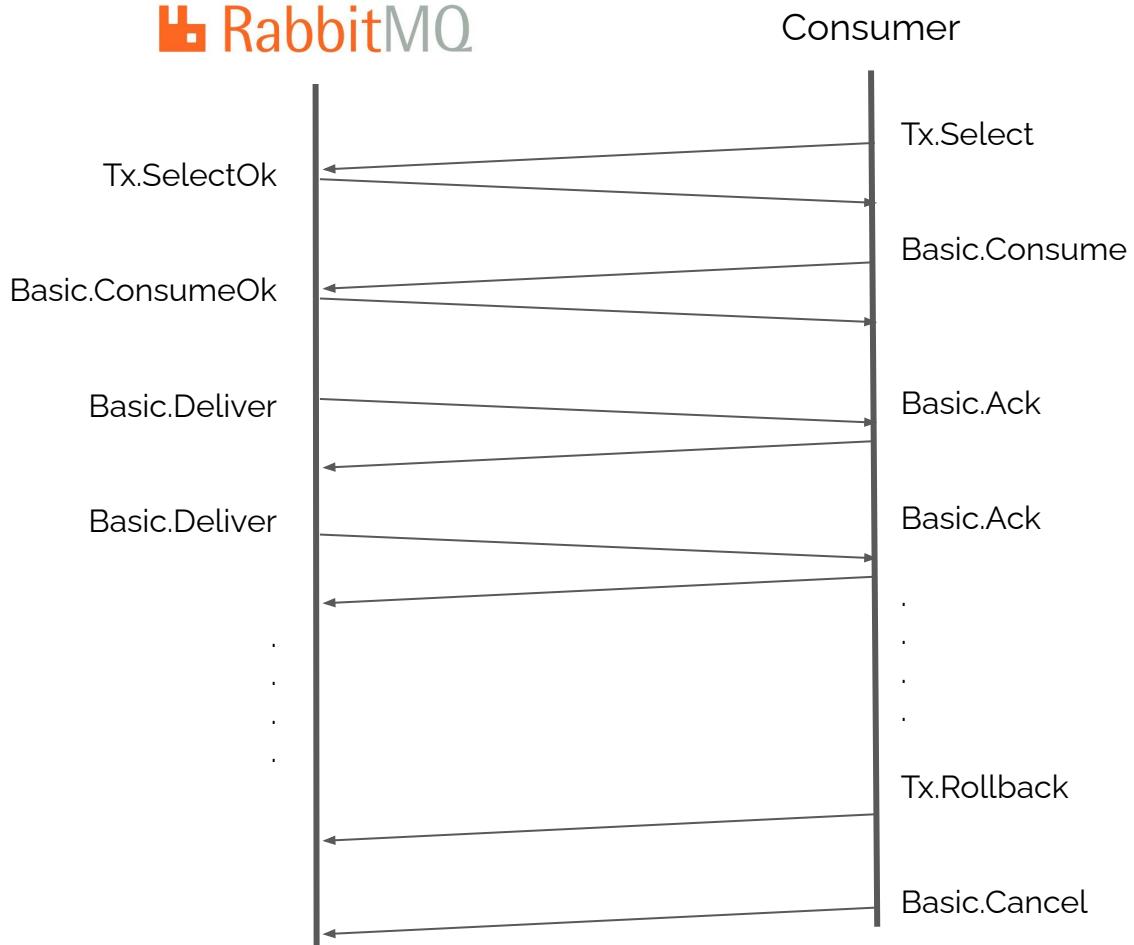
- Negative acknowledgment with **requeue=false**

→ remove message from the queue

# Transactions



# Transactions



# Transactions



**Transactions decrease performance.** Overall throughput is decreased by factor close to 250. Consider "publisher confirms" or smart acknowledgments implementation.

Publisher

 RabbitMQ

Tx.Select

Basic.Publish

Basic.Publish

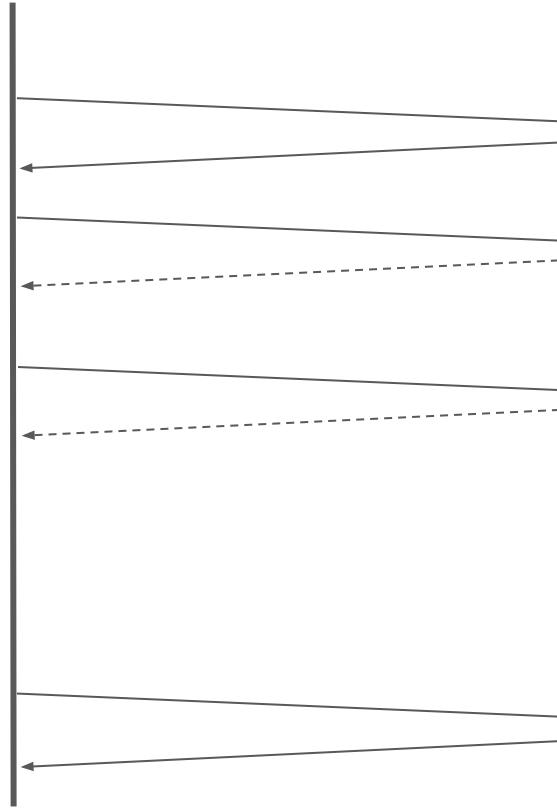
Tx.Commit

Tx.SelectOK

may respond  
with  
Basic.Return

may respond  
with  
Basic.Return

Tx.CommitOk



# Transactions (tx) - batch

Transactions in RabbitMQ are closer to batching feature than ACID concept.

**txSelect** - set channel to use transactions. Method must be called at least once before calling txCommit or txRollback.

**txCommit/txRollback** - commits/rollbacks all message publications and (ack)nowledgments performed in current transaction

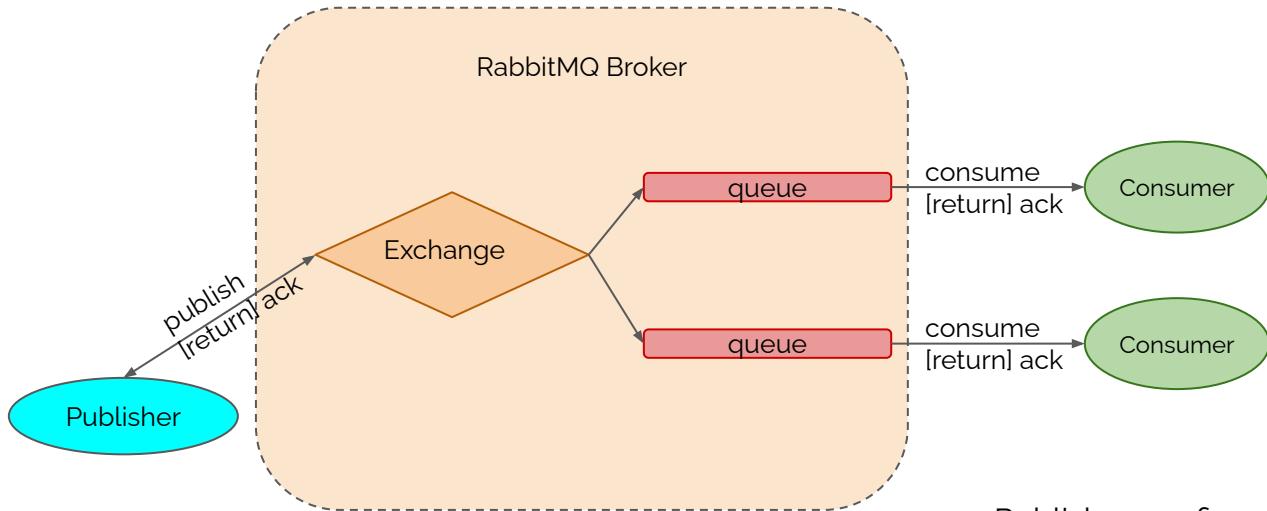
```
DeliverCallback deliverCallback = (consumerTag, delivery) -> {
    channel.txSelect();
    String message = new String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Received '" + message + "'");
    channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);

    if( message.startsWith("Final Message") ) {
        //channel.txCommit();
        channel.txRollback();
    }
};
channel.basicConsume(QUEUE_NAME, /*autoAck*/false, deliverCallback, consumerTag -> { });
```



**Transactions decrease performance;** consider "publisher confirms" or smart acknowledgments implementation.

# Publisher Confirms



Better than transactions - transactions can decrease throughput **by hundreds of times**

Publisher confirms is a RabbitMQ extension to implement reliable publishing (not enabled by default). To make sure published messages have safely reached the broker.

Both type of acknowledgments are essential for **data safety**

# Publisher Confirms

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");

Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, /*durable*/false, /*exclusive*/false, /*autoDelete*/true, /*arguments*/null);
channel.confirmSelect();

channel.basicPublish(/*exchange*/"", /*routingKey*/QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));

// uses a 5 second timeout to wait synchronously for confirmation
channel.waitForConfirmsOrDie(5_000);
```

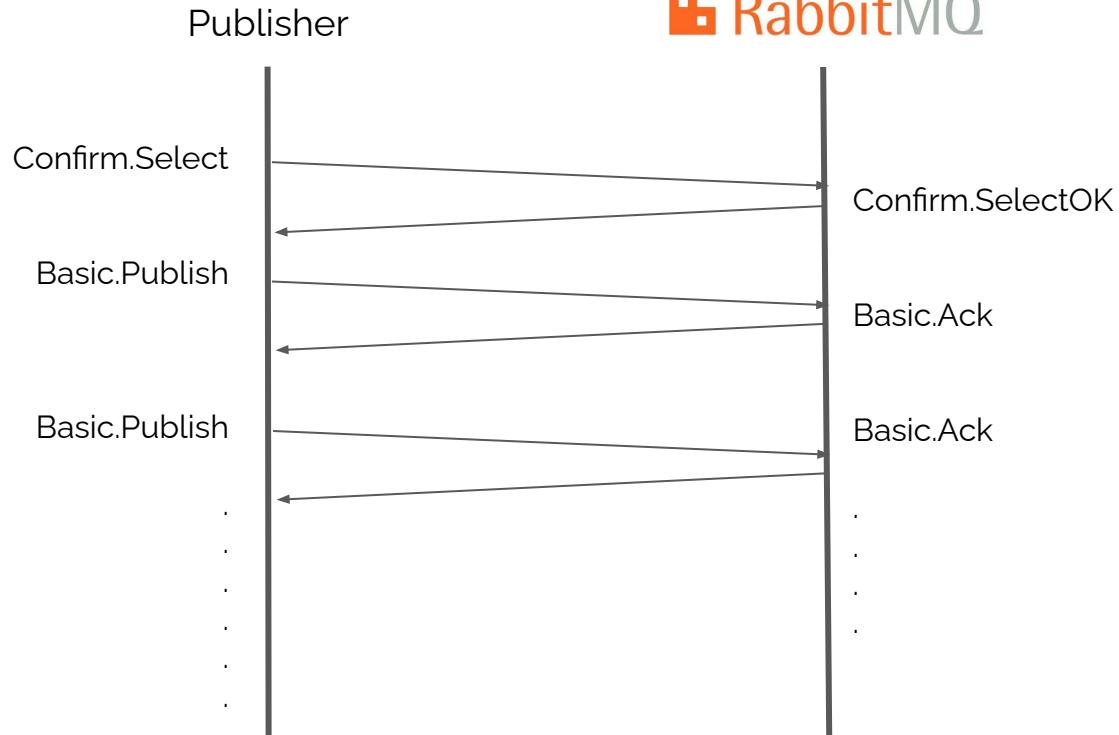


Ack is when all queues accepted the message and message is saved on the disk (persistent, durable mode)



**Performance warning!** Run example code and note:  
Published 50,000 messages in 1,768 ms  
Published 50,000 messages in 14,218 ms  
Published 50,000 messages in batch in 2,798 ms  
Published 50,000 messages and handled confirms asynchronously in 1,786 ms

# Publisher Confirms



 **Performance warning!** Run example code and note:

(no confirms)

Published 50,000 messages in **1,768 ms**

(confirms every message)

Published 50,000 messages in **14,218 ms**

(confirms in batch)

Published 50,000 messages in batch in **2,798 ms**

(confirms in batch)

Published 50,000 messages and handled confirms asynchronously in **1,786 ms**

# Data safety - summary

To make sure your system is reliable and to avoid losing messages due to any unpredicted network issues

- **Durable queue, durable exchange and persistent messages**  
To survive accidental RabbitMQ restart or failure
- **Consumer to manually confirm**  
Default mode is an automatic ACK
- **Transactions (tx)**  
Use transactions only when set of messages must be delivered as a package (all or nothing) and there is no way to send them as a single message (degraded overall throughput by factor of 250)
- **Publisher Confirms**  
Enable confirm mode and retry in case of nAck returned by RabbitMQ

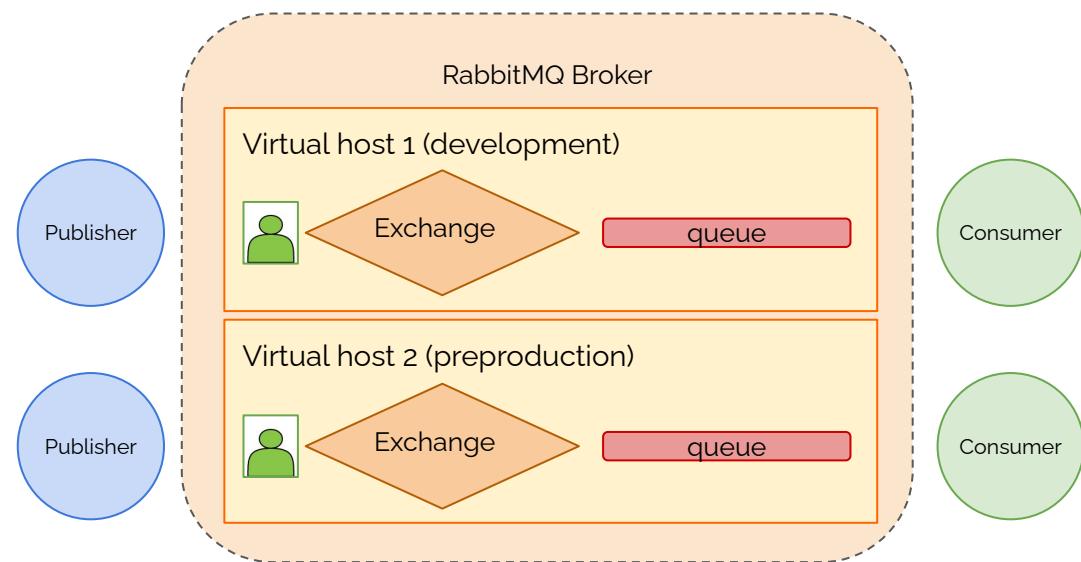


**Publisher Confirms** - wait asynchronously on confirmations to don't impact on throughput more than needed.



## Vhosts

Virtual hosts are logical groups of entities

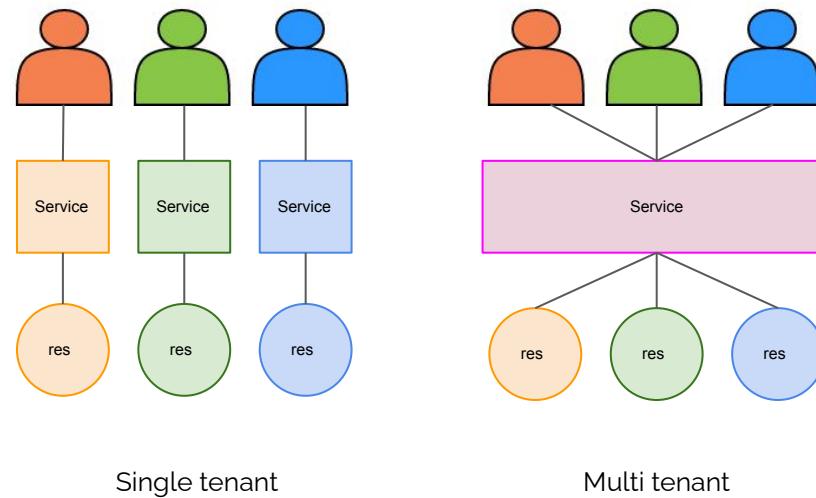


# Vhosts

Vhosts support multi-tenant architecture. Vhost can be an application name (app1, app2), environment (development, ppe, production) and so on.

Default vhost is / (here why we use %2f in REST api requests)

Separate service, separate resources  
vs  
Same service, separate resources



# Vhosts - Summary

VHosts introduce an additional level of logical isolation

- **Monitoring is better**

Connections, channels, exchanges, queues are created inside particular Vhost

- **Security - logically isolated playgrounds**

Resources can "talk each other" only when they exist in the same virtual host. Simplifies user's permissions

- **Security - Limits**

Apply limits of open connections, queues etc differently for each Vhost

- **Administration**

One cluster for bigger audience instead of many smaller ones.



**Isolation is only a function split!** -  
When exchange or queue in one  
Vhost causes performance issues,  
other Vhosts are impacted as well

# Vhosts REST API

## List all vhosts

```
curl -s -XGET -u guest:guest "http://localhost:15682/api/vhosts" -H "Content-Type: application/json"
```

## List all permissions for given vhost

```
curl -s -XGET -u guest:guest "http://localhost:15672/api/vhosts/app2_production/permissions" -H "Content-Type: application/json"
```

## List all users

```
curl -s -XGET -u guest:guest "http://localhost:15672/api/users" -H "Content-Type: application/json"
```

## List all users permissions

```
curl -s -XGET -u guest:guest "http://localhost:15672/api/users/guest/permission" -H "Content-Type: application/json"
```

## Queues

▼ All queues (8)

Pagination

Page  of 1 - Filter:   Regex

Displaying 8 items ,

Overview						Messages			Message rates		
Virtual host	Name	Node	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
/	application_logs	rabbit1@localhost	classic	D Args	idle	0	0	0			
app1_development	application_logs	rabbit1@localhost	classic	D Args	idle	0	0	0			
app2_production	application_logs	rabbit1@localhost	classic	D Args	idle	0	0	0			



Policies

# Policies

What policies are?

- **Optional arguments for groups of queues, exchanges or plugins**  
Flexibility of updating queue or exchange parameters

Why to use policies?

- **Dynamically change optional arguments for groups of queues, exchanges or plugins**  
Without policies, you must delete and create every resource again to change its properties
- **Vhost scoped**  
Different set of policies for each virtual host
- **Protocol specific settings**  
Plugins extend list of protocols and features are extensible



Good naming convention helps with applying policies!

# Policies

▼ Add / update a policy

Virtual host:  \*

Name:

Pattern:  \*

Apply to:  \*

Priority:

Definition:  =  String

Queues [All types] [Max length](#) | [Max length bytes](#) | [Overflow behaviour](#) [Auto expire](#) | ?  
[Dead letter exchange](#) | [Dead letter routing key](#)

Queues [Classic] [HA mode](#) ? | [HA params](#) ? | [HA sync mode](#) ?  
[HA mirror promotion on shutdown](#) ? | [HA mirror promotion on failure](#) ?  
[Message TTL](#) | [Lazy mode](#) | [Master Locator](#)

Queues [Quorum] [Max in memory length](#) ? | [Max in memory bytes](#) ? | [Delivery limit](#) ?

Queues [Stream] [Max age](#) ? | [Max segment size in bytes](#) ?

Exchanges [Alternate exchange](#) ?

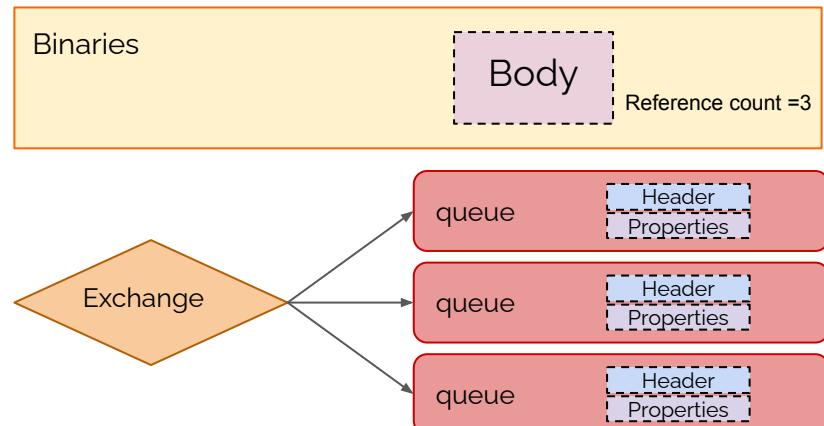
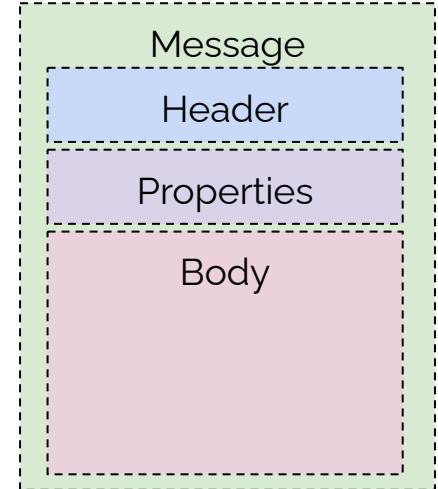
Federation [Federation upstream set](#) ? | [Federation upstream](#) ?



Lazy queues - memory optimization

# Queues - memory

- **Queues always keep part of messages in memory**  
RabbitMQ deliver messages to consumers as fast as possible
- **Queue is an Erlang process**  
Has its own heap (security & reliability)
- **Body is stored separately in a separate memory**  
Body of the message is stored in "Binaries"
- **Service booting**  
When RabbitMQ starts, up to 16384 messages smaller than 4k are loaded into memory



# Lazy Queues - experiment

1. Publish 1 000 000 messages, 1 KB each
2. Monitor node and queue memory used
3. Apply Lazy mode
4. Repeat publication of 1 000 000 messages, 1 KB each



# Lazy Queues

Lazy Queues move messages to disk as early as possible, and load them into RAM when requested by consumers. Used to support **very long queues** (many millions of messages).

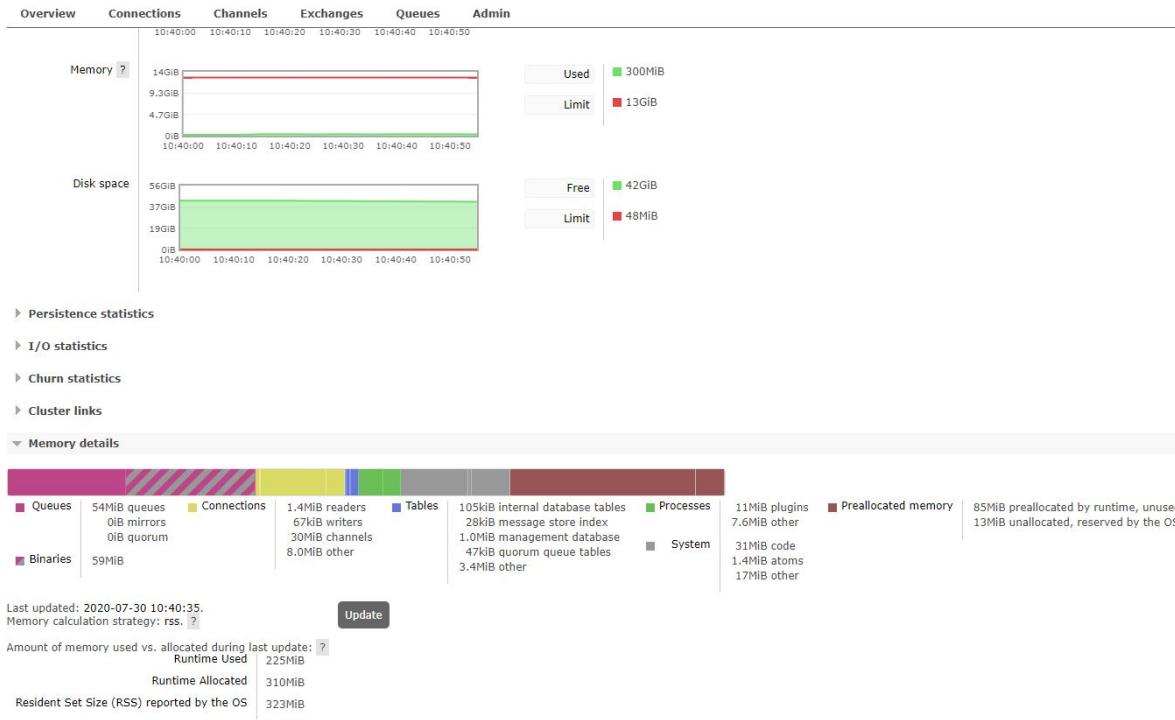
Queue mode	Queue process memory	Messages in memory	Memory used by messages	Node memory
default	257 MB	386,307	368 MB	734 MB
lazy	159 KB	0	0	117 MB



Performance test details are in  
“Performance testing” section of this  
presentation

# Lazy Queues

Sample results of performance test lazy vs not-lazy



id: test 1, time: 99,194s, sent: 18832 msg/s  
id: test 1, time: 100,318s, sent: 19243 msg/s  
id: test 1, time: 101,426s, sent: 10790 msg/s  
id: test 1, time: 102,736s, sent: 7338 msg/s  
id: test 1, time: 103,736s, sent: 34856 msg/s  
id: test 1, time: 104,850s, sent: 9694 msg/s  
id: test 1, time: 105,895s, sent: 16094 msg/s  
id: test 1, time: 106,918s, sent: 18791 msg/s  
id: test 1, time: 107,996s, sent: 20058 msg/s  
id: test 1, time: 109,050s, sent: 11398 msg/s  
id: test 1, time: 110,103s, sent: 15974 msg/s  
id: test 1, time: 111,103s, sent: 24089 msg/s  
id: test 1, time: 112,538s, sent: 13355 msg/s  
id: test 1, time: 113,750s, sent: 23792 msg/s  
id: test 1, time: 114,785s, sent: 16251 msg/s  
id: test 1, time: 116,241s, sent: 11552 msg/s  
id: test 1, time: 117,241s, sent: 21745 msg/s  
id: test 1, time: 118,259s, sent: 21125 msg/s

# Lazy Queues

Sample results of performance test lazy vs not-lazy



id: test 1, time: 48,154s, sent: 8955 msg/s  
id: test 1, time: 49,154s, sent: 37068 msg/s  
id: test 1, time: 50,192s, sent: 20587 msg/s  
id: test 1, time: 51,448s, sent: 20947 msg/s  
id: test 1, time: 52,945s, sent: 11235 msg/s  
id: test 1, time: 54,005s, sent: 6855 msg/s  
id: test 1, time: 55,334s, sent: 16273 msg/s  
id: test 1, time: 56,868s, sent: 28159 msg/s  
id: test 1, time: 58,041s, sent: 12345 msg/s  
id: test 1, time: 59,431s, sent: 19061 msg/s  
id: test 1, time: 60,431s, sent: 9038 msg/s  
id: test 1, time: 61,636s, sent: 18324 msg/s  
id: test 1, time: 62,924s, sent: 20570 msg/s  
id: test 1, time: 63,924s, sent: 28898 msg/s  
id: test 1, time: 65,165s, sent: 11522 msg/s  
id: test 1, time: 66,171s, sent: 19231 msg/s  
id: test 1, time: 67,407s, sent: 7778 msg/s  
id: test 1, time: 68,408s, sent: 38345 msg/s  
id: test 1, time: 69,408s, sent: 22440 msg/s  
id: test 1, time: 70,479s, sent: 1538 msg/s  
id: test 1, time: 72,149s, sent: 20142 msg/s  
id: test 1, time: 73,149s, sent: 14781 msg/s  
id: test 1, time: 74,721s, sent: 10466 msg/s  
id: test 1, time: 76,233s, sent: 19109 msg/s  
id: test 1, time: 77,462s, sent: 31286 msg/s  
id: test 1, time: 78,932s, sent: 14632 msg/s  
id: test 1, time: 80,374s, sent: 8292 msg/s  
id: test 1, time: 81,532s, sent: 18727 msg/s  
id: test 1, time: 82,776s, sent: 27184 msg/s  
id: test 1, time: 83,847s, sent: 13518 msg/s  
id: test 1, time: 85,497s, sent: 21698 msg/s

# Lazy Queues

```
Map<String, Object> args = new HashMap<String, Object>();
args.put("x-queue-mode", "lazy");
channel.queueDeclare("myqueue", false, false, false, args);
```

Overview    Connections    Channels    Exchanges    Queues    Admin

## Policies

User policies

Add / update a policy

Name: policy-4-lazy-queues \*

Pattern: ^lazy-queues\$ \*

Apply to: Queues

Priority:

Definition: queue-mode = lazy String String

Queues [All types] Max length | Max length bytes | Overflow behaviour ?  
Dead letter exchange | Dead letter routing key

Queues [Classic] HA mode ? | HA params ? | HA sync mode ?  
HA mirror promotion on shutdown ? | HA mirror promotion on failure ?  
Message TTL | Auto expire | Lazy mode | Master Locator

Queues [Quorum] Max in memory length ? | Max in memory bytes ? | Delivery limit ?

Exchanges Alternate exchange ?

Federation Federation upstream set ? | Federation upstream ?

Add / update pol.

```
> rabbitmqctl set_policy Lazy "^\lazy-queue$" "{\"queue-mode\":\"lazy\"}" --apply-to queues
```

# Lazy Queues

Overview				Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
lazy-queue	classic	D Args policy-4-lazy-queues	idle	0	0	0			
my_first_queue	classic	D Args	idle	4	0	4	0.00/s	0.00/s	0.00/s
sample4test	classic	Args	idle	4	0	4	0.00/s		

# Lazy Queues - test results

	Queue	Lazy Queue
Publication time	86,415 ms	72,366 ms
Node memory peak	2,4 GB	163 MB
Messages in memory	1 000 000	0
Queue process memory	642 MB	894 KB

Overview			De
Name	User name	State	SS
127.0.0.1:4797	guest	flow	

Queue	→	Messages	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
		Message body bytes	954 MiB	954 MiB	0 B	954 MiB	1,000,000	0
		Process memory	642 MiB					

Lazy Queue	→	Messages	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
		Message body bytes	954 MiB	954 MiB	0 B	0 B	1,000,000	0
		Process memory	894 kiB					

# Lazy Queues - summary

- **Memory over throughput**  
When memory is more important than throughput
- **Consumers are slow**  
Queues grow and consume more memory than needed
- **Queues to store tons of messages / DLX**  
Apply lazy mode on dead letter queues



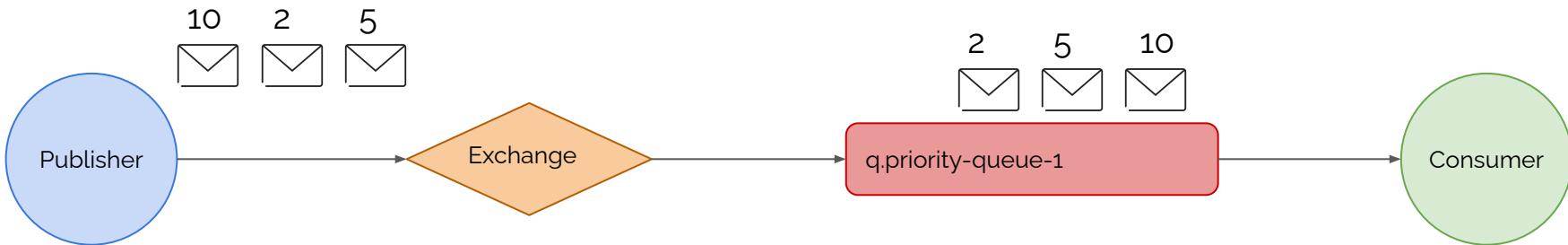
Lazy mode can be set in the moment of **queue creation** time or by using **policies**



Priority queues

# Priority queues

Queues act as FIFO, but some messages have a higher priority than others and need to be handled before other messages. Priority (**x-max-priority**) is an optional queue attribute set at queue creation time **only**. Priority range is between 1 and 255.





# Priority queues - test

Publication of 300 000 messages with 1KB payload each

1. 300 000 messages to regular queue
2. 300 000 messages with the same priority to the priority queue
3. 300 000 messages with different priorities to the priority queue



# Priority queues - test

Publication of 300 000 messages with 1KB payload each

	Queue				Priority Queue			
	Time	Node Memory (at the end)	Node Memory (max)	Process memory	Time	Node Memory (at the end)	Node Memory (max)	Process memory
All messages with the same priority	26 577 ms (26 sec)	621 MB	930 MB	181 MB	105 104 ms (1 min 45 sec)	595 MB	1.3 GB	183 MB
Messages with different priorities	n/a	n/a	n/a	n/a	146 864 ms (2 min 26 sec)	719 MB	2.2 GB	421 MB

# Priority queues - summary

- **Overall performance is lower**  
More CPU & RAM are needed; bigger latency, bigger load, lower throughput
- **Priority can be enabled only when queue is created**  
Priority feature cannot be applied via policies - RabbitMQ creates separate queue for every priority behind the scene. x-max-priority = 10 means 10 separate queues
- **Max priority is 255**  
Values between 1 and 10 are recommended
- **Order is not guaranteed**  
Messages with biggest Priority may not be consumed before lower ones - its suggestion only
- **Careful with TTL and max-length when using priority queues**  
Messages with TTL are not removed and messages with highest priority can be accidentally dropped



Windows Service

# Windows Service

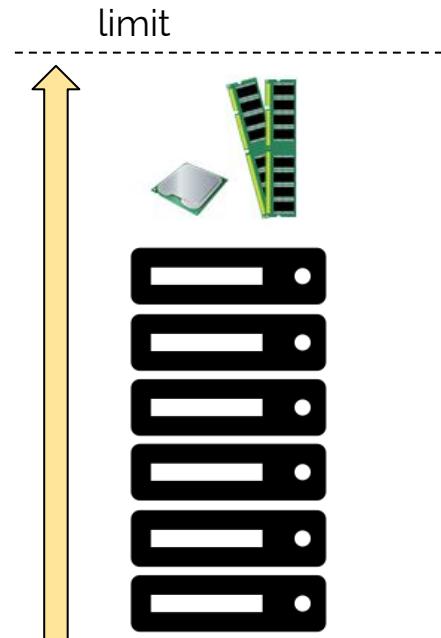
```
> set RABBITMQ_BASE=c:\RabbitMQ\data  
> set ERLANG_HOME=c:\Program Files\erl-24.1.7
```

Name	Location
RABBITMQ_BASE	%APPDATA%\RabbitMQ
RABBITMQ_CONFIG_FILE	%RABBITMQ_BASE%\rabbitmq
RABBITMQ_MNESIA_BASE	%RABBITMQ_BASE%\db
RABBITMQ_MNESIA_DIR	%RABBITMQ_MNESIA_BASE%\%RABBITMQ_NODENAME%-mnesia
RABBITMQ_LOG_BASE	%RABBITMQ_BASE%\log
RABBITMQ_LOGS	%RABBITMQ_LOG_BASE%\%RABBITMQ_NODENAME%.log
RABBITMQ_PLUGINS_DIR	<i>Installation-directory</i> /plugins
RABBITMQ_PLUGINS_EXPAND_DIR	%RABBITMQ_MNESIA_BASE%\%RABBITMQ_NODENAME%-plugins-expand
RABBITMQ_ENABLED_PLUGINS_FILE	%RABBITMQ_BASE%\enabled_plugins
RABBITMQ_PID_FILE	(Not currently supported)

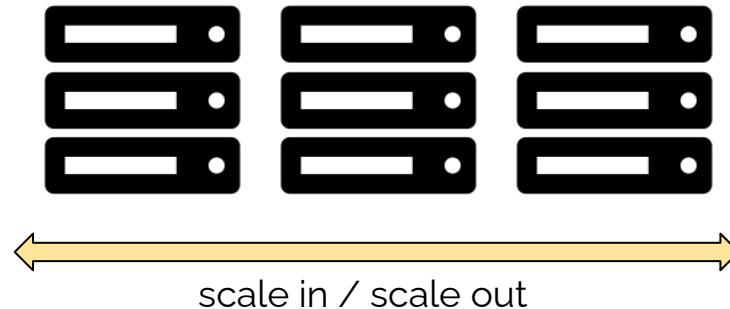


Distributed brokers

# Horizontal scaling vs Vertical scaling



1. **High availability**- system might perform in degraded state
2. **Fault-tolerant** is a higher bar. User is not expressing any impact on the fault, SLA is met
3. **Accept bigger load** - process more data faster



# Distributed RabbitMQ



## → **Clustering**

Forms a single logical broker; inter-node communication is transparent for consumers; reasonable network (LAN-like) latency; consumer to read for the queue even if not located on node it is connected to

## → **Federation**

Creates an upstream link to transmit messages. Federated exchange or Federated queue to receive messages published to an exchange or queue on upstream broker. Used to **replicate messages**, between local clusters or datacenters (across WAN)

## → **Shovel**

Similar to Federation; well-designed built-in consumer. Shovel consumes messages from cluster A and publishes them to cluster B (across WAN)



Increase data safety through replication; Support migration and cluster upgrades

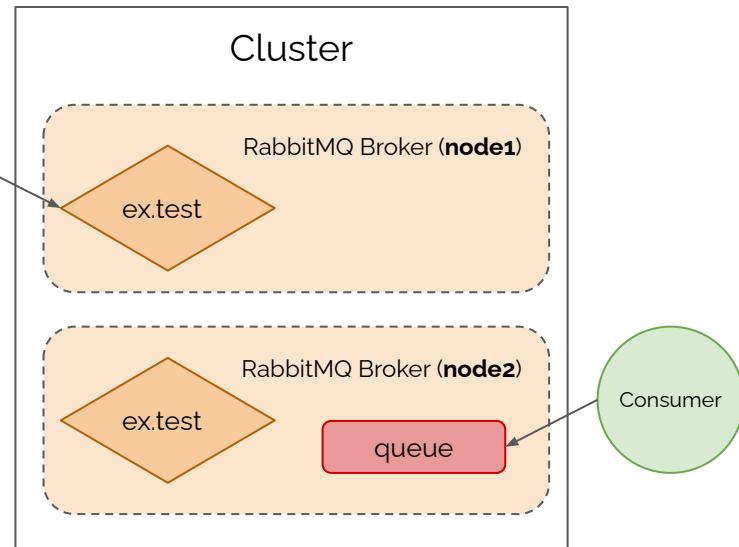


Clustering

# RabbitMQ cluster

Form a cluster to:

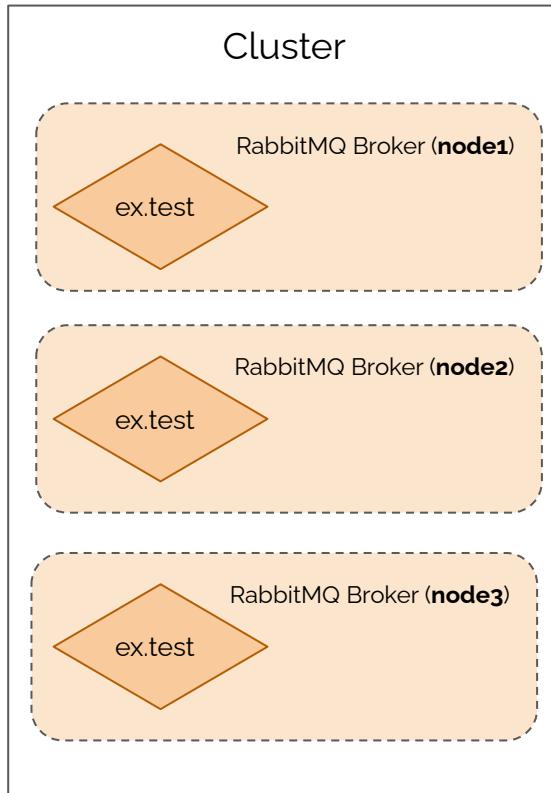
- Fault tolerance/high availability for client operations
- Increase performance and overall throughput
- Increase data safety



Always remember:

- Nodes in the same version (or compatible version)
- Decide on recovery mode (partitions)
- Choose number of nodes wisely (performance test, cluster reliability etc.)

# RabbitMQ cluster



Prefix

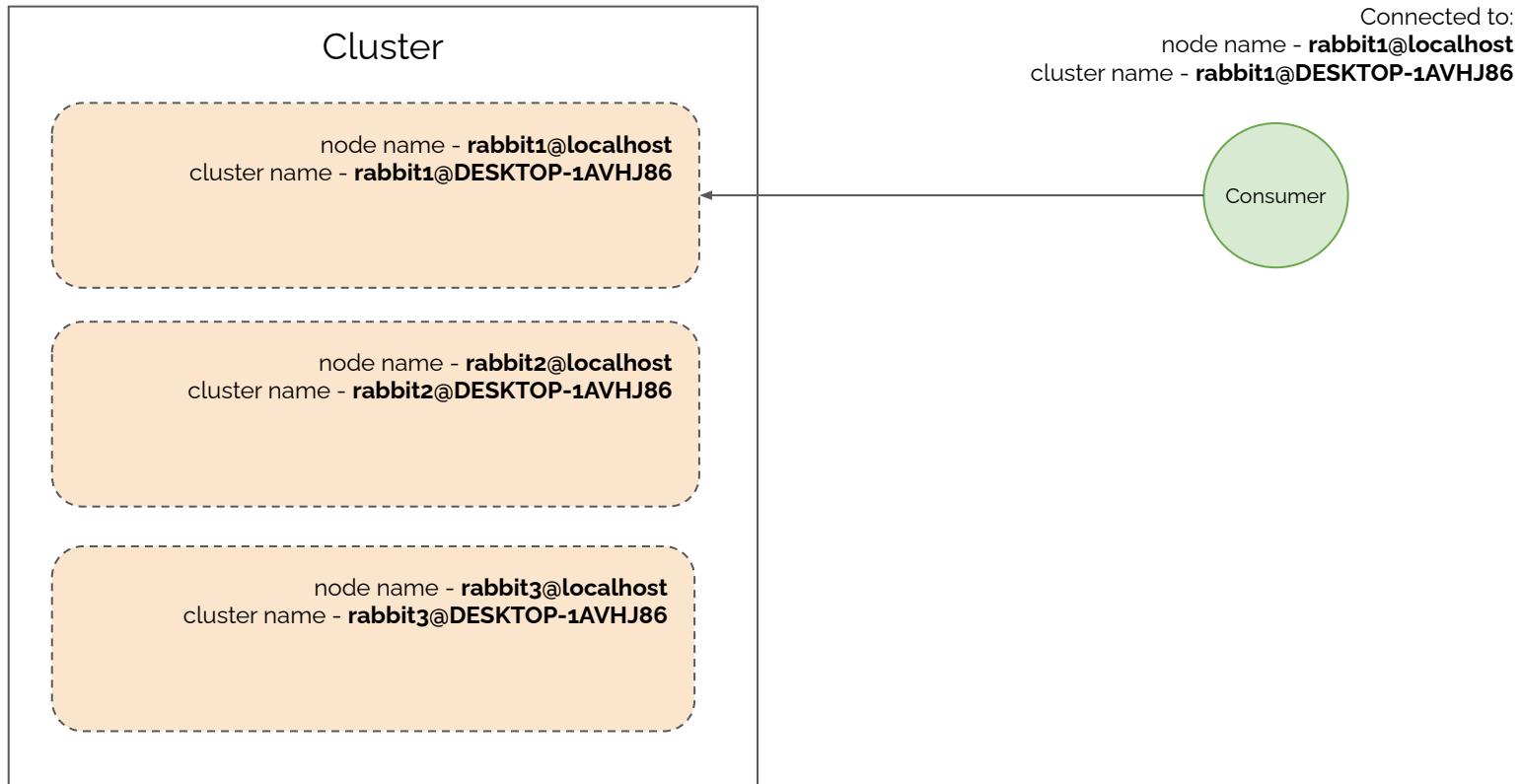
rabbit1@localhost

Hostname

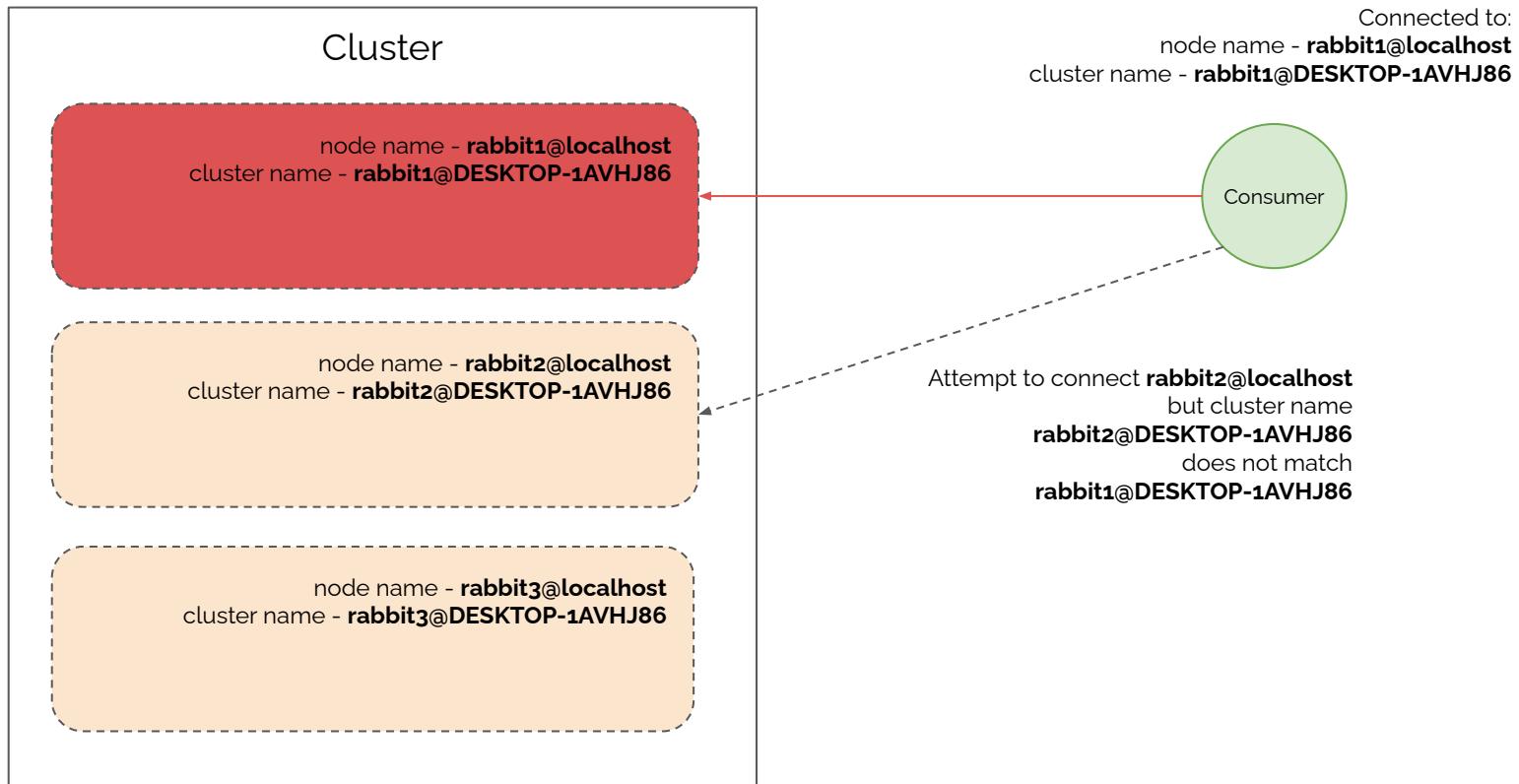
rabbit2@localhost

rabbit3@localhost

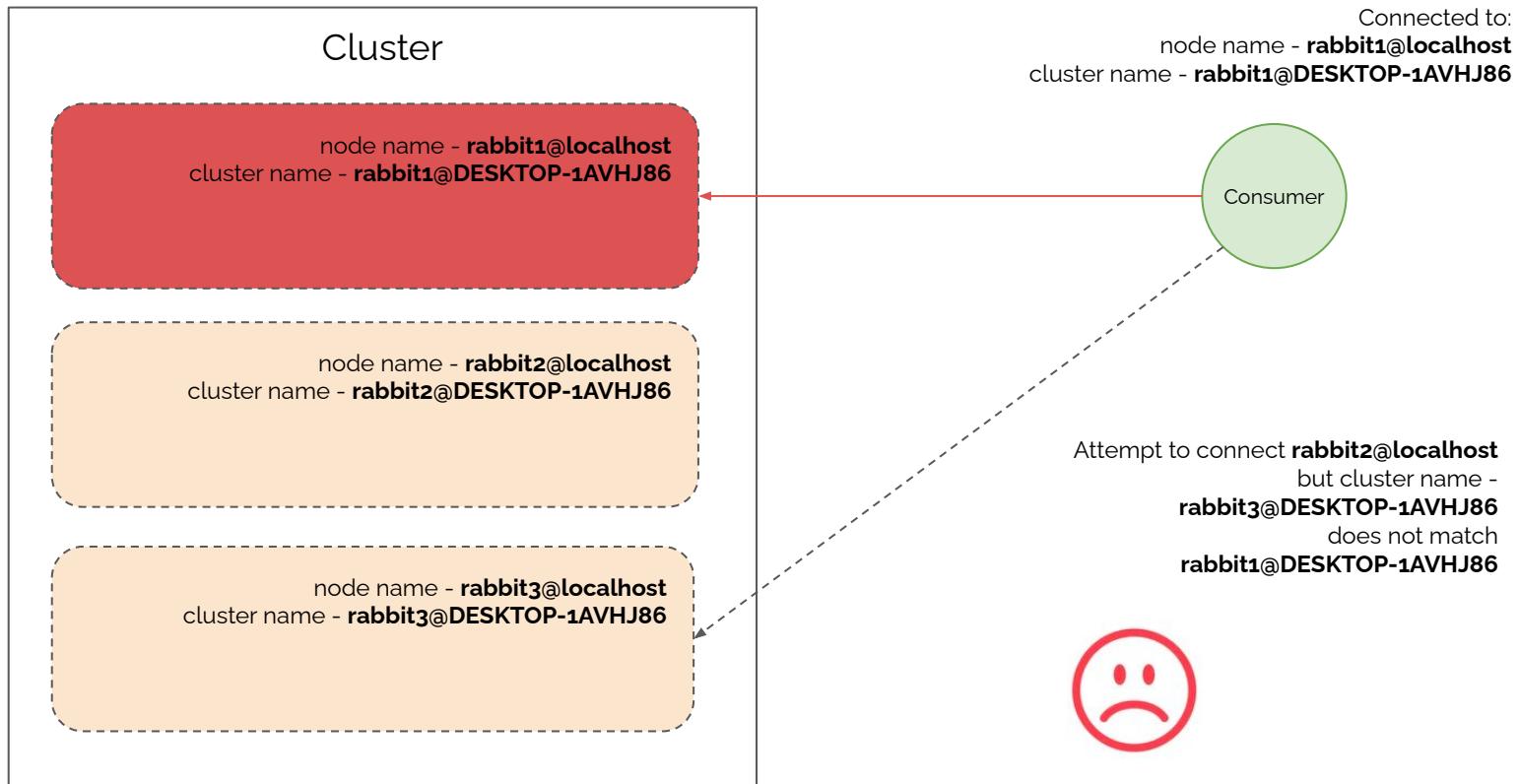
# RabbitMQ cluster



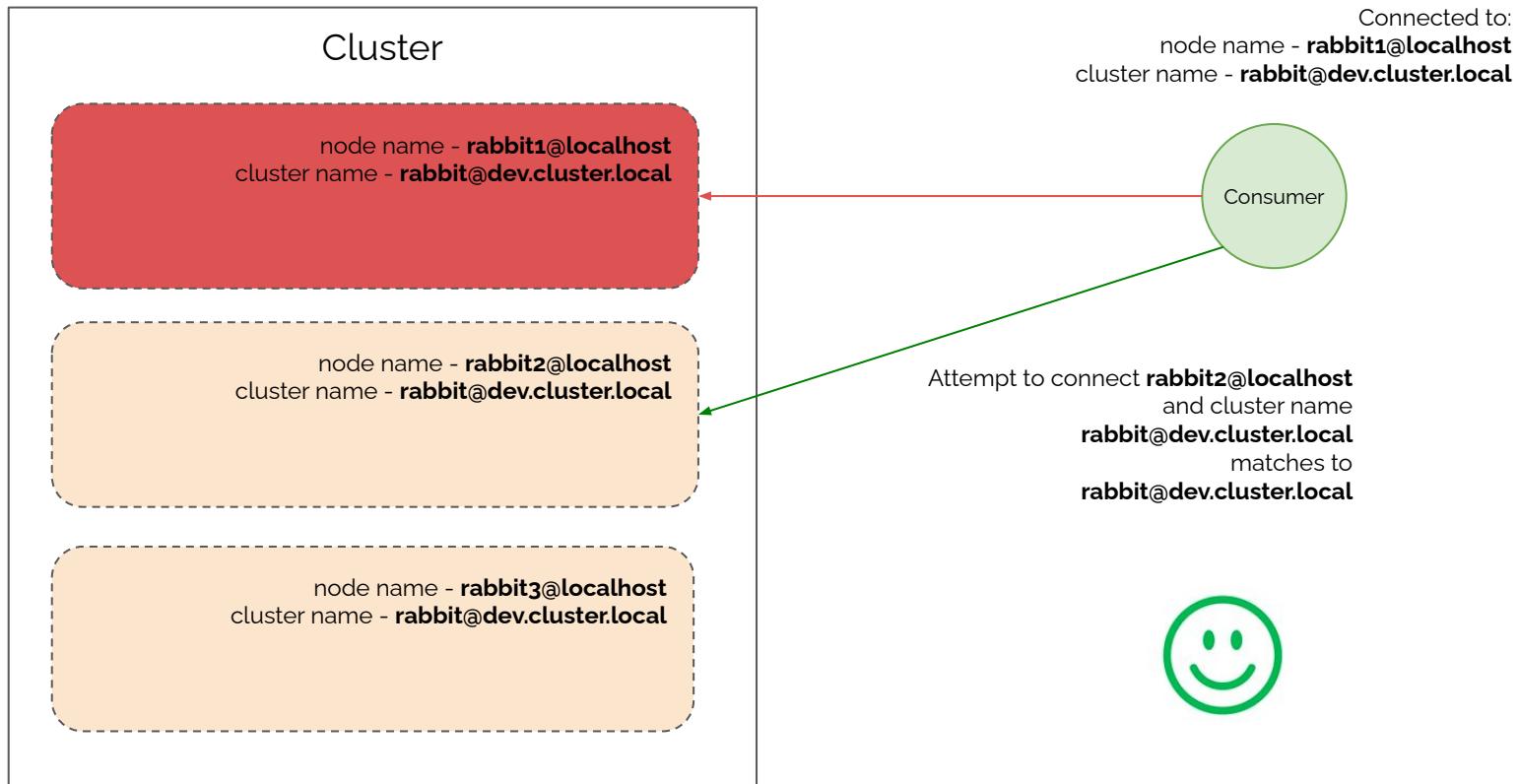
# RabbitMQ cluster



# RabbitMQ cluster



# RabbitMQ cluster



# RabbitMQ cluster - configuration

- Download example config file

<https://github.com/rabbitmq/rabbitmq-server/blob/masterdeps/rabbit/docs/rabbitmq.conf.example>

- Save as 2 separate files

C:\RabbitMQ\rabbitmq\_server-3.9.10\_node2\config\rabbitmq.conf  
C:\RabbitMQ\rabbitmq\_server-3.9.10\_node3\config\rabbitmq.conf

- Set the same cluster name in both

cluster\_name = dev.rabbitmq.cluster.local

```
cd c:\RabbitMQ\rabbitmq-server-windows-3.9.10\sbin
set HOMEDRIVE=C:
set HOMEPORT=\Users\%USERNAME%
set ERLANG_HOME=c:\Program Files\erl-24.1.7
set RABBITMQ_NODE_PORT=5672
set RABBITMQ_DIST_PORT=25672
set RABBITMQ_NODENAME=rabbit1@localhost
set RABBITMQ_MNESIA_BASE=C:\data\rabbit1
set RABBITMQ_MNESIA_DIR=C:\data\rabbit1\data
set RABBITMQ_LOG_BASE=C:\data\rabbit1\logs
REM Change rabbit1.conf; management.tcp.port = 15672
REM The Erlang runtime automatically appends the .conf extension to the value of this variable.
set RABBITMQ_CONFIG_FILE=C:\RabbitMQ\rabbitmq-server-windows-3.9.10\config\rabbitmq
set RABBITMQ_ENABLED_PLUGINS_FILE=C:\data\rabbit1\enabled_plugins

rabbitmq-server.bat
```

```
cd c:\RabbitMQ\rabbitmq_server-3.9.10_node2\sbin
set HOMEDRIVE=C:
set HOMEPORT=\Users\%USERNAME%
set ERLANG_HOME=c:\Program Files\erl-24.1.7
set RABBITMQ_NODE_PORT=5682
set RABBITMQ_DIST_PORT=25682
set RABBITMQ_NODENAME=rabbit2@localhost
set RABBITMQ_MNESIA_BASE=C:\data\rabbit2
set RABBITMQ_MNESIA_DIR=C:\data\rabbit2\data
set RABBITMQ_LOG_BASE=C:\data\rabbit2\logs
REM Change rabbit2.conf; management.tcp.port = 15682
REM The Erlang runtime automatically appends the .conf extension to the value of this variable.
set RABBITMQ_CONFIG_FILE=C:\RabbitMQ\rabbitmq_server-3.9.10_node2\config\rabbitmq
set RABBITMQ_ENABLED_PLUGINS_FILE=C:\data\rabbit2\enabled_plugins
```

rabbitmq-server.bat -detached

```
rabbitmqctl.bat -node rabbit2@localhost stop_app
rabbitmqctl.bat -node rabbit2@localhost join_cluster rabbit1@localhost
rabbitmqctl.bat -node rabbit2@localhost start_app
```

# RabbitMQ cluster

## Overview

### Totals

Queued messages [last minute](#) [?](#)



Ready: 7  
Unacked: 0  
Total: 7

Message rates [last minute](#) [?](#)



Publish: 0.00/s	Deliver (auto ack): 0.00/s	Get (manual ack): 0.00/s	Unroutable (return): 0.00/s
Publisher confirm: 0.00/s	Consumer ack: 0.00/s	Get (auto ack): 0.00/s	Unroutable (drop): 0.00/s
Deliver (manual ack): 0.00/s	Redelivered: 0.00/s	Get (empty): 0.00/s	Disk read: 0.00/s
			Disk write: 0.00/s

Global counts [?](#)

Connections: 0    Channels: 0    Exchanges: 11    Queues: 6    Consumers: 0

### Nodes

Name	File descriptors <a href="#">?</a>	Socket descriptors <a href="#">?</a>	Erlang processes	Memory <a href="#">?</a>	Disk space	Uptime	Info	Reset stats	+/-
rabbit1@localhost	0 32768 available	0 29401 available	490 1048576 available	123MiB 13GiB high watermark	42GiB 48MiB low watermark	1d 3h	basic disc 1 rss	This node All nodes	
rabbit2@localhost	0 65536 available	0 58893 available	476 1048576 available	115MiB 13GiB high watermark	42GiB 48MiB low watermark	0m 15s	basic disc 1 rss	This node All nodes	

# RabbitMQ cluster

RabbitMQ can form a cluster in many ways:

- Manually with **rabbitmqctl**

```
rabbitmqctl.bat --node rabbit2@localhost join_cluster rabbit1@localhost
```

- Declaratively by listing cluster nodes in **config file**

```
cluster_formations.peer_discovery_backend = classic_config
```

- Declaratively using **DNS-based discovery**

```
cluster_formations.peer_discovery_backend = dns
```

- Declaratively using **AWS (EC2) instance discovery** (via a plugin)

- Declaratively using **Kubernetes discovery** (via a plugin)

- Declaratively using **Consul-based discovery** (via a plugin)

- Declaratively using **etcd-based discovery** (via a plugin)

# RabbitMQ cluster

Classic example config:

```
cluster_formation.peer_discovery_backend = classic_config

cluster formation.classic config.nodes.1 = rabbit1@node1.example.local
cluster formation.classic config.nodes.2 = rabbit2@node2.example.local
cluster_formation.classic_config.nodes.3 = rabbit3@node3.example.local
```



Hostnames must be resolved using **DNS** or **local hosts** file:  
**/etc/hosts** on Unix  
**C:\Windows\System32\drivers\etc\hosts** on Windows

# RabbitMQ cluster

## DNS-based example config:

```
cluster_formation.peer_discovery_backend = dns  
  
cluster_formation.dns.hostname = discovery.example.local
```

Assume that `discovery.example.local` has 2 DNS A records (192.168.0.1 and 192.168.0.2). Reverse DNS lookups return **node1.example.local** and **node2.example.local** for those IPs. If current node's name is not set and defaults to `rabbit@$hostname`, then final list of nodes will be: **rabbit@node1.example.local** and **rabbit@node2.example.local**.

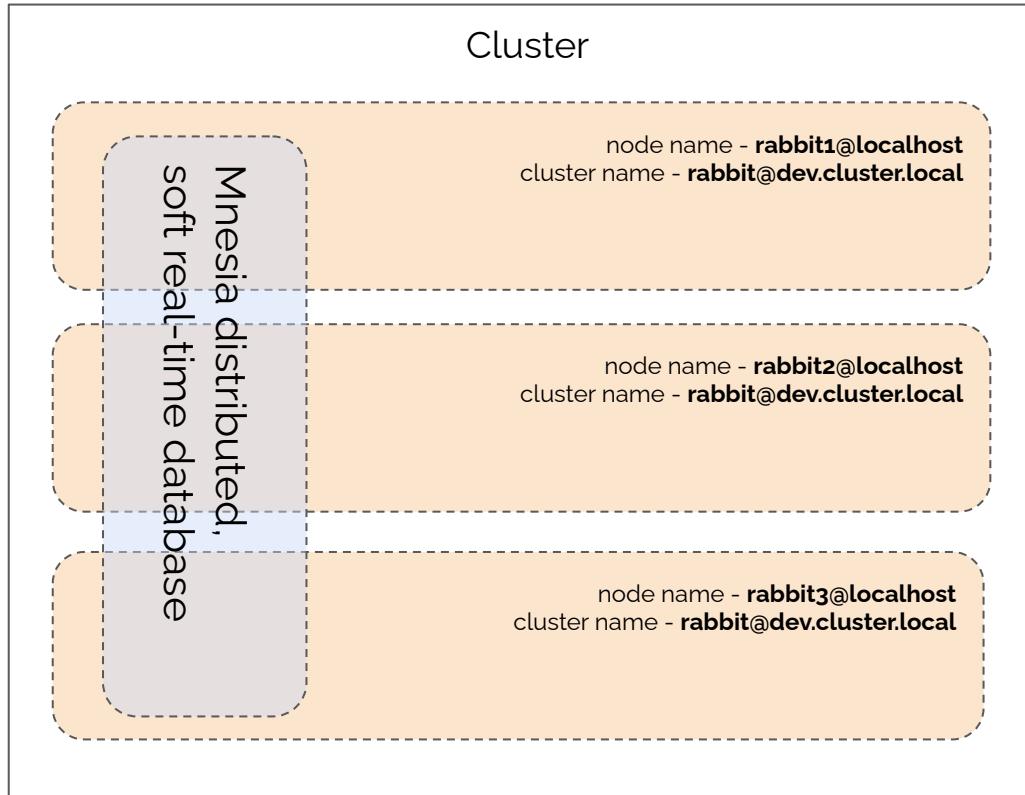
### DNS entries

node1.example.local	A	192.168.0.1
node2.example.local	A	192.168.0.2
discovery.example.local	A	192.168.0.1
discovery.example.local	A	192.168.0.2



Clustering - Scale In

# RabbitMQ cluster



# RabbitMQ remove node

 RabbitMQ™ 3.8.2 Erlang 22.2

Refreshed 2020-02-04 18:10:03 Refresh every 5 seconds ▾  
Virtual host All ▾  
Cluster dev3.eng.megacorp.local  
User guest Log out

**Overview** Connections Channels Exchanges Queues Admin

## Overview

▼ **Totals**

Queued messages last minute ?  
Currently idle  
Message rates last minute ?  
Currently idle  
Global counts ?

Connections: 0 Channels: 0 Exchanges: 7 Queues: 0 Consumers: 0

▼ **Nodes**

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	+/-
rabbit1@localhost	Node not running								
rabbit2@localhost	0 65536 available	0 58893 available	443 1048576 available	95MiB 6.4GiB high watermark	10GiB 48MiB low watermark	59m 19s	basic disc 1 rss	This node All nodes	

▶ Churn statistics  
▶ Ports and contexts  
▶ Export definitions  
▶ Import definitions

---

[HTTP API](#) [Server Docs](#) [Tutorials](#) [Community Support](#) [Community Slack](#) [Commercial Support](#) [Plugins](#) [GitHub](#) [Changelog](#)

# RabbitMQ remove stopped node

Must be done explicitly using a **rabbitmqctl** command. How to remove **stopped** `rabbit3@localhost`:

From the console of `rabbit2@localhost` or `rabbit1@localhost` or

```
> rabbitmqctl cluster_status --formatter json  
> rabbitmqctl forget_cluster_node rabbit3@localhost
```

# RabbitMQ remove stopped node

Must be done explicitly using a **rabbitmqctl** command. How to remove **stopped** `rabbit1@localhost`:

On the `rabbit2@localhost`

```
> rabbitmqctl cluster_status --formatter json  
> rabbitmqctl forget_cluster_node rabbit1@localhost
```

Try to run `rabbit1@localhost`

```
BOOT FAILED
=====
Error description:
  init:do_boot/3 line 817
  init:start_em/1 line 1109
  rabbit:start_it/1 line 474
  rabbit:'-boot/0-fun-0-'/0 line 327
  rabbit_mnesia:check_cluster_consistency/0 line 702
throw:{error,{inconsistent_cluster,"Node rabbit1@localhost thinks it's clustered with node rabbit2@localhost, but rabbit2@localhost disagrees"}}
Log file(s) (may contain more information):
  C:/tmp/rabbit1/logs/rabbit1@localhost.log
  C:/tmp/rabbit1/logs/rabbit1@localhost_upgrade.log

{"init terminating in do_boot",{error,{inconsistent_cluster,"Node rabbit1@localhost thinks it's clustered with node rabbit2@localhost, but rabbit2@localhost disagrees"}}}
init terminating in do_boot ({error,{inconsistent_cluster,Node rabbit1@localhost thinks it's clustered with node rabbit2@localhost, but rabbit2@localhost disagrees}})

Crash dump is being written to: C:\tmp\rabbit1\logs\erl_crash.dump...done
```

# RabbitMQ remove running node

Must be done explicitly using a **rabbitmqctl** command. How to remove **running** `rabbit2@localhost`

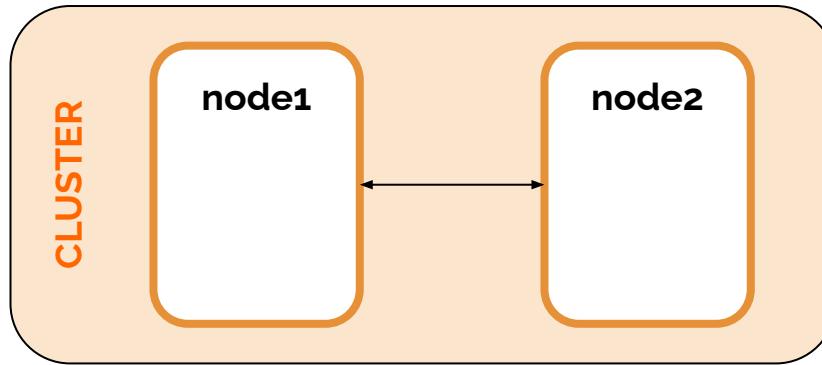
From the console of the `rabbit1@localhost` or `rabbit2@localhost`

```
> rabbitmqctl cluster_status --formatter json  
  
> rabbitmqctl --node rabbit2@localhost stop_app  
> rabbitmqctl --node rabbit2@localhost reset  
> rabbitmqctl --node rabbit2@localhost start_app
```



## Partitions

# RabbitMQ Cluster - Partitions



Node waits 6s (by default) for its peer. After 6s, both having thought the other is down and they **start acting separately**. Queues, bindings, exchanges can be created or deleted separately!

Situation is a classic **split-brain effect** and split-brain effect causes **network partitions**. RabbitMQ cluster requires fast and reliable network. Network connection failures between nodes have an effect on data consistency and availability to client operations (as in the CAP theorem). RabbitMQ supports different partition handling strategies. Depends on chosen strategy RabbitMQ can ignore them waiting for human action or try to fix them automatically.

Run **rabbitmq-diagnostics cluster\_status**, use **RESTful API** or **logs** to check for active partitions (no active partitions = everything is fine)

# RabbitMQ Partitions - manual recovery

Manual recovery is a default behavior. To recover from split-brain in manual mode, follow steps:

Option A)

- **Select one partition you trust the most.**

This partition will become the authority for the state of the system (schema, messages); other partitions will be lost.

- **Stop all nodes in the other partitions, then start them again.**

Once they rejoin the cluster, their state will be restored from the trusted partition.

- **Restart node in the trusted partition**

Just to clear the warning.

Option B)

- **Stop entire cluster**

- **Start node in trusted partition**

- **Start rest of nodes**

# RabbitMQ Partitions - auto recovery

Auto recovery doesn't eliminate the problem, just makes it transparent/less problematic sometimes.

- **ignore (default)**

Means - manual recovery mode

- **pause-minority**

Chooses partition tolerance over availability from the CAP theorem by pausing minority nodes  
(most the nodes in a single partition will continue to run)

- **pause-if-all-down**

Similar to pause-minority mode, but allows admin to decide (i.e. pause all nodes on rackB  
when network between rackA and rackB is lost)

- **autoheal**

RabbitMQ to automatically select trusted partition and restart nodes



Use settings prefixed by `cluster_partition_handling` in config file to  
apply partitions strategy

# RabbitMQ Partitions - auto recovery

When to use auto recovery

- **ignore (default)**  
network really is reliable, small cluster
- **pause-minority / pause-if-all-down**  
network is less reliable; cluster through 3 datacenters, 1 DC can be down at time, remaining 2 to continue providing service
- **autoheal**  
network may not be reliable; focus on providing service over data integrity; can be two nodes cluster



Use settings prefixed by `cluster_partition_handling` in config file to apply partitions strategy

# RabbitMQ Partitions - summary

- **Monitor partitions**

Solve partitions quickly to avoid growing data inconsistency

- **Odd number of nodes and pause\_minority**

Use the pause\_minority strategy with an odd number of nodes (3, 5, 7, and so on) if you are not sure for which element from the CAP theorem your cluster should be prepared

- **Autoheal**

network may not be reliable; focus on providing service over data integrity; can be two nodes cluster



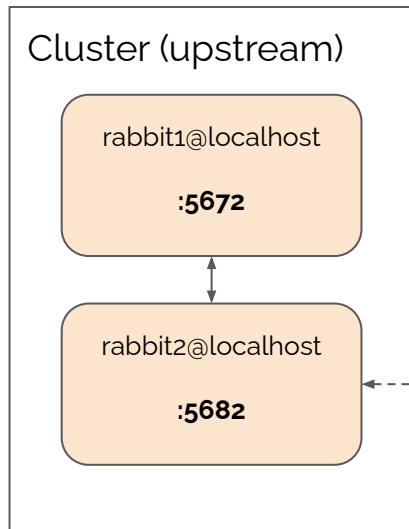
Use settings prefixed by `cluster_partition_handling` in config file to apply partitions strategy



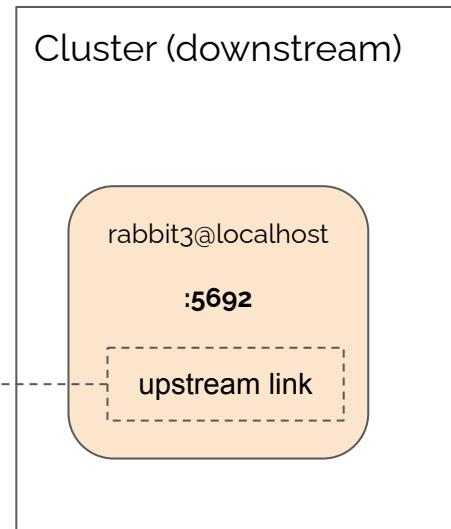
Federation & Shovel

# RabbitMQ federation - configuration

*source*



*target*



- From upstream to downstream
- Many downstreams
- Downstream shares details with upstream
- Downstream initiates the connection

# RabbitMQ federation

## → Federated Exchange

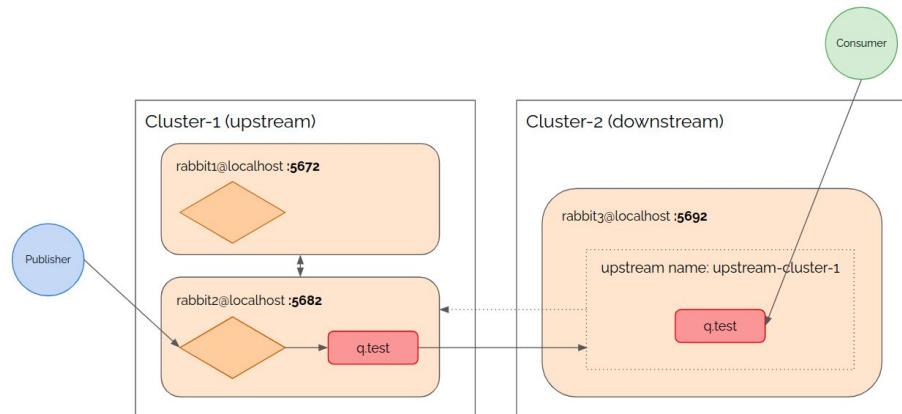
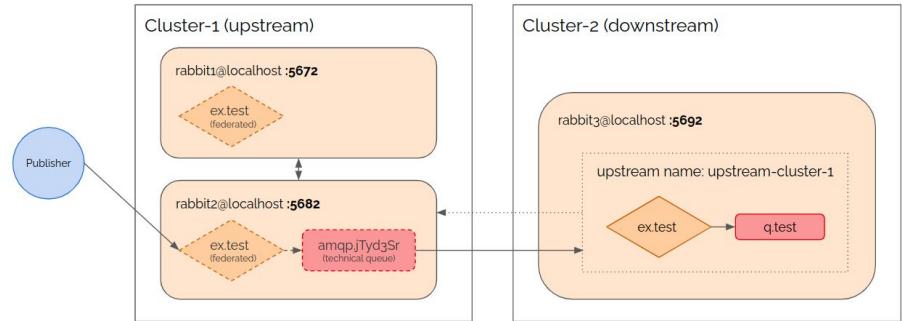
Exchange federates to one or more other exchanges. An Exchange on the downstream (destination) node receives a copy of messages that are published to the upstream node.

## → Federated Queue

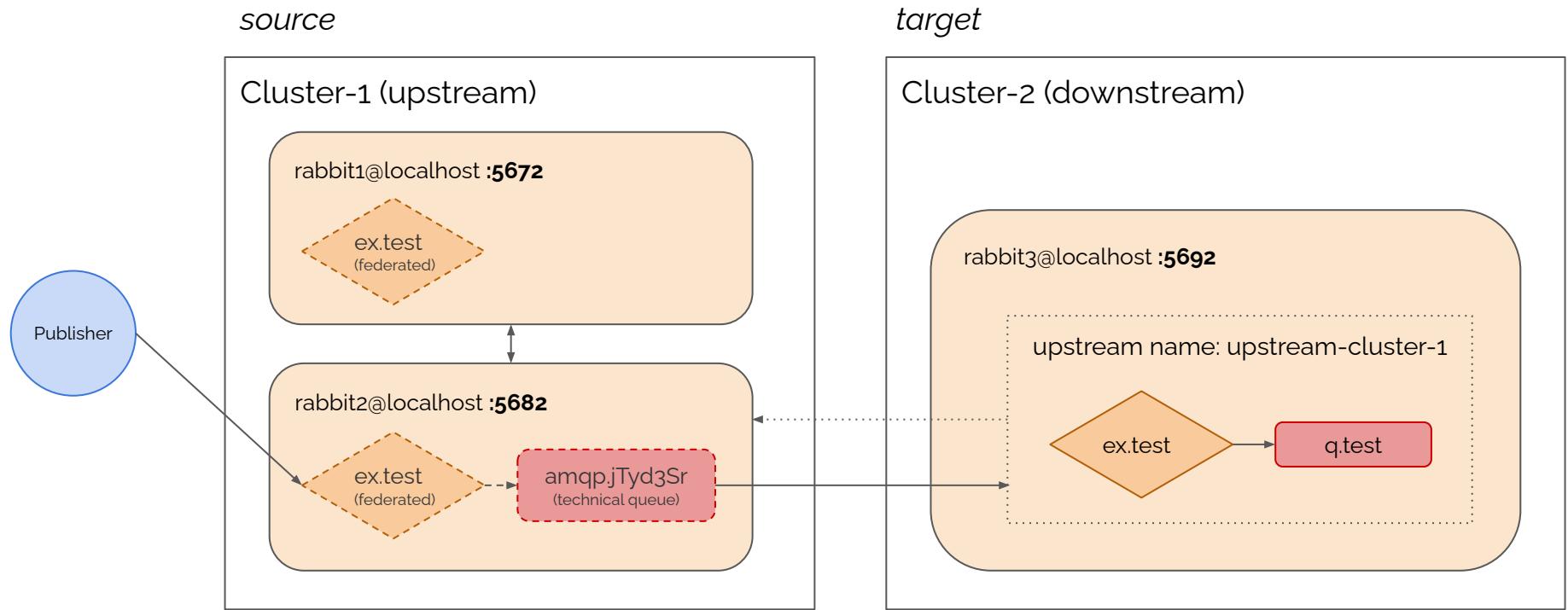
Consumers can be connected to the queue on both the upstream(source) and downstream(destination) nodes, Any messages consumed by the downstream (federated) queue will not be available for consumers on the upstream queue



Federated Queues are introduced in RabbitMQ 3.2.x



# Federated exchange



# Federated exchange - summary

Federated exchanges can be used to transmit subset of messages to remote locations or can be used to implement massive fanout with a single "source" exchange in one cluster.

- **Exchange to exchange model**

Federated exchange represents concepts of two changes bound together, but between clusters connected by WAN

- **Don't federate all exchanges**

Federate exchanges that are needed by the architecture of your system

- **Downstream propagates bindings to the upstream**

- **Subset of messages can be transmitted to the downstream**

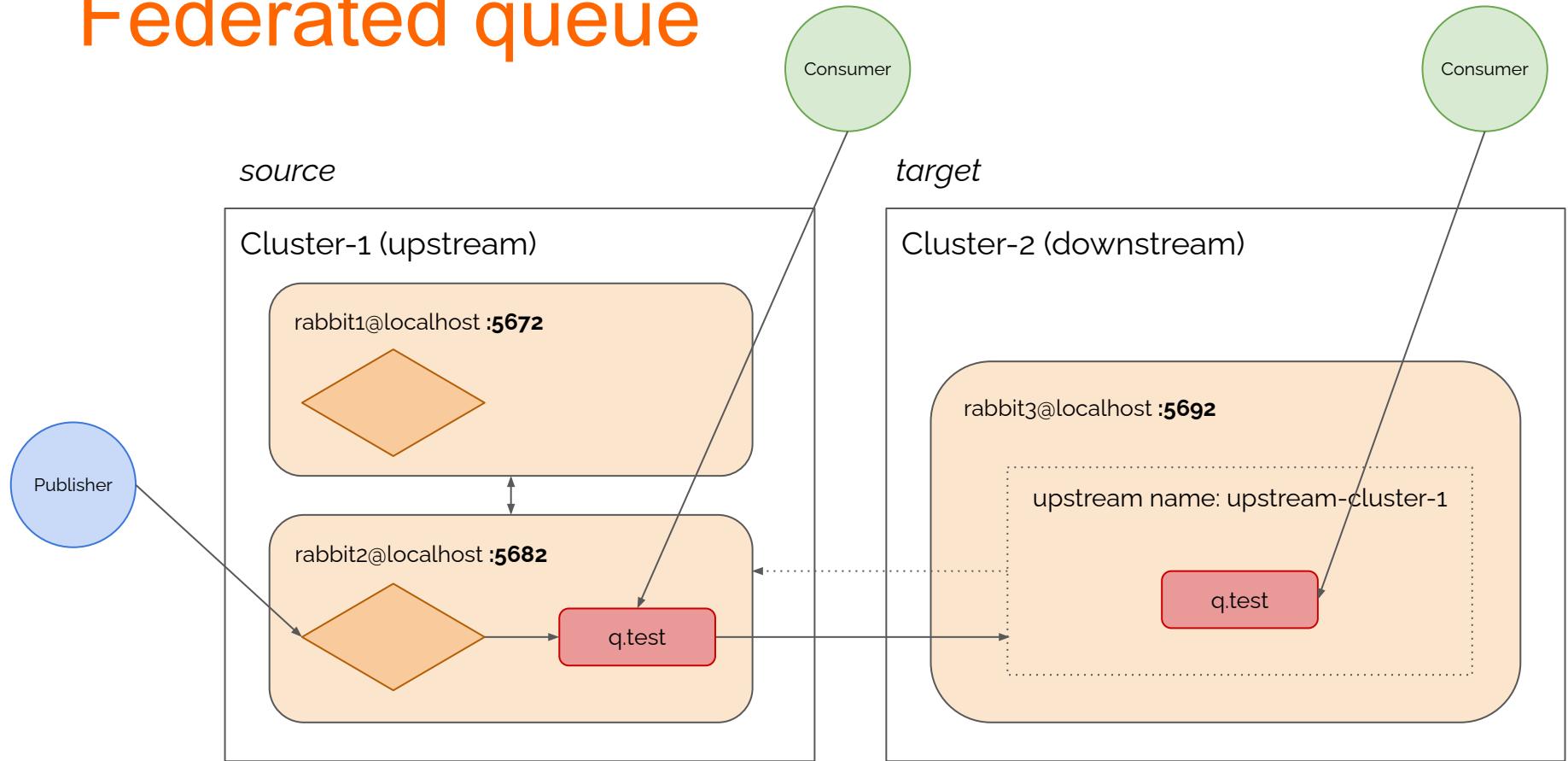
Shovel is a well written publisher/consumer. Federations with combination of topic, or direct exchanges and custom routing keys allow to design complex messages routing

- **Only for build-in exchange types**

Custom exchanges installed as plugins may not work properly

- **Default exchange and internal exchanges cannot be federated**

# Federated queue



# Federated queue - summary

Federated queues can be used to balance the load of a single logical queue across nodes or clusters

- **Upstream queue don't need to be reconfigured**

Federation assumes that other queue is located on a separated node or in a separate cluster connected by WAN

- **Messages are transmitted when consumer is connected**

When downstream queue has no consumers, messages are not transmitted

- **Pooling of messages is not supported**

Consumers that use basic.get cannot consume messages over federation, because it doesn't fit to federation's availability and partition tolerance from the CAP theorem

- **Downstream queue can be empty**

Attempt to consume causes messages transmission from the upstream

- **Queue federation doesn't propagate bindings**

- **Prefer local consumers than remote ones**

Consumer priorities introduced in RabbitMQ 3.2.0 is used for that

# RabbitMQ federation - configuration

Install federation plugins

```
> rabbitmq-plugins enable rabbitmq_federation  
> rabbitmq-plugins enable rabbitmq_federation_management
```

The screenshot shows the RabbitMQ Management Console interface. At the top, there is a navigation bar with tabs: Overview, Connections, Channels, Exchanges, Queues, Admin (which is currently selected), and User guest Log out. Below the navigation bar, the main content area has a title "Federation Upstreams". Under this title, there is a section titled "Upstreams" with a sub-section "Add a new upstream". There is also a link "URI examples". To the right of the main content area, there is a sidebar with several links: Users, Virtual Hosts, Feature Flags, Policies, Limits, Cluster, Federation Status (which is highlighted with an orange background and a cursor icon), and Federation Upstreams.

Cluster: `devs.eng.megacorp.local`

User: guest [Log out](#)

Overview    Connections    Channels    Exchanges    Queues    Admin

Federation Upstreams

Upstreams

... no upstreams ...

Add a new upstream

URI examples

Users

Virtual Hosts

Feature Flags

Policies

Limits

Cluster

Federation Status

Federation Upstreams

HTTP API    Server Docs    Tutorials    Community Support    Community Slack    Commercial Support    Plugins    GitHub    Changelog

# RabbitMQ federation - configuration

- 1 Create rabbit3@localhost listening on port 5692.
- 2 Define an upstream to rabbit2@localhost.

*We defined the upstream but there is no "link" yet.  
Alternatively use command below or RESTful API*

```
> rabbitmqctl set_parameter federation-upstream my-upstream ^
  "{\"uri\":\"amqp://localhost:5692\",\"expires\":3600000}"
```

```
PUT /api/parameters/federation-upstream/%2f/my-upstream
{"value":{"uri":"amqp://localhost:5692","expires":3600000}}
```

Add a new upstream

General parameters

Name:	my-upstream
URI:	amqp://localhost:5682
Prefetch count:	
Reconnect delay:	?
Acknowledgement Mode:	On confirm
Trust User-ID:	No

Federated exchanges parameters

Exchange:	
Max hops:	
Expires:	ms
Message TTL:	ms
HA Policy:	

Federated queues parameter

Queue:	
Consumer tag:	

**Add upstream**

## Federation Upstreams

### Upstreams

Name	URI	Prefetch Count	Reconnect Delay	Ack mode	Trust User-ID	Exchange	Max Hops	Expiry	Message TTL	HA Policy	Queue	Consumer tag
my-upstream	amqp://localhost:5672			on-confirm	?	?			?	?		

**Add a new upstream**

# RabbitMQ federation - configuration

3

On rabbit3@localhost create an exchange "ex.test"

4

On rabbit3@localhost create policy to federate "ex.test" exchange (via command line or RESTful API). We should see running upstream link on rabbit3@localhost

```
> rabbitmqctl set_policy --apply-to exchanges federate-ex "^ex\." ^
  "{\"federation-upstream-set\":\"all\"}"
```

```
> rabbitmqctl set_policy --apply-to exchanges federate-ex "^ex\." ^
  "{\"federation-upstream\":\"my-upstream\"}"
```

```
> rabbitmqctl set_policy --apply-to queues federate-queue "^q\." ^
  "{\"federation-upstream-set\":\"all\"}"
```

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

## Federation Status

### Running Links

Upstream	URI	Exchange / Queue	State	Inbound message rate	Last changed	ID	Consumer tag	Operations
my-upstream	amqp://localhost:5672	ex.test exchange	running		2021-04-26 23:54:06	bf6017c1		<button>Restart</button>

# RabbitMQ federation - configuration

On rabbit2@localhost we see federated exchange

Queues

All queues (11)

Pagination

Page 1 of 1 - Filter:   Regex ?

Overview				Messages			Message rates			+/-	
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
/	federation: ex.test -> rabbit3@clustername	classic	D Epx Args	idle	0	0	0				
/	q.hash1	classic	D Args	idle	8	0	8	0.00/s			
/	q.hash2	classic	D Args	idle	0	0	0				
/	q.messages	classic	D Args	idle	3	0	3				

## Exchanges

All exchanges (20, filtered down to 4)

Pagination

Page 1 of 1 - Filter:   Regex ?

Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
/	ex.hash	x-consistent-hash	D	0.00/s	0.00/s	
/	ex.messages	direct	D			
/	ex.test	direct	D			
/	federation: ex.test -> rabbit3@clustername B	x-federation-upstream	D AD I Args			

# RabbitMQ federation - testing

localhost:15682/#/exchanges

RabbitMQ™ RabbitMQ 3.8.8 Erlang 23.0.4

Refreshed 2020-09-14 21:19:

Overview Connections Channels **Exchanges** Queues Admin

Exchanges

All exchanges (9)

Pagination

Page 1 of 1 - Filter:   Regex ?

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D	0.00/s	0.00/s	
amq.direct	direct	D			
amq.fanout	fanout	D			
amq.headers	headers	D			
amq.match	headers	D			
amq.rabbitmq.trace	topic	D I			
amq.topic	topic	D			
ex.test	direct	D			

federation: ex.test -> rabbit3@localhost B x-federation-upstream D AD I Args

Send message to federated exchange on rabbit2@localhost and observe queue on rabbit3@localhost

localhost:15692/#/federation

RabbitMQ™ RabbitMQ 3.8.8 Erlang 23.0.4

Overview Connections Channels Exchanges Queues **Admin**

Federation Status

Running Links

Upstream	URI	Exchange / Queue	State	Inbound message rate	Last changed	ID	Consumer tag	Operations
test	amqp://localhost:5682	ex.test exchange	running	0.00/s	2020-09-14 21:15:21	a66351fb		<span style="border: 1px solid #ccc; padding: 2px;">Restart</span>

EXERCISE

# RabbitMQ federation - testing

Overview    Connections    Channels    **Exchanges**    Queues    Admin

---

## Exchanges

▼ All exchanges (12)

Pagination

Page **1** of 1 - Filter:   Regex [?](#)

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D			
amq.direct	direct	D federate-me			
amq.fanout	fanout	D federate-me			
amq.headers	headers	D federate-me			
amq.match	headers	D federate-me			
amq.rabbitmq.trace	topic	D I federate-me			
amq.topic	topic	D federate-me			
federation: amq.direct -> dev3.eng.megacorp.local B	x-federation-upstream	D AD I Args			
federation: amq.fanout -> dev3.eng.megacorp.local B	x-federation-upstream	D AD I Args			
federation: amq.headers -> dev3.eng.megacorp.local B	x-federation-upstream	D AD I Args			
federation: amq.match -> dev3.eng.megacorp.local B	x-federation-upstream	D AD I Args			
federation: amq.topic -> dev3.eng.megacorp.local B	x-federation-upstream	D AD I Args			

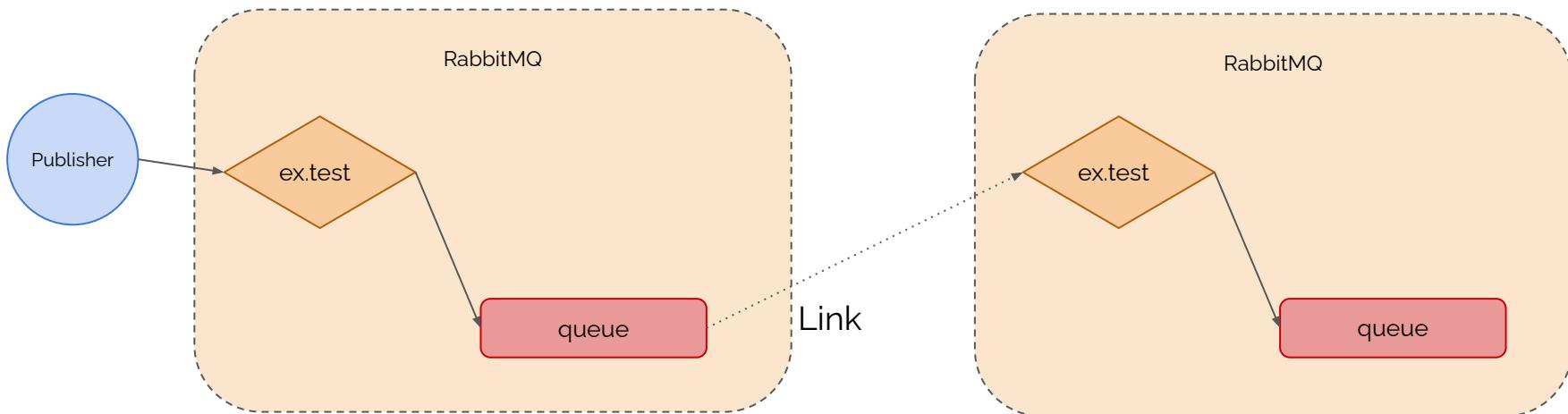
▶ Add a new exchange

# RabbitMQ shovel



Shovels support AMQP protocol only

Shovel is more flexible than Federation. It's well written RabbitMQ consumer.



# RabbitMQ shovel - configuration

Install shovel plugin

```
> rabbitmq-plugins enable rabbitmq_shovel  
> rabbitmq-plugins enable rabbitmq_shovel_management
```

The screenshot shows the RabbitMQ Admin interface. The top navigation bar includes links for Overview, Connections, Channels, Exchanges, Queues, and Admin. The Admin link is currently selected and highlighted in dark grey. To the right of the Admin link, the cluster name is listed as "Cluster dev3.eng.megacorp.local" and the user is identified as "User guest". A "Log out" button is also present.

The main content area displays the "Dynamic Shovels" page. It features a heading "Dynamic Shovels" and a sub-section "Shovels" with a note "... no shovels ...". Below this, there are two links: "Add a new shovel" and "URI examples".

On the right side, there is a sidebar with several links: "Users", "Virtual Hosts", "Feature Flags", "Policies", "Limits", "Cluster", "Shovel Status" (which is highlighted with a red background and a cursor icon), and "Shovel Management". At the bottom of the sidebar, there are links for "Federation Status" and "Federation Upstreams".

At the very bottom of the interface, there is a footer navigation bar with links for "HTTP API", "Server Docs", "Tutorials", "Community Support", "Community Slack", "Commercial Support", "Plugins", "GitHub", and "Changelog".

# RabbitMQ shovel - configuration

Create shovel on cluster rabbit1@localhost and rabbit2@localhost.

Overview    Connections    Channels    Exchanges    Queues    **Admin**

### Dynamic Shovels

▼ Shovels

... no shovels ...

▼ Add a new shovel

Name: my-shovel \*

Source: AMQP 0.9.1

URI: ? Queue: ?  
amqp://localhost:5672 \* test-ha2

Prefetch count: ?

Auto-delete ? Never

Destination: AMQP 0.9.1

URI ? Queue: ?  
amqp://localhost:5692 \* test-ha2

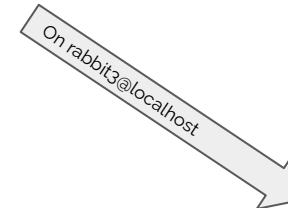
Add forwarding headers: ? No

Reconnect delay: ? s

Acknowledgement mode: ? On confirm

**Add shovel**

▶ URI examples



Overview				Messages			Message rates		
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
test	classic	D Args	idle	0	0	0	0.00/s	0.00/s	0.00/s
test-ha2	classic	D	idle	1	0	1	0.00/s	0.00/s	0.00/s

# RabbitMQ shovel - configuration

Static Shovels	Dynamic Shovels
Defined in the broker advanced configuration file (advanced.config)	Defined using the broker's runtime parameters (i.e. using RabbitMQ Management UI)
Require a restart of the hosting broker to change.	Can be created and deleted at any time.
Queues, exchanges or bindings need to be declared manually at startup	Slightly more opinionated: the queues, exchanges and bindings used by the shovel will be declared automatically.
Must be defined in every node in the cluster	Shovel definition is replicated between nodes



Shovel is running on a single node in the cluster only

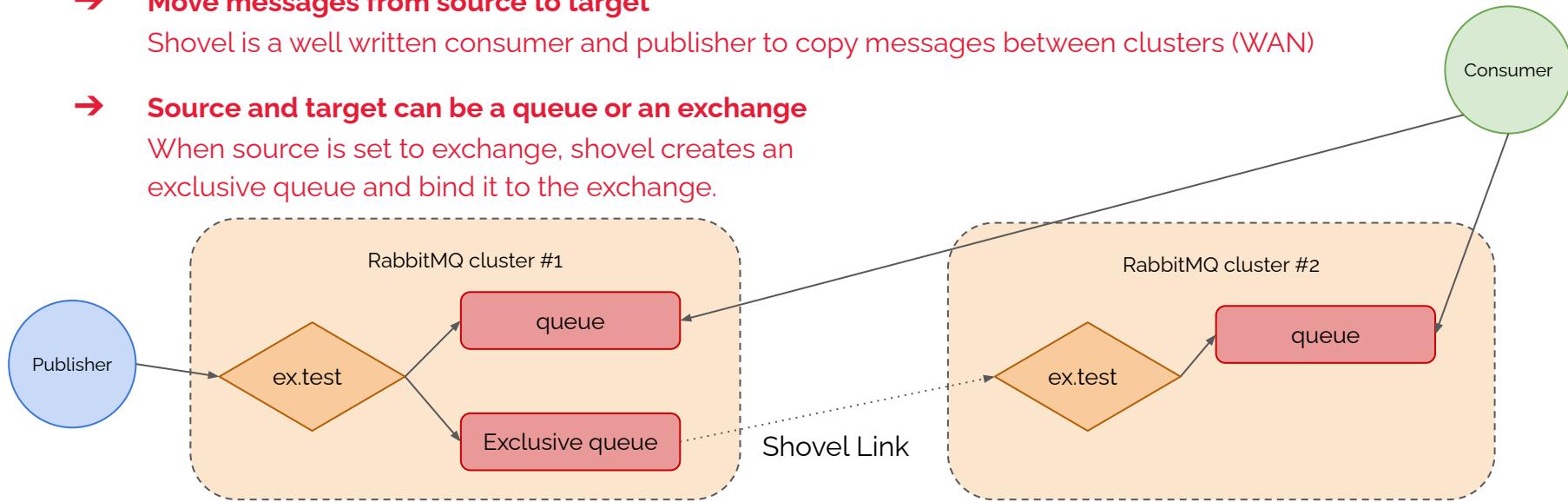
# RabbitMQ shovel - summary

- **Move messages from source to target**

Shovel is a well written consumer and publisher to copy messages between clusters (WAN)

- **Source and target can be a queue or an exchange**

When source is set to exchange, shovel creates an exclusive queue and bind it to the exchange.



- **Dynamic or Static**

Both do the same, declaration is different (runtime vs configuration file)

# Distributed RabbitMQ - Summary

Federation / Shovel	Clustering
Brokers are logically separate and may have different owners.	A cluster forms a single logical broker.
Brokers can run different (and incompatible in certain ways) versions of RabbitMQ and Erlang.	Nodes must run compatible versions RabbitMQ and Erlang.
Brokers can be connected via WAN.	Brokers must be connected via reasonably reliable LAN.
Brokers can be connected in whatever topology you arrange. Links can be one-way or two-way.	All nodes connect to all other nodes in both directions.
Emphasizes Availability and Partition Tolerance (AP) from the CAP theorem.	Emphasizes Consistency and Partition Tolerance (CP) from the CAP theorem.
Some exchanges in a broker may be federated while some may be local.	Clustering is all-or-nothing.
A client connecting to any broker can only use non-exclusive queues in that broker.	A client connecting to any node can use non-exclusive queues on all nodes.



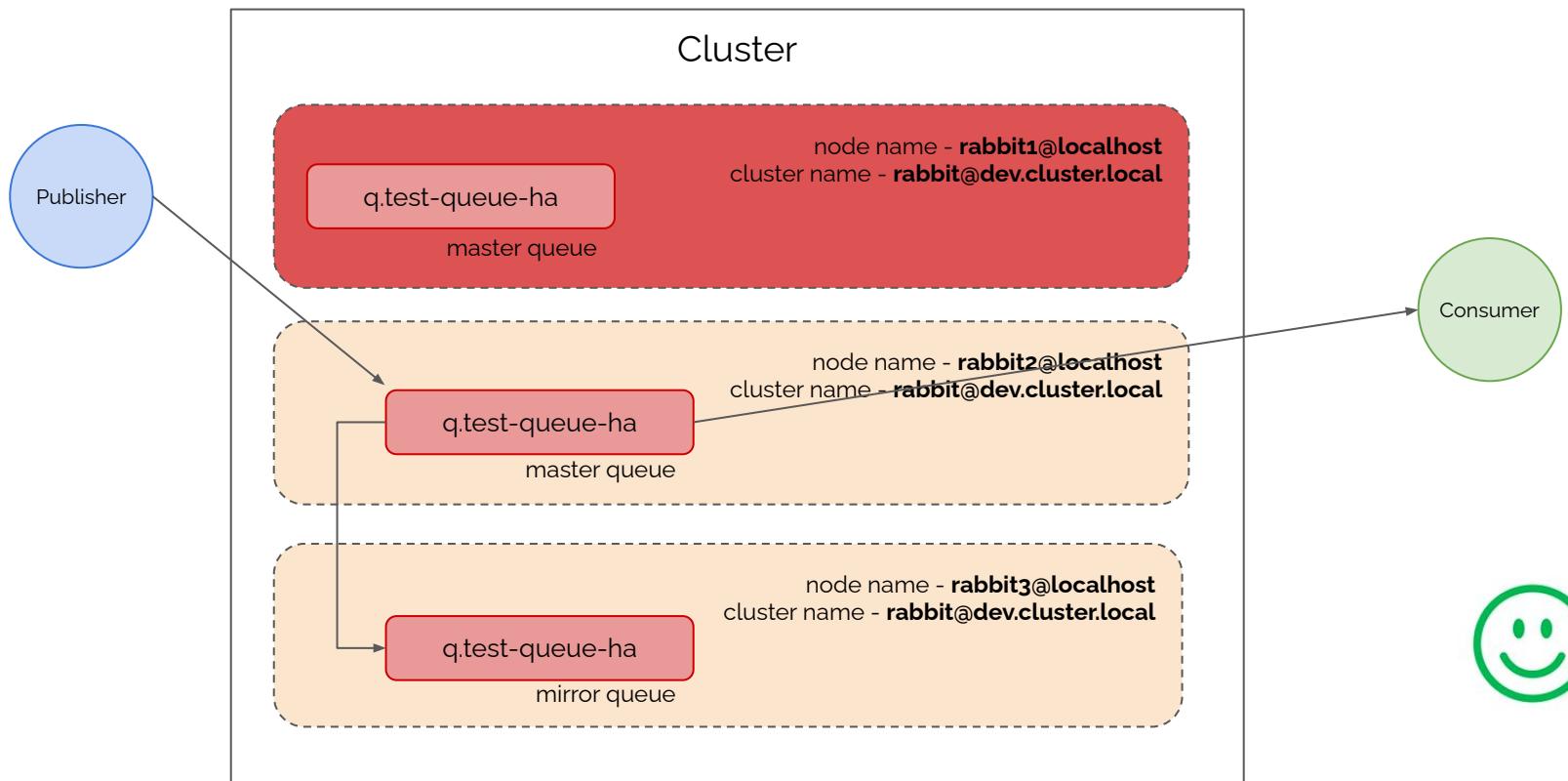
Highly Available (Mirrored) Queues

# High Availability vs Fault tolerance

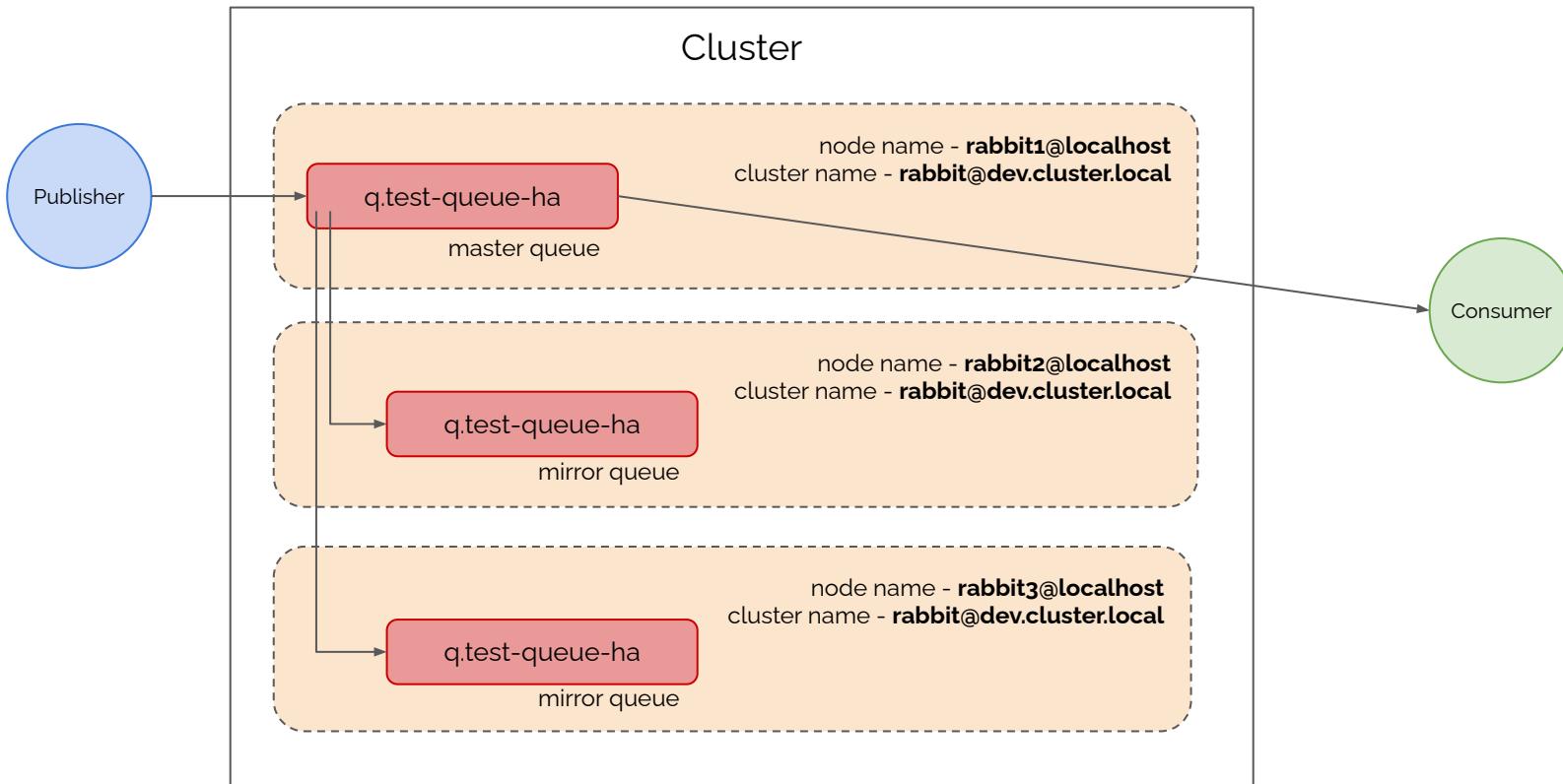
**High availability**- system might perform in degraded state

**Fault-tolerant** is a higher bar. User is not expressing any impact on the fault, SLA is met

# Highly Available (Mirrored) Queues



# Highly Available (Mirrored) Queues



# Highly Available (Mirrored) Queues

<b>Attribute</b>	<b>Description</b>
ha-mode	Possible values are: all, exactly or nodes
ha-params	Number of nodes or name of nodes (depends on the ha-mode value)
ha-sync-mode	manual (default) or automatic (increase ha-sync-batch-size on your own risk, 1 by default)
ha-promote-on-shutdown	Graceful shutdown. Possible values: when-synced (default) or always
ha-promote-on-failure	Node failure. Possible values: when-synced or always (default)

# Highly Available (Mirrored) Queues

HA queues are stored in many nodes (brokers). They work in **master/mirrors** mode. Only one node is responsible for messages handling (the one who keeps the master). Such node will replicate message to the mirrors.

When “the master” node goes down, queue on another node will be promoted to the master role. The way how it happens can be configured in the policy:

```
ha-promote-on-failure = always (or 'when-synced')  
ha-promote-on-shutdown = when-synced (or 'always')
```



Graceful shutdown (ha-promote-on-shutdown)

vs

failure (ha-promote-on-failure)

when-synced means that only a mirror that is in sync with the master can be promoted.



mirroring of classic queues will be removed in a future version of RabbitMQ. Consider using quorum queues or a non-replicated classic queue



By using HA, means we take care about messages to be safely transmitted. Use HA together with Publisher Confirms feature

# Highly Available (Mirrored) Queues

## → Exactly

2 means: 1 queue master and 1 queue mirror.

## → All

mirrored across all nodes in the cluster

## → Nodes

mirrored to particular nodes

```
PUT /api/policies/ha-two
{
  "pattern": "^two\\.",
  "definition": {
    "ha-mode": "exactly",
    "ha-params": 2,
    "ha-sync-mode": "automatic"
  }
}
```

```
> rabbitmqctl set_policy policy-test-ha "^test-ha*" "{\"ha-mode\":\"exactly\", \"ha-params\":2, \"ha-sync-mode\":\"automatic\"}”
```

# RabbitMQ Highly Available (Mirrored) Queues

**Queues**

All queues (1)

Pagination

Page 1 of 1 - Filter:   Regex ?

Overview							Messages	
Name	Node	Type	Features	State	Ready	Unacked		
test-ha2	rabbit1@localhost +1	classic	D Args policy-test-ha	idle	0	0		

**Policy: policy-test-ha**

Overview

Pattern: ^test-ha\*

Apply to: all

Definition:

- ha-mode: exactly
- ha-param: 2
- ha-sync-mode: automatic

Priority: 0

**Queue test-ha3**

Overview

Queued messages last minute ?

Message rates last minute ?

Currently idle

Details

Features: arguments: x-queue-type: classic  
durable: true

Policy: policy-test-ha

Operator policy

Effective policy definition: ha-mode: exactly  
ha-param: 2  
ha-sync-mode: automatic

Node	Mirrors
rabbit1@localhost	rabbit3@localhost



Any cluster command can be executed via:  
Command line, Management UI or REST API

# RabbitMQ why (not?) to use HA?

HA queues have some well known issues:

- **High network utilization**

Use more network than needed due to the massive communication between nodes and fact that mirrors always start empty. When non empty mirror joins the master, RabbitMQ wipes data to start replication from an empty queue

- **Replication (Synchronization) is a blocking operation**

Master queue does not accept any operations during synchronization process (fine for small queues)

- **Known redundancy issues**

In version 3.9 queue rebalance feature has been added to solve well known replication issues

- **Network partitions – bad reputation**

Slow detection of network partitions (split brain issues)

- **Lack of features**

i.e. we can't consume from mirrors



mirroring of classic queues will be removed in a future version of RabbitMQ. Consider using quorum queues or a non-replicated classic queue

# HA (Mirrored) Queues - Summary

- **Mirrored (HA) queue increase availability only**

Publishers write and consumers read from master queue only - mirrors are only used to support high availability of classic queues

- **Oldest synchronized mirror is promoted for a new master queue**

When node where master queue is located goes down

- **Each mirrored queue = 1 Erlang process**

(1 master + 2 mirrors = 3 processes)

- **One master queue may have many mirrors**

Messages are stored on master, then propagated to mirrors. Rest of operations and attributes can be applied exactly like in typical non-mirrored classic queues

- **Use Mirroring (HA) inside cluster only**

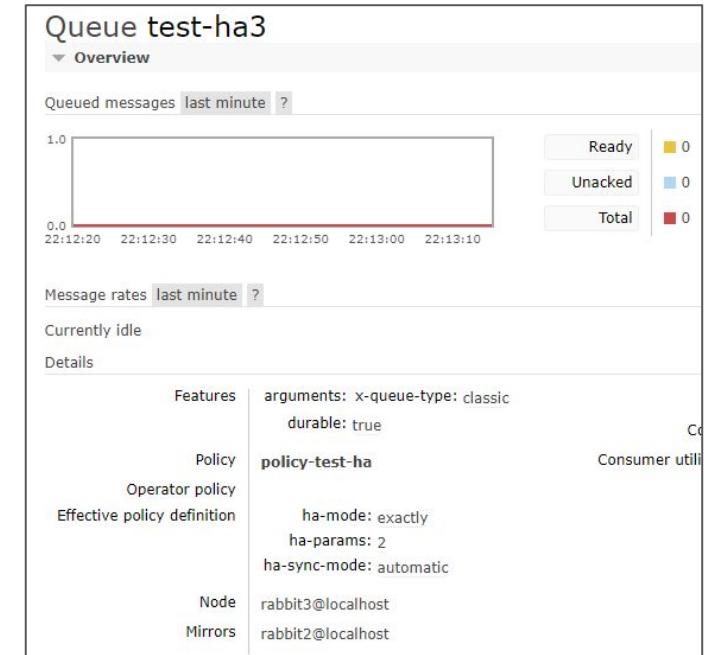
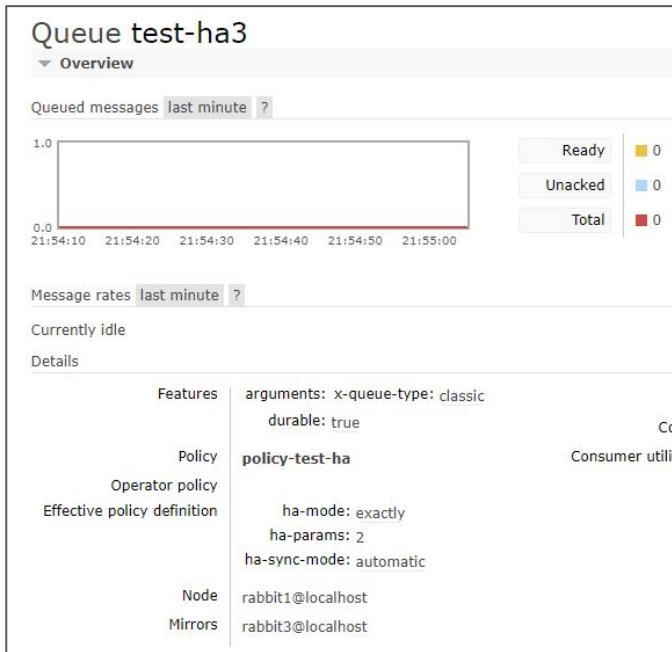
Avoid mirroring across WAN (use Federation or Shovel instead)



mirroring of classic queues will be removed in a future version of RabbitMQ. Consider using quorum queues or a non-replicated classic queue

# HA Queues - Troubleshooting

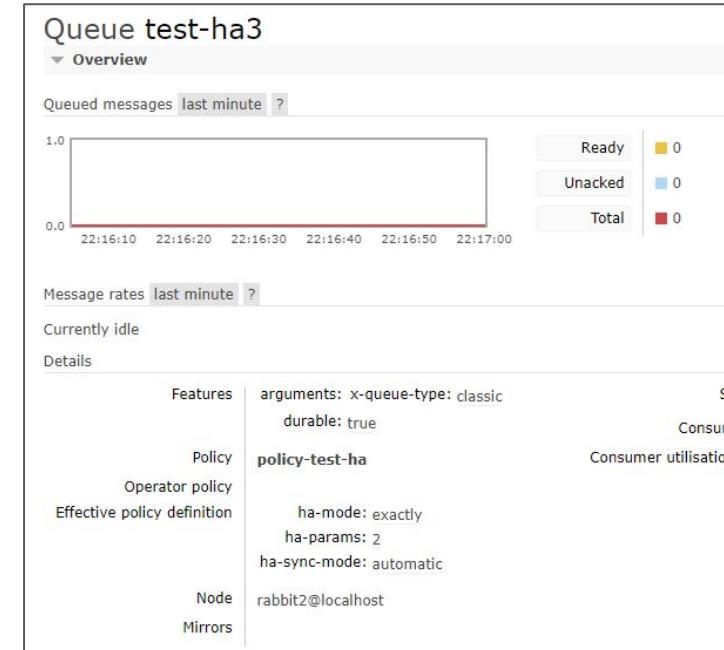
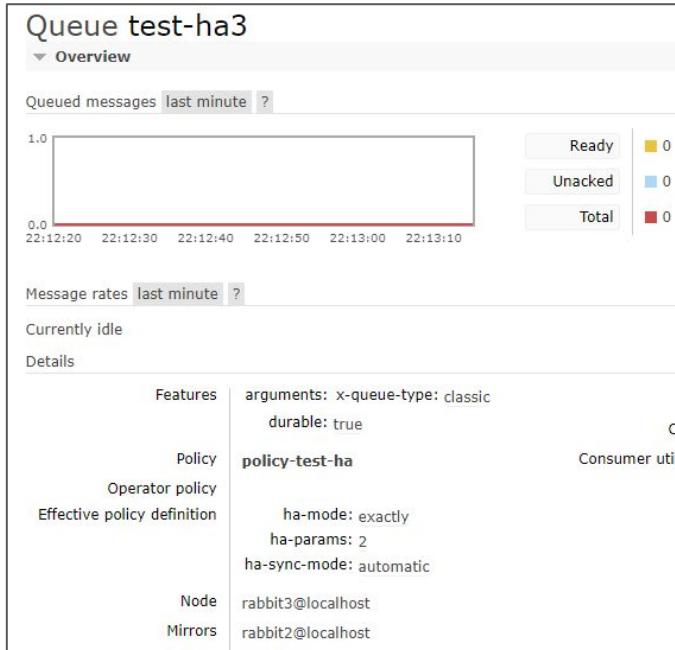
Let's disable rabbit1@localhost (the first node)



EXERCISE

# HA Queues - Troubleshooting

Let's disable rabbit3@localhost (the third node)



EXERCISE

# HA Queues - Troubleshooting

Force the mirror into **unsynchronized** state. Restore one of previously disabled nodes, i.e. rabbit1@localhost. Publish message, disable mirror, publish second message, restore mirror. ha-sync-mode can be "manual" or "automatic".

Details	
Features	arguments: x-queue-type: classic durable: true
Policy	policy-test-ha
Operator policy	Consumer utilisati
Effective policy definition	ha-mode: exactly ha-params: 2 ha-sync-mode: automatic
Node	rabbit2@localhost
Mirrors	rabbit1@localhost

➡

Details	
Features	arguments: x-queue-type: classic durable: true
Policy	policy-test-ha
Operator policy	Consumer utilisati
Effective policy definition	ha-mode: exactly ha-params: 2 ha-sync-mode: automatic
Node	rabbit2@localhost
Mirrors	rabbit1@localhost

To sync in manual mode

```
> rabbitmqctl sync_queue <vhost> <queue>
```



# HA Queues - Troubleshooting

Remember that in non-durable queues (where messages are not persisted on disk) messages will be lost after a restart.

```
> rabbitmqctl list_queues name slave_pids synchronised_slave_pids
```

```
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name      slave_pids      synchronised_slave_pids
logs_topic_queue
logs_queue_1
logs_queue_2
RetryQueue
dlxqueue2
test-ha3      [<rabbit1@localhost.1596021191.610.0>]  [<rabbit1@localhost.1596021191.610.0>]
```

```
> rabbitmqctl cancel_sync_queue <name>
```

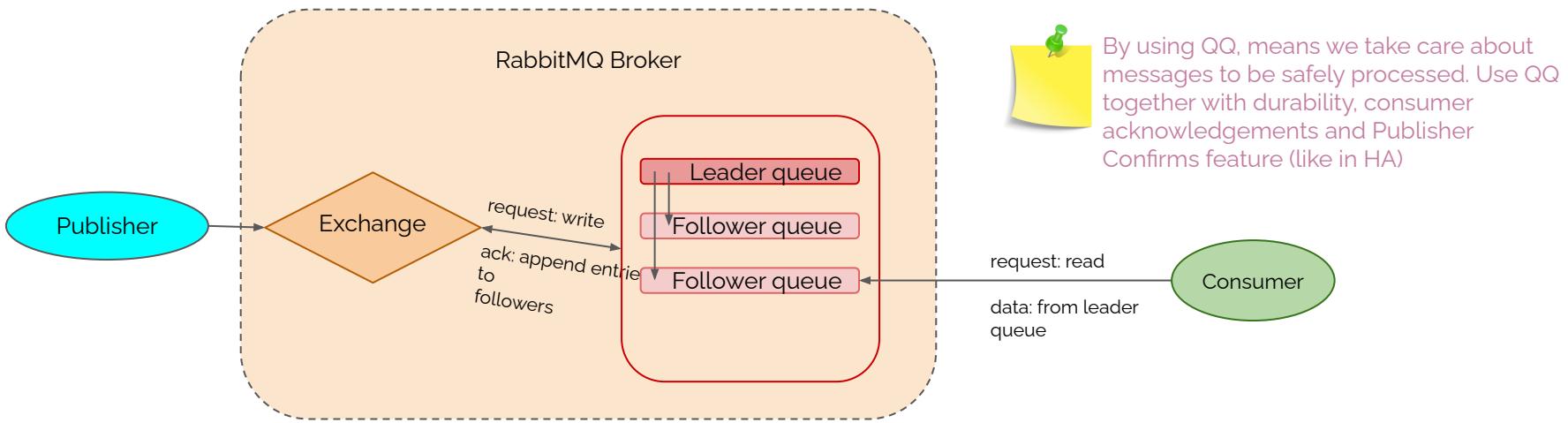


Quorum queues - new replicated queue type

# RabbitMQ Quorum Queues

Durable, replicated FIFO queue based on the **Raft** consensus algorithm. It is available as of RabbitMQ 3.8.0. It's an alternative to HA queues, but shouldn't be used in all use cases.

Quorum Queues provide comprehensible guarantees and clear failure handling over HA (HA may confirm messages too early causing potential data loss). QQ is a **log of commands** to keep followers to be in sync. Such log keeps the changes (push, pop) together with offset. Queue is a **snapshot** of operations from the log.



# RabbitMQ why (not?) to use HA?

HA queues have some well known issues:

- **High network utilization**

Use more network than needed due to the massive communication between nodes and fact that mirrors always start empty. Even if non empty mirror joins the master, RabbitMQ wipes data to start with an empty queue

- **Replication is a blocking operation**

Master does not accept any operations during data replication process (fine for small queues)

- **Known redundancy issues**

In version 3.9 queue rebalance feature has been added to solve replication issues

- **Network partitions – bad reputation**

Slow detection of Network partitions

- **Synchronization problems**

HA to be configured for consistency (pause minority) or availability (autoheal/ignore)

- **Lack of features**

i.e. we can't consume from mirrors

# Why Quorum Queues?

The long term plan is QQ to replace HA queues

- Reasonable network utilization  
The Raft algorithm reduced amount of communication between nodes
- Replication is a not blocking operation  
QQ accepts new log events during replication, no need to wipe-out messages
- New features  
We can consume from followers
- Synchronization  
Very fast detection when node goes down
- Network partitions  
Smart algorithm to solve network partitions – all thanks to the log; once the follower joins the leader, leader can revert latest changes from the follower and replicate it from the specific offset (no need to wipe out like in HA)

# Quorum Queues

Create Quorum Queue. Default replication factor is 5. It means that cluster with 3 nodes will have 3 replicas, one on each node. In a cluster of 7 nodes, 5 nodes will have replica but remaining two won't host any replicas.

▼ Add a new queue

Type: Quorum

Name: test-qq \*

Node: rabbit1@localhost

Arguments: x-quorum-initial-group-size = 5  
=

Add Max length | Max length bytes | Delivery limit |  
Dead letter exchange | Dead letter routing key | Single active co  
Max in memory bytes

**Add queue**



The replication factor we like to use is minimum 3 (x-quorum-initial-group-size)

EXERCISE

# Quorum Queues - price?

What is the price of new feature?

- CPU: Similar CPU usage
- Throughput: Similar throughput
- RAM: Higher memory usage
- HDD: Overall performance can suffer (SDD is recommended)
- Features: Lack of features



Quorum Queues store all messages in memory by default. To save the memory, use:

- x-max-in-memory-length
- x-max-in-memory-bytes

but IOPS will be higher

Quorum Queues can perform badly on HDD, because they are sensitive to IO latency. Sometimes the same message need to be saved twice.

# Quorum Queues Features



Feature	Classic Mirrored (HA)	Quorum
Non-durable queues	yes	no
Exclusivity	yes	no
Per message persistence	per message	always
Membership changes	automatic	manual
TTL	yes - both queue (x-message-ttl) and message (expiration)	no - (message "expiration" property is ignored)
Priority	yes	no
Queue length limits	yes	yes
Dead letter exchanges	yes	yes
Lazy behaviour	yes	partial (see Memory Limit)
Policies	yes	partial (dlx, queue length limits)
Reacts to memory alarms	yes	partial (truncates log of commands)
Poison message handling	no - but can be designed with DLX	Yes - when consumer repeatedly requeue a delivery RabbitMQ can delete "poison" message
Global QoS Prefetch	yes	no

# Quorum Queues use-case

Use when:

**high availability** and **data safety** are more important than **low latency and queue features** (like TTL, lazy mode, priority, single active consumer mode etc.).

Avoid using when:

- **Queue is size is very long**

quorum queues keep log segments and messages in memory (can be limited)

- **Data safety is not a priority**

- **Low latency is a very top priority**

- **Special features are needed**

Like TTL, lazy mode, priority, single active consumer etc.

- **Resources matter**

requires more resources (SSD - disk IOPS and RAM) than classic mirrored queues



Performance testing

# Performance testing - Why?

Performance Testing is not only a domain of QA engineers.

- **Stability**

Cluster to survive in a function of growing load (Ramp-Up test), stable load (Flat Line test) or mixed (Flat Line with occasional high peaks)

- **Maximum throughput and minimum latency**

Realize limits of your cluster, define SLA, be prepared for scaling-out nodes when number of published messages is growing

- **Best optimal architecture**

Which plugin, how many exchanges, what type of exchanges, what type of the Queue (Classic, Quorum, Stream), how many mirrors etc.

- **Best optimal publication and consumption parameters**

Frame size, batching, prefetch, transactions, publisher confirms etc.

# Performance testing - When?

As often as possible, on production, but doesn't impact clients of your infrastructure.

- **On demand**

Often it happens when clients start complain on performance or development team deployed new version of the software - in practice **it is too rare**

- **On a schedule**

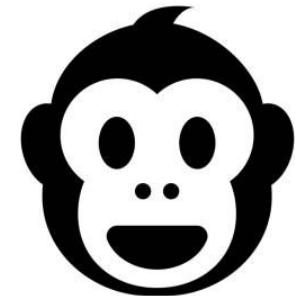
Every day or every week when your infrastructure is relatively idle. Realize any cluster failures in advance before your clients face the issue

# Game Days & Chaos Monkey

Constantly and randomly fail parts of your infrastructure, to be prepared for any unexpected situations in the real production environment. Your infrastructure must survive by performing automatic DNS failovers, automatic scaling, re-running Docker containers etc.

More sophisticated scenarios assume that SLA must be met during entire test

- **Disconnect network adapters**  
Observe the throughput, optionally count messages
- **Randomly put down nodes**  
Observe the throughput, optionally count messages
- **Randomly change access rights, exchanges or bindings**  
Drop exchanges, bindings, modify access rights, disconnect users or close random channels
- **Disconnect upstream links, stop shovels**  
Messages must reach the destination



# Performance testing tools

- **jMeter**

Plugin is available here:

<https://github.com/jlavallee/JMeter-Rabbit-AMQP#build-dependencies>

- **Benchmark testing with PerfTest**

Official RabbitMQ throughput testing tool



Performance testing - PerfTest

# RabbitMQ PerfTest

PerfTest is an official RabbitMQ throughput testing tool. It is based on the Java client and can be configured to simulate basic and advanced workloads as well

<https://www.rabbitmq.com/java-tools.html>



Download percompiled version of PerfTest

<https://bintray.com/rabbitmq/java-tools/perf-test>

or

<https://github.com/rabbitmq/rabbitmq-perf-test/releases/>



Run

runjava.bat com.rabbitmq.perf.PerfTest --help

or

./runjava com.rabbitmq.perf.PerfTest --help



Windows package has runjava.bat script

Unix package has runjava script



99th percentile is missing in console output till version 2.4 (Windows builds)

# PerfTest basic usage

PerfTest is written in Java - you need to have JRE installed.

Sample usage:

```
\bin\runjava.bat com.rabbitmq.perf.PerfTest --help
```

```
\bin\runjava.bat com.rabbitmq.perf.PerfTest <test parametres> -h amqp://user:pass@<ip>:5672
```



Windows package has runjava.bat script  
Unix package has runjava script

# RabbitMQ performance testing

- **1 producer, 2 consumers, 1kB message size**

```
> runjava com.rabbitmq.perf.PerfTest --producers 1 --consumers 2 --queue "q.test-1" --size 1000 --autoack  
--time 20 --id "test 1" -h amqp://user:pass@<ip>:5672
```

- **throughput vs latency**

```
> runjava com.rabbitmq.perf.PerfTest --producers 1 --consumers 1 --queue "q.test-1" --size 1000000 --autoack  
--time 20 --id "test 2" --framemax 5000
```

```
> runjava com.rabbitmq.perf.PerfTest --producers 1 --consumers 1 --queue "q.test-1" --size 1000000 --autoack  
--time 20 --id "test 2" --framemax 2000000
```

- **custom queue attributes**

```
> runjava com.rabbitmq.perf.PerfTest --time 20 --producers 1 --consumers 2 --queue "q.test-2" --size 1000  
--autoack --queue-args x-max-length=10,x-max-priority=5,x-dead-letter-exchange=ex.dlx-exchange-name  
--auto-delete false --id "test 3"
```

# RabbitMQ performance testing

 RabbitMQ™ | RabbitMQ 3.8.5 | Erlang 23.0.3

Overview Connections Channels Exchanges **Queues** Admin

## Queue throughput-test-1

Queued messages last minute ?



Ready: 0  
Unacked: 0  
Total: 0

Message rates last minute ?



Action	Rate
Publish	48,108/s
Deliver (manual ack)	0.00/s
Deliver (auto ack)	48,110/s
Consumer ack	0.00/s
Redelivered	0.00/s
Get (manual ack)	0.00/s
Get (auto ack)	0.00/s
Get (empty)	0.00/s

Details

Features	auto-delete: true	State	running	Messages ?	Total	Ready	Unacked	In memory	Persistent	Transient	Paged Out
Policy		Consumers	2	Message body bytes ?	0iB	0iB	0iB	0iB	0iB	0iB	0iB
Operator policy		Consumer utilisation ?	100%	Process memory ?	682kiB						
Effective policy definition											

Consumers

Channel	Consumer tag	Ack required	Exclusive	Prefetch count	Active ?	Activity status	Arguments
127.0.0.1:59922 (1)	amq.ctag-nOuLMlx1ugqDFYekiuOgg	○	○	0	•	up	
127.0.0.1:59923 (1)	amq.ctag-BXMfcpk3x2VZTgaqpul3hg	○	○	0	•	up	

Bindings

# RabbitMQ performance testing

## → Lazy queues test

Overview				Messages					Message bytes		Message rates		
Name	Type	Features	State	Ready	Unacked	In Memory	Persistent	Total	In Memory	Persistent	incoming	deliver / get	ack
q.lazy-queue	classic	D Args	idle	761,159	0	0	761,159	761,159	0iB	726MiB	0.00/s		
q.not-lazy-queue	classic	D	idle	739,776	0	229,872	739,776	739,776	219MiB	706MiB	0.00/s		

```
> runjava com.rabbitmq.perf.PerfTest --producers 50 --consumers 0 --queue "q.lazy-queue" --size 1000  
--autoack --id "test 1" -f persistent --auto-delete false --queue-args x-queue-mode=lazy --time 20
```

```
> runjava com.rabbitmq.perf.PerfTest --producers 50 --consumers 0 --queue "q.not-lazy-queue" --size  
1000 --autoack --id "test 1" -f persistent --auto-delete false --time 20
```

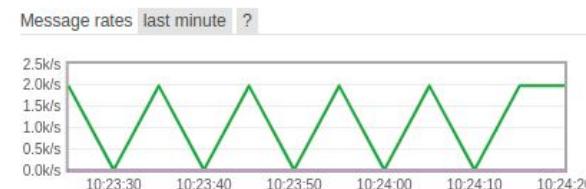
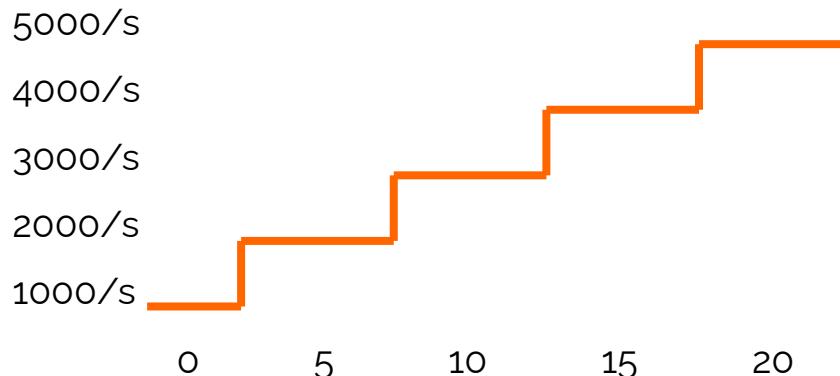
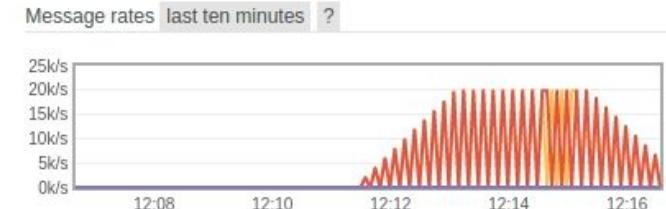


Check the memory usage during the test.  
Sample screenshots available in "Lazy Queues" section of this presentation.

# RabbitMQ performance testing

## → Ramp-Up test

```
> for i in {1..10}; do ./runjava com.rabbitmq.perf.PerfTest \  
-time 240 --producers 1 --consumers 1 \  
--queue "q.test-2" --size 1000 --autoack \  
-rate 1000 \  
-id "test 5" -h amqp://guest:guest@localhost:5672 & sleep 5; done
```



Publish	1,981/s
Publisher confirm	0.00/s
Deliver (manual ack)	0.00/s

# RabbitMQ performance testing

→ Throughput in function of number of producers

Hard limit: **52796 messages/s**

Overview			Messages					Message bytes		Message rates		
Name	Type	Features	State	Ready	Unacked	In Memory	Persistent	Total	In Memory	Persistent	incoming	deliver / get
q.perf-test-1	classic	D AD Args	idle	158,487	0	0	158,487	158,487	0iB	15MiB	11,577/s	
q.perf-test-10	classic	D AD Args	idle	166,940	0	0	166,940	166,940	0iB	16MiB	12,094/s	
q.perf-test-2	classic	D AD Args	idle	160,409	0	0	160,409	160,409	0iB	15MiB	11,223/s	
q.perf-test-3	classic	D AD Args	idle	159,949	0	0	159,949	159,949	0iB	15MiB	11,180/s	
q.perf-test-4	classic	D AD Args	idle	161,316	0	0	161,316	161,316	0iB	15MiB	11,480/s	
q.perf-test-5	classic	D AD Args	idle	159,069	0	0	159,069	159,069	0iB	15MiB	11,024/s	
q.perf-test-6	classic	D AD Args	running	176,125	0	0	176,125	176,125	0iB	17MiB	10,280/s	
q.perf-test-7	classic	D AD Args	idle	173,206	0	0	173,206	173,206	0iB	17MiB	13,277/s	
q.perf-test-8	classic	D AD Args	idle	172,478	0	0	172,478	172,478	0iB	16MiB	11,360/s	
q.perf-test-9	classic	D AD Args	idle	166,302	0	0	166,302	166,302	0iB	16MiB	11,800/s	

1 pub. 1 ch.	1 pub. 10 ch.	5 pub. 2 ch.	<b>10 pub. 1 ch.</b>
18307/s	13093/s	18006/s	<b>33765/s</b>



```
> runjava com.rabbitmq.perf.PerfTest --time 20 --queue-pattern 'q.perf-test-%d'  
--queue-pattern-from 1 --queue-pattern-to 10 --producers 1 --producer-channel-count 10  
--consumers 0 --size 100 --queue-args x-queue-mode=lazy --flag persistent
```

```
> runjava com.rabbitmq.perf.PerfTest --time 20 --queue-pattern 'q.perf-test-%d'  
--queue-pattern-from 1 --queue-pattern-to 10 --producers 10 --consumers 0  
--size 100 --queue-args x-queue-mode=lazy --flag persistent
```



more publishers - higher  
throughput (rate of  
messages/s is higher)

# RabbitMQ performance testing

→ **Simulate IoT workloads without requiring too many resources, especially threads**

Share sockets and threads when not used. NIO stands for non-blocking I/O operations.

--nio-threads 10	non-blocking IO enabled with pool of 10
--producer-scheduler-threads 10	by default on thread is used to simulate 50 producers. I declared 2000 producers, so <b>instead of 40, only 10 threads will be used</b> . Publishers publish 1 message/s, so we can limit it to 10 to save machine resources
--consumers-thread-pools 10	by default pool of threads is equal to the number of consumers. I declared 2000 consumers, so <b>instead of 2000 threads, only 10 will be used</b>
--heartbeat-sender-threads 10	<b>separate thread is used</b> for every producer and every consumer <b>to send heartbeats</b> . Set limit to reasonable value when using lot of producers and consumers and avoid java.lang.OutOfMemoryError: unable to create new native thread
--publishing-interval 1	publish message every second
--producer-random-start-delay 60	producers randomly start between 1s and 60s

```
> runjava com.rabbitmq.perf.PerfTest --time 120 --queue-pattern 'perf-test-%05d'  
--queue-pattern-from 1 --queue-pattern-to 2000 --producers 2000 --consumers 0 --nio-threads 10  
--producer-scheduler-threads 10 --consumers-thread-pools 10 --publishing-interval 1 --size 512  
--heartbeat-sender-threads 10 --flag persistent --producer-random-start-delay 60
```

# PerfTest - Summary

- **Stability**

Cluster to survive in a function of growing load (Ramp-Up test), stable load (Flat Line test) or mixed (Flat Line with occasional high peaks)

- **Maximum throughput and minimum latency**

Realize limits of your cluster, define SLA, be prepared for scaling-out nodes when number of published messages is growing

- **Best optimal architecture**

Which plugin, how many exchanges, what type of exchanges, what type of the Queue (Classic, Quorum, Stream), how many mirrors etc.

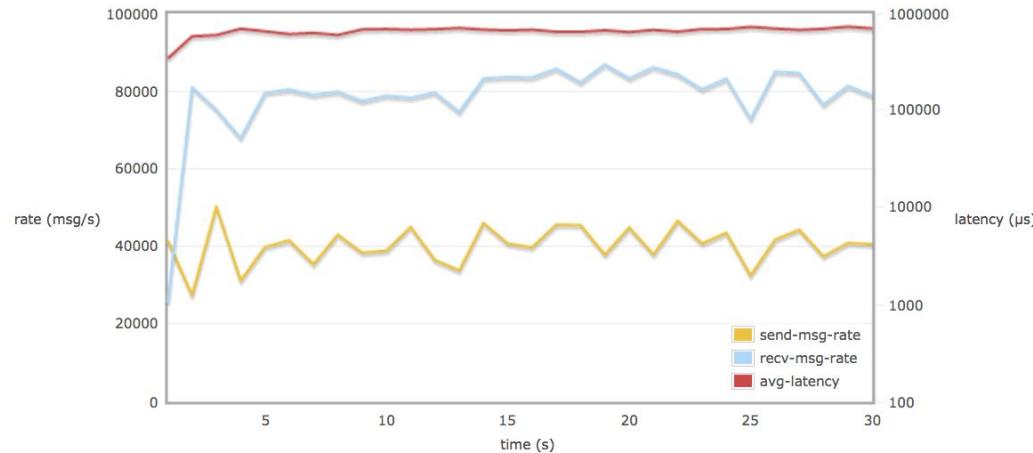
- **Best optimal publication and consumption parameters**

Frame size, batching, prefetch, transactions, publisher confirms etc.

# HTML Performance tools

Use HTML Performance Tools to visualize results

<https://github.com/rabbitmq/rabbitmq-perf-test/blob/master/html/README.md>



# HTML Performance tools

## → **Simple**

4 producers, 2 consumers, 30s test: how time affect on message rate per second and latency

```
> rabbitmq-perf-test-2.1.2\bin\runjava.bat com.rabbitmq.perf.PerfTestMulti publish-consume-spec.js  
publish-consume-result.js
```

## → **Batch test**

- 1 producer, 1 consumer, 30s no ACKs: how time affect on message rate per second and latency
- how number of producers affect message rate per second,
- rate message sizes: how message size affects the message rate per second,
- rate attempted vs latency: compare the sending rate of messages vs. the latency,

```
> rabbitmq-perf-test-2.1.2\bin\runjava.bat com.rabbitmq.perf.PerfTestMulti various-spec.js  
various-result.js
```



## Monitoring



<https://github.com/bigdotsoftware/rabbitmq-tools>

# Monitoring - why?

## → Prevent not cure

Monitor to avoid infrastructure issues before they appear and cause serious consequences. Respond to issues proactively, **preventing loss of time and money**. Minimize downtime and increase operational efficiency.

## → Detect anomalies / understand the characteristics of your data

Prepare for high spikes and unusual load by understanding the characteristics of your data (Is growing or unexpected load a DDOS attack or regular behaviour?)

## → Autoscaling

Scale-out your RabbitMQ nodes when facing high data ingest due to planned and unplanned traffic spikes. Scale-in when load is coming back to the typical value.

## → Notify right people (alerting)

When facing an outage - notify responsible people ASAP

## → Find the root cause

Drill down into specific infrastructure component - precisely determine where a problem originates

# Monitoring - how?

## → RabbitMQ RESTful API

Requires RabbitMQ management plugin installed. By default:

`http://localhost:15672/api/`

Script is a part of the management plugin:

`rabbitmqadmin --help`

## → Command line tools

They communicate using RabbitMQ distribution port (25672 by default). Root privileges needed.

`rabbitmqctl --help`

`rabbitmq-diagnostics --help`



Output can be formatted:  
`json`, `csv`, `erlang`,  
`pretty_table`, `table`

# Monitoring - what?

## 1. Cluster metrics

typical cluster metrics, like number of nodes, alarms, partitions and total rates and counts to understand the typical work environment of our cluster

## 2. Each node metrics

uptime, CPU, memory, free disk space, partitions, high churn rates

## 3. Queues size, age and resources

Some queues are more important than others, so depends on particular resources we will choose different set of metrics. Most common scenario is to monitor rates on queues and exchanges.



Management plugins to help:  
rabbitmq-plugins enable rabbitmq\_top



Set-up memory and disk alarms.  
Producers will be blocked once threshold is met



Monitoring - collecting metrics

# Cluster metrics

```
> rabbitmqctl cluster_status
```

```
> rabbitmqctl cluster_status --formatter json
```

```
+[1mDisk Nodes=[0m
rabbit1@localhost
rabbit2@localhost
rabbit3@localhost
-[1mRunning Nodes=[0m
rabbit1@localhost
rabbit2@localhost
rabbit3@localhost
-[1mVersions=[0m
rabbit1@localhost: RabbitMQ 3.9.10 on Erlang 24.1.7
rabbit2@localhost: RabbitMQ 3.9.10 on Erlang 24.1.7
rabbit3@localhost: RabbitMQ 3.9.10 on Erlang 24.1.7
-[1mMaintenance status=[0m
Node: rabbit1@localhost, status: not under maintenance
Node: rabbit2@localhost, status: not under maintenance
Node: rabbit3@localhost, status: not under maintenance
-[1mAlarms=[0m
(none)
-[1mNetwork Partitions=[0m
(none)
-[1mListeners=[0m
Node: rabbit1@localhost, interface: [::], port: 25672, protocol: clustering, purpose: inter-node and CLI tool communication
Node: rabbit1@localhost, interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbit1@localhost, interface: 0.0.0.0, port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbit2@localhost, interface: [::], port: 25682, protocol: clustering, purpose: inter-node and CLI tool communication
Node: rabbit2@localhost, interface: [::], port: 5682, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbit2@localhost, interface: 0.0.0.0, port: 5682, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbit3@localhost, interface: [::], port: 25692, protocol: clustering, purpose: inter-node and CLI tool communication
Node: rabbit3@localhost, interface: [::], port: 5692, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbit3@localhost, interface: 0.0.0.0, port: 5692, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
-[1mFeature flags=[0m
Flag: implicit_default_bindings, state: enabled
Flag: max_memory_usage, state: enabled
Flag: quorum_queue, state: enabled
Flag: stream_queue, state: enabled
Flag: user_limits, state: enabled
Flag: virtual_host_metadata, state: enabled
```

## Basics

Cluster name: rabbit@cluster.local

## Disk Nodes

rabbit@zuko-Latitude

## Running Nodes

rabbit@zuko-Latitude

## Versions

rabbit@zuko-Latitude: RabbitMQ 3.8.2 on Erlang 22.2.7

## Alarms

(none)

## Network Partitions

(none)

## Listeners

Node: rabbit@zuko-Latitude, interface: [::], port: 25672, protocol: clustering, purpose: in
Node: rabbit@zuko-Latitude, interface: [::], port: 5672, protocol: amqp, purpose: AMQP 0-9-1 and AMQP 1.0
Node: rabbit@zuko-Latitude, interface: [::], port: 15672, protocol: http, purpose: HTTP API

## Feature flags

Flag: drop\_unroutable\_metric, state: disabled
Flag: empty\_basic\_get\_metric, state: disabled
Flag: implicit\_default\_bindings, state: enabled
Flag: quorum\_queue, state: enabled
Flag: virtual\_host\_metadata, state: enabled

# Cluster metrics

```
> curl -XGET http://localhost:15672/api/overview
```

```
{  
    "management_version": "3.8.2",  
    "rates_mode": "basic",  
    "sample_retention_policies": {  
        "global": [  
            600,  
            3600,  
            28800,  
            86400  
        ],  
        "basic": [  
            600,  
            3600  
        ],  
        "detailed": [  
            600  
        ]  
    },  
    "exchange_types": [  
        {  
            "name": "direct",  
            "description": "AMQP direct exchange, as per the AMQP specification",  
            "enabled": true  
        },  
        {  
            "name": "fanout",  
            "description": "AMQP fanout exchange, as per the AMQP specification",  
            "enabled": true  
        },  
        {  
            "name": "headers",  
            "description": "AMQP headers exchange, as per the AMQP specification",  
            "enabled": true  
        },  
        {  
            "name": "topic",  
            "description": "AMQP topic exchange, as per the AMQP specification",  
            "enabled": true  
        },  
        {  
            "name": "x-consistent-hash",  
            "description": "Consistent Hashing Exchange",  
            "enabled": true  
        }  
    ],  
    "rabbitmq_version": "3.8.2",  
    "cluster_name": "rabbit@cluster.local",  
    "erlang_version": "22.2.7",  
    "erlang_full_version": "Erlang/OTP 22 [erts-10.6.4] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:128]",  
    "disable_stats": false,  
    "enable_queue_totals": false,  
    "message_stats": {
```

# Cluster metrics - summary

	Command line tools	RESTful API	Monitoring
Cluster name	y	y	
Disk Nodes	y	Use /api/nodes	
Running Nodes	y	Use /api/nodes	✓
Versions	y	y	
Alarms	y	Use /api/nodes	✓
Network partitions	y	Use /api/nodes	✓
Listeners	y	y	
Feature flags	y	y	
Exchange types	n/a	y	
Message statistics	n/a - feature of RabbitMQ management plugin	y	✓
Churn rates	n/a - feature of RabbitMQ management plugin	y	✓
Totals (number of all messages, queues, exchanges, connections, channels)	Use other parameters, like: list_queues, list_exchanges etc.	y	

# Node metrics

```
> rabbitmqctl status
```

```
> rabbitmqctl status --formatter json
```

```
> rabbitmqctl report
```

```
> rabbitmq-diagnostics memory_breakdown
```

## Runtime

```
OS PID: 1321
OS: Linux
Uptime (seconds): 384798
RabbitMQ version: 3.8.2
Node name: rabbit@zuko-Latitude
Erlang configuration: Erlang/OTP 22 [erts-10.6.4] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:128]
Erlang processes: 462 used, 1048576 limit
Scheduler run queue: 1
Cluster heartbeat timeout (net_ticktime): 60
```

## Plugins

```
Enabled plugin file: /etc/rabbitmq/enabled_plugins
Enabled plugins:
```

- \* rabbitmq\_shovel\_management
- \* rabbitmq\_shovel
- \* rabbitmq\_consistent\_hash\_exchange
- \* rabbitmq\_management
- \* rabbitmq\_web\_dispatch
- \* rabbitmq\_management\_agent
- \* cowboy
- \* amqp\_client
- \* amqp10\_client
- \* amqp10\_common
- \* cowlib

## Data directory

```
Node data directory: /var/lib/rabbitmq/mnesia/rabbit@zuko-Latitude
```

## Config files

## Log file(s)

- \* /var/log/rabbitmq/rabbit@zuko-Latitude.log
- \* /var/log/rabbitmq/rabbit@zuko-Latitude\_upgrade.log

## Alarms

(none)

## Memory

```
Calculation strategy: rss
Memory high watermark setting: 0.4 of available memory, computed to: 6.6709 gb
allocated_unused: 0.0999 gb (34.65 %)
plugins: 0.0932 gb (32.32 %)
code: 0.0306 gb (10.6 %)
memt_db: 0.0244 gb (8.47 %)
```

# Node metrics

```
{  
    "management_version": "3.8.2",  
    "rates_mode": "basic",  
    "sample_retention_policies": {  
        "global": [  
            600,  
            3600,  
            28800,  
            86400  
        ],  
        "basic": [  
            600,  
            3600  
        ],  
        "detailed": [  
            600  
        ]  
    },  
    "exchange_types": [  
        {  
            "name": "direct",  
            "description": "AMQP direct exchange, as per the AMQP specification",  
            "enabled": true  
        },  
        {  
            "name": "fanout",  
            "description": "AMQP fanout exchange, as per the AMQP specification",  
            "enabled": true  
        },  
        {  
            "name": "headers",  
            "description": "AMQP headers exchange, as per the AMQP specification",  
            "enabled": true  
        },  
        {  
            "name": "topic",  
            "description": "AMQP topic exchange, as per the AMQP specification",  
            "enabled": true  
        },  
        {  
            "name": "x-consistent-hash",  
            "description": "Consistent Hashing Exchange",  
            "enabled": true  
        }  
    ],  
    "rabbitmq_version": "3.8.2",  
    "cluster_name": "rabbit@cluster.local",  
    "erlang_version": "22.2.7",  
    "erlang_full_version": "Erlang/OTP 22 [erts-10.6.4] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:128]",  
    "disable_stats": false,  
    "enable_queue_totals": false,  
    "message_stats": {  
}
```

```
> curl -XGET http://localhost:15672/api/nodes
```

```
> curl -XGET http://localhost:15672/api/nodes/<nodename>
```

```
> curl -XGET http://localhost:15672/api/nodes/<nodename>?memory=true&binary=true
```

# Node metrics - Summary

	Command line tools	RESTful API	Monitoring
Uptime	y	y	✓
Versions	y	y	
Alarms	y	y	✓
Erlang processes	y	y	✓?
File and socket descriptors	y	y	✓?
Free disk space	y	y	✓?
Total allocated memory	y	y	✓
Memory breakdown	y	y	✓?
Partitions	n - use cluster_status	y	✓
Churn rates (i.e. connections, channels, queues, exchanges, vhosts)	n/a - feature of RabbitMQ management plugin	y	✓

# Node metrics - Channels monitoring

## → Channel leaks

Application (publisher or consumer) opens channels without closing them

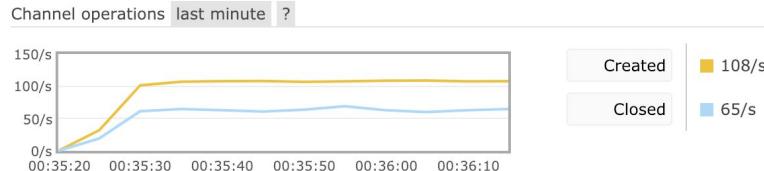
```
> curl -XGET 'http://127.0.0.1:15672/api/nodes/'  
> rabbitmqctl list_connections name channels -q
```

Global counts ?

Connections: 2    Channels: 1816    Exchanges: 21    Queues: 0    Consumers: 0

## → High channel churn

Both: Rate of newly opened channels rate of closed channels are consistently high

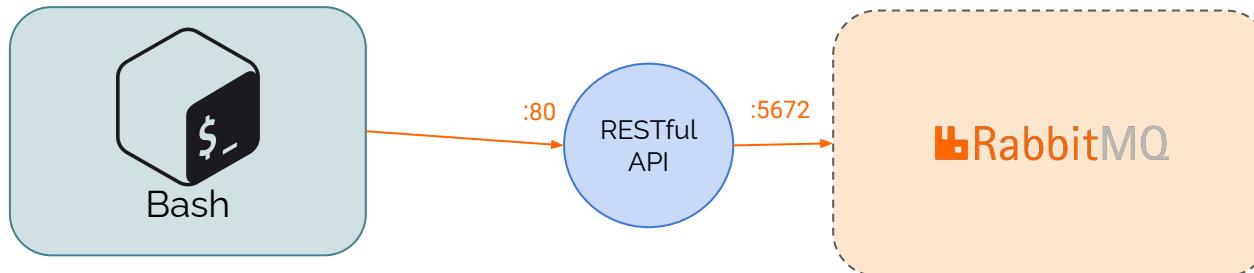
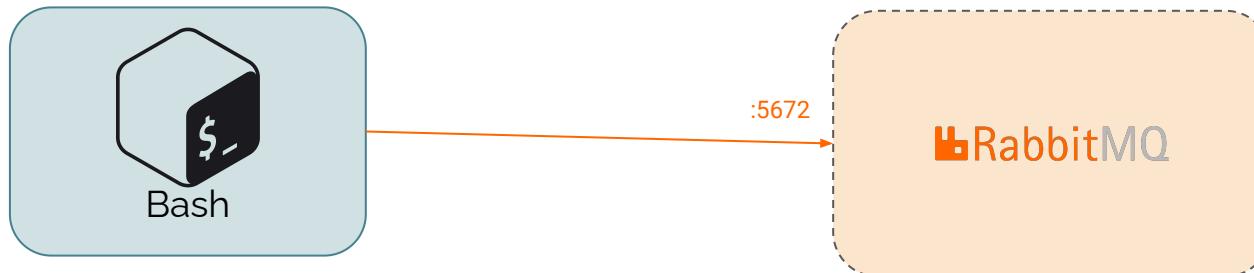


```
> curl -XGET 'http://127.0.0.1:15672/api/nodes/'
```

```
connection_created: 54  
connection_created_details:  
    rate: 0  
    connection_closed: 54  
connection_closed_details:  
    rate: 5.2  
    channel_created: 54  
channel_created_details:  
    rate: 0  
    channel_closed: 54  
channel_closed_details:  
    rate: 5.2
```

# Node metrics - Channels monitoring

To avoid "High channel churn" while interacting with i.e. Bash, we need to introduce an additional layer into our architecture, like RESTful API service.



# Node metrics - Channels monitoring

```
> rabbitmqctl list_connections name channels -q
```

vhost	connection	number	prefetch_count	messages_unconfirmed
/	<rabbit1@localhost.1596021187.10794.0>	1	0	0
	<rabbit1@localhost.1596021187.10815.0>	1	0	0
	<rabbit1@localhost.1596021187.10826.0>	1	0	0
	<rabbit1@localhost.1596021187.10837.0>	1	0	0
	<rabbit1@localhost.1596021187.10848.0>	1	0	0
	<rabbit1@localhost.1596021187.10850.0>	1	0	0
	<rabbit1@localhost.1596021187.10870.0>	1	0	0
	<rabbit1@localhost.1596021187.10881.0>	1	0	0
	<rabbit1@localhost.1596021187.10892.0>	1	0	0
	<rabbit1@localhost.1596021187.10903.0>	1	0	0
	<rabbit1@localhost.1596021187.10914.0>	1	0	0
	<rabbit1@localhost.1596021187.10925.0>	1	0	0
	<rabbit1@localhost.1596021187.10936.0>	1	0	0
	<rabbit1@localhost.1596021187.10947.0>	1	0	0
	<rabbit1@localhost.1596021187.10958.0>	1	0	0
	<rabbit1@localhost.1596021187.10969.0>	1	0	0
	<rabbit1@localhost.1596021187.10980.0>	1	0	0
	<rabbit1@localhost.1596021187.10991.0>	1	0	0
	<rabbit1@localhost.1596021187.11002.0>	1	0	0
	<rabbit1@localhost.1596021187.11013.0>	1	0	0
	<rabbit1@localhost.1596021187.11024.0>	1	0	0
	<rabbit1@localhost.1596021187.11035.0>	1	0	0
	<rabbit1@localhost.1596021187.11046.0>	1	0	0
	<rabbit1@localhost.1596021187.11057.0>	1	0	0
	<rabbit1@localhost.1596021187.11068.0>	1	0	0
	<rabbit1@localhost.1596021187.11079.0>	1	0	0

name	channels
[::1]:59376 -> [::1]:5672	1
[::1]:59377 -> [::1]:5672	1
[::1]:59378 -> [::1]:5672	1
[::1]:59379 -> [::1]:5672	1
[::1]:59380 -> [::1]:5672	1
[::1]:59381 -> [::1]:5672	1
[::1]:59382 -> [::1]:5672	1
127.0.0.1:59383 -> 127.0.0.1:5672	1
[::1]:59384 -> [::1]:5672	1
127.0.0.1:59385 -> 127.0.0.1:5672	1
127.0.0.1:59386 -> 127.0.0.1:5672	1
127.0.0.1:59387 -> 127.0.0.1:5672	1
127.0.0.1:59388 -> 127.0.0.1:5672	1
[::1]:59389 -> [::1]:5672	1
[::1]:59390 -> [::1]:5672	1
127.0.0.1:59391 -> 127.0.0.1:5672	1
127.0.0.1:59392 -> 127.0.0.1:5672	1
[::1]:59393 -> [::1]:5672	1
[::1]:59394 -> [::1]:5672	1
127.0.0.1:59395 -> 127.0.0.1:5672	1
127.0.0.1:59396 -> 127.0.0.1:5672	1
127.0.0.1:59397 -> 127.0.0.1:5672	1
[::1]:59398 -> [::1]:5672	1
[::1]:59399 -> [::1]:5672	1
127.0.0.1:59400 -> 127.0.0.1:5672	1
127.0.0.1:59401 -> 127.0.0.1:5672	1

channel count  
per connection

```
> rabbitmqctl list_channels -q vhost connection number prefetch_count messages_unconfirmed
```

# Resources metrics

```
> rabbitmqctl list_exchanges
```

```
> rabbitmqctl help list_exchanges
```

```
> rabbitmqctl list_exchanges --formatter json
```

```
> rabbitmqctl list_queues
```

```
> rabbitmqctl help list_queues
```

```
> rabbitmqctl list_queues --formatter json
```

```
> rabbitmqctl list_vhosts
```

```
> rabbitmqctl list_policies
```

```
> rabbitmqctl list_users
```

```
Listing exchanges for vhost / ...
```

name	type
amq.match	headers
ex.test-system-x	direct
amq.rabbitmq.trace	topic
ex.balance-app2	x-consistent-hash
amq.topic	topic
ex.application1	fanout
amq.fanout	fanout
amq.headers	headers
amq.direct	direct
	direct

```
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
```

name	messages
q.perf-test-3	312437
q.perf-test-10	300681
q.perf-test-8	316176
q.perf-test-9	316527
q.perf-test-1	322144
q.perf-test-7	313310
q.perf-test-4	301409
q.perf-test-5	337107
q.perf-test-6	311395
q.perf-test-2	312437

# Resources metrics

```
{  
    "arguments": {},  
    "auto_delete": false,  
    "durable": false,  
    "internal": false,  
    "message_stats": {  
        "publish_in": 135601,  
        "publish_in_details": {  
            "rate": 2970.6  
        },  
        "publish_out": 135604,  
        "publish_out_details": {  
            "rate": 2971.2  
        }  
    },  
    "name": "direct",  
    "type": "direct",  
    "user_who_performed_action": "guest",  
    "vhost": "/"  
}
```

```
{  
    "cluster_state": {  
        "rabbit@zuko-Latitude": "running"  
    },  
    "description": "",  
    "message_stats": {  
        "ack": 100538,  
        "ack_details": {  
            "rate": 2972.4  
        },  
        "confirm": 0,  
        "confirm_details": {  
            "rate": 0  
        },  
        "deliver": 100538,  
        "deliver_details": {  
            "rate": 2972.2  
        },  
        "deliver_get": 100538,  
        "deliver_get_details": {  
            "rate": 2972.2  
        },  
        "deliver_no_ack": 0,  
        "deliver_no_ack_details": {  
            "rate": 0  
        }  
    },  
    "publish": 12168,  
    "publish_details": {  
        "rate": 990.4  
    },  
    "redeliver": 0,  
    "redeliver_details": {  
        "rate": 0  
    }  
}
```

```
"head_message_timestamp": null,  
"memory": 6666088,  
"message_bytes": 0,  
"message_bytes_paged_out": 0,  
"message_bytes_persistent": 0,  
"message_bytes_ram": 0,  
"message_bytes_ready": 0,  
"message_bytes_unacknowledged": 0,  
"message_stats": {  
    "ack": 12152,  
    "ack_details": {  
        "rate": 990.6  
    },  
    "deliver": 12152,  
    "deliver_details": {  
        "rate": 990.6  
    },  
    "deliver_get": 12152,  
    "deliver_get_details": {  
        "rate": 990.6  
    },  
    "deliver_no_ack": 0,  
    "deliver_no_ack_details": {  
        "rate": 0  
    },  
    "get": 0,  
    "get_details": {  
        "rate": 0  
    },  
    "get_empty": 0,  
    "get_empty_details": {  
        "rate": 0  
    },  
    "get_no_ack": 0,  
    "get_no_ack_details": {  
        "rate": 0  
    },  
    "publish": 12168,  
    "publish_details": {  
        "rate": 990.4  
    },  
    "redeliver": 0,  
    "redeliver_details": {  
        "rate": 0  
    }  
},  
1
```

```
> curl -XGET http://localhost:15672/api/exchanges
```

```
> curl -XGET http://localhost:15672/api/queues
```

```
> curl -XGET http://localhost:15672/api/vhosts
```

# Resources metrics - Summary

	Command line tools	RESTful API	Monitoring
Queue rates (created, deleted, publish, consume)	n/a - feature of RabbitMQ management plugin	y	✓
Queue bytes (publish, consume)	n/a - feature of RabbitMQ management plugin	y	✓
Queue size	y	y	✓
Exchanges (publish rates and number of bytes)	n/a - feature of RabbitMQ management plugin	y	✓
Channels rate (created, deleted, publish, consume)	n/a - feature of RabbitMQ management plugin	y	✓
Channels (number of bytes published and delivered)	n/a - feature of RabbitMQ management plugin	y	✓
Users (number of connected)	y	y	✓
Consumers	y	y	
Connections (send, recv bytes)	n/a - feature of RabbitMQ management plugin	y	✓
Bindings	y	y	
VHosts (send bytes, recv bytes, rates)	n/a - feature of RabbitMQ management plugin	y	✓



Monitoring - memory model

# Queues - memory

- **Queues always keep part of messages in memory**

RabbitMQ deliver messages to consumers as fast as possible

- **Queue is an Erlang process**

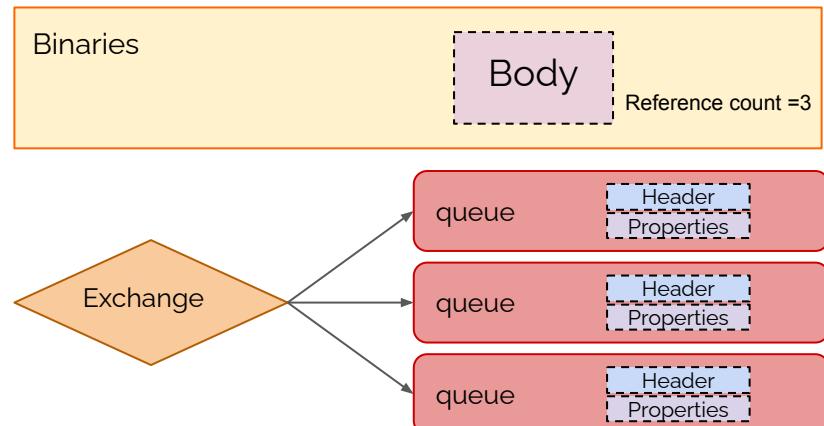
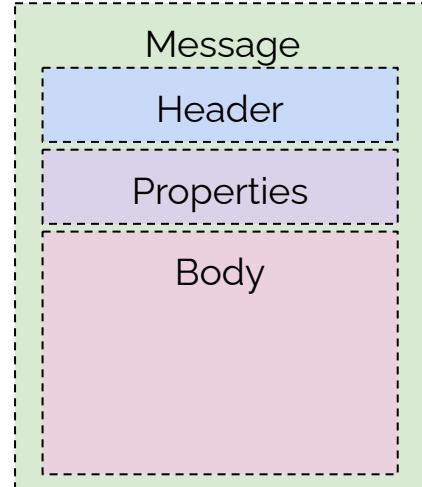
Has its own heap (security & reliability)

- **Body is stored separately in a separate memory**

Body of the message is stored in "Binaries"

- **Service booting**

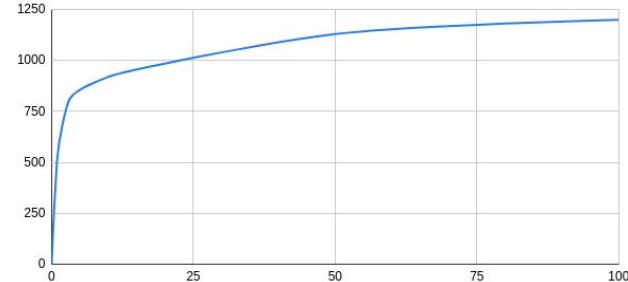
When RabbitMQ starts, up to 16384 messages smaller than 4k are loaded into memory



# Memory test - results

300 000 messages \* 1kB each = 300 MB

<b>Test #1</b>	<b>Start</b>	<b>End</b>
<b>Binaries</b>	0	322 MB
<b>Memory Queue</b>	0	194 MB
<b>Memory RSS</b>	256 MB	814 MB ( $\uparrow$ 558 MB)



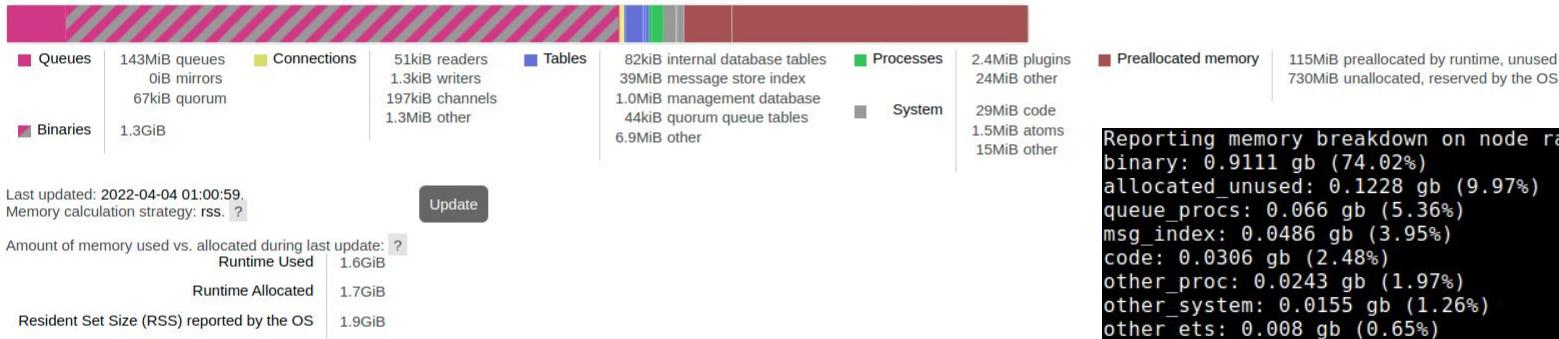
<b>Test #2</b>	<b>Start</b>	<b>End</b>
<b>Binaries</b>	0	321 MB
<b>Memory Queue</b>	0	584 MB
<b>Memory RSS</b>	254 MB	1.4 GB ( $\uparrow$ 1.1 GB)

We don't consume 3x more memory, but we store 3x more messages



# Memory breakdown

## Memory details



k, kiB →	kibibytes	( $2^{10} - 1,024$ bytes)
M, MiB →	mebibytes	( $2^{20} - 1,048,576$ bytes)
G, GiB →	gibibytes	( $2^{30} - 1,073,741,824$ bytes)
kB →	kilobytes	( $10^3 - 1,000$ bytes)
MB →	megabytes	( $10^6 - 1,000,000$ bytes)
GB →	gigabytes	( $10^9 - 1,000,000,000$ bytes)

```
Reporting memory breakdown on node rabbit@zuko-Latitude...
binary: 0.9111 gb (74.02%)
allocated_unused: 0.1228 gb (9.97%)
queue_procs: 0.066 gb (5.36%)
msg_index: 0.0486 gb (3.95%)
code: 0.0306 gb (2.48%)
other_proc: 0.0243 gb (1.97%)
other_system: 0.0155 gb (1.26%)
other_ets: 0.008 gb (0.65%)
plugins: 0.0018 gb (0.14%)
atom: 0.0015 gb (0.12%)
mgmt_db: 0.0005 gb (0.04%)
metrics: 0.0002 gb (0.02%)
mnesia: 0.0001 gb (0.01%)
quorum_queue_procs: 0.0001 gb (0.01%)
quorum_ets: 0.0 gb (0.0%)
connection_other: 0.0 gb (0.0%)
connection_readers: 0.0 gb (0.0%)
connection_writers: 0.0 gb (0.0%)
connection_channels: 0.0 gb (0.0%)
queue_slave_procs: 0.0 gb (0.0%)
reserved_unallocated: 0.0 gb (0.0%)
```

```
> rabbitmq-diagnostics memory_breakdown
```

```
> curl -XGET 'http://127.0.0.1:15672/api/nodes/rabbit1@localhost/memory'
```

```
> curl -XGET 'http://127.0.0.1:15672/api/nodes/rabbit1@localhost?memory=true&binary=true'
```

# Memory calculation strategy

## → **Legacy (erlang)**

Oldest one and fairly inaccurate strategy - underreport the real memory usage.

## → **Allocated**

Available from version 3.6.11 (August 2017). It queries Erlang memory allocator for details. This strategy is used by default on Windows.

## → **RSS**

Available from version 3.6.11 (August 2017). OS-specific approach - it queries the kernel to find RSS (Resident Set Size) value of the process. This strategy is most precise and used by default on unixes



How memory is calculated:  
`vm_memory_calculation_strategy = legacy/rss/allocated`



Currently RabbitMQ calculates memory usage using both strategies at the same time, so you can compare the difference. Selected strategy tells RabbitMQ which value should be used to determine if the memory usage reaches the watermark or paging to disk is required.

# Queue Metrics monitoring

## → Queue size

How many messages wait to be consumed.

## → Maximum messages age

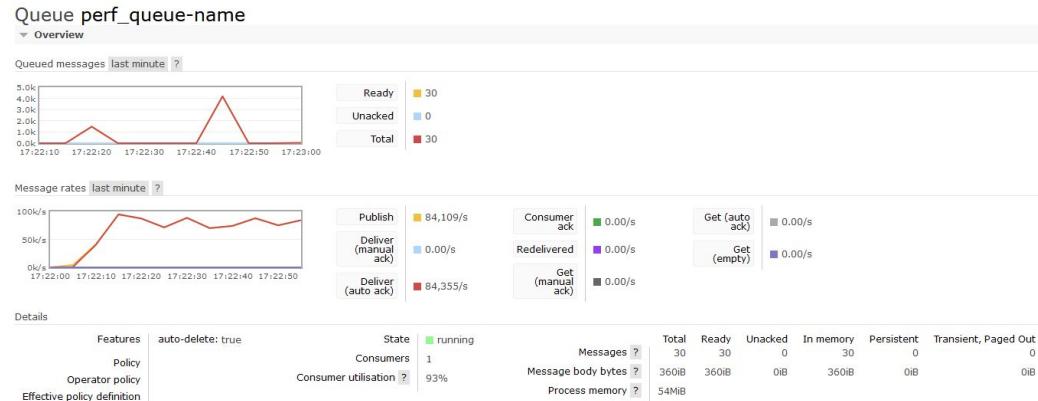
Similarly to `IteratorAge` in AWS Kinesis - number of ms message spent in the queue. There is no out-of-the-box solution, measure it manually.

→ Incoming / Outgoing bytes

## Throughput in bytes

## → Incoming / Outgoing messages

## Throughput in number of messages



# Queue Metrics monitoring

Messages age (Iterator age) - this need to be implemented manually. There are two suggested approaches

- **Consumer oriented**

Producer to store publication timestamp in message header. Consumer to measure time and emit metric. Known issue: no consuming = no metrics.

- **Producer oriented**

Producer to emit metric about message publication + store publication timestamp in message header. Consumer to close the metric.

# Node statistics monitoring

- **Connections and Channels**

Number of connections and channels, channel leaks and “High channel churn”

- **Memory usage**

Memory breakdown is useful to decide on a possible optimizations

- **Active Partitions**

Expected result is : No active partitions

- **Active Alarms**

# Monitoring - Other

Use rabbitmq **management plugin**, **RESTful API** , **rabbitmqctl** or **rabbitmq-diagnostics** command line tools

```
curl -s -XGET http://localhost:15672/api/overview
```

```
curl -s -XGET http://localhost:15672/api/nodes
```

```
curl -s -XGET http://localhost:15672/api/nodes/<name>
```

```
curl -s -XGET http://localhost:15672/api/connections
```

```
curl -s -XGET http://localhost:15672/api/channels
```

```
curl -s -XGET http://localhost:15672/api/channels/<name>
```

```
curl -s -XGET http://localhost:15672/api/exchanges
```

```
curl -s -XGET http://localhost:15672/api/exchanges/<name>
```

```
curl -s -XGET http://localhost:15672/api/queues
```

```
curl -s -XGET http://localhost:15672/api/vhosts/
```

```
> rabbitmq-diagnostics memory_breakdown  
> rabbitmq-diagnostics check_if_node_is_quorum_critical  
  
> rabbitmqctl status  
> rabbitmqctl list_queues
```

# RabbitMQ Monitoring - Tracer

Tracer is very simple AMQP 0-9-1 protocol analyzer. The easiest way to play with it is to locate Tracer between Producer and RabbitMQ.

1. Stop RabbitMQ
  2. Start RabbitMQ on different port, i.e 5678 by setting:  
set RABBITMQ\_NODE\_PORT=5678
  3. Start RabbitMQ
  4. Run tracer:

```
> runtracer 5672 localhost 5678
```

```
1597098054715: <Tracer-10> ch#1 -> {#method<basic.publish>(ticket=0, exchange=direct, routing-key='perf-test-10', mandatory=false, immediate=false), #contentHeader<basic>(content-type=null, content-encoding=null, headers=null, delivery-mode=2, priority=null, correlation-id=null, reply-to=null, expiration=null, message-id=null, timestamp=null, type=null, user-id=null, app-id=null, cluster-id=null), " \? \u{u-}?"?d"}  
1597098054715: <Tracer-10> ch#1 -> {#method<basic.publish>(ticket=0, exchange=direct, routing-key='perf-test-10', mandatory=false, immediate=false), #contentHeader<basic>(content-type=null, content-encoding=null, headers=null, delivery-mode=2, priority=null, correlation-id=null, reply-to=null, expiration=null, message-id=null, timestamp=null, type=null, user-id=null, app-id=null, cluster-id=null), " \? \u{u-}({$}"?d"}  
1597098054715: <Tracer-10> ch#1 -> {#method<basic.publish>(ticket=0, exchange=direct, routing-key='perf-test-10', mandatory=false, immediate=false), #contentHeader<basic>(content-type=null, content-encoding=null, headers=null, delivery-mode=2, priority=null, correlation-id=null, reply-to=null, expiration=null, message-id=null, timestamp=null, type=null, user-id=null, app-id=null, cluster-id=null), " \? \u{u-}({?}"?d"}  
1597098054716: <Tracer-10> ch#1 -> {#method<basic.publish>(ticket=0, exchange=direct, routing-key='perf-test-10', mandatory=false, immediate=false), #contentHeader<basic>(content-type=null, content-encoding=null, headers=null, delivery-mode=2, priority=null, correlation-id=null, reply-to=null, expiration=null, message-id=null, timestamp=null, type=null, user-id=null, app-id=null, cluster-id=null), " \? \u{u-}?)?"?d"}  
1597098054716: <Tracer-10> ch#1 -> {#method<basic.publish>(ticket=0, exchange=direct, routing-key='perf-test-10', mandatory=false, immediate=false), #contentHeader<basic>(content-type=null, content-encoding=null, headers=null, delivery-mode=2, priority=null, correlation-id=null, reply-to=null, expiration=null, message-id=null, timestamp=null, type=null, user-id=null, app-id=null, cluster-id=null), " \? \u{u-}?)?"?d"}
```

# Monitoring - Tracer





Monitoring - Alarms

# RabbitMQ Alarms

RabbitMQ set and respect thresholds. When they are reached, RabbitMQ raises an alarm and blocks connections that publish messages. Alarms can be raised:

- When memory goes above the safe limit
- When free disk space drops below the safe limit
- When number of file and socket descriptors are too high
- When number of Erlang processes is too high

In all scenarios RabbitMQ temporarily blocks connections. Connection for heartbeat monitoring is also disabled.

There is additional “Flow Control” feature related to connections which publish or consume too fast.

Overview						
Name	Exclusive	Parameters	Policy	State	Ready	
idle-q		D		idle	0	
persistent-q		D		flow	117,215	
transient-q		D		running	0	

# RabbitMQ Alarms

## → **File descriptors**

Managed by operating system. On Unix use “**ulimit -n**”, on Windows set **ERL\_MAX\_PORTS** environment variable

## → **Socket descriptors**

Subset of file descriptors initially set as **80%-90% of file descriptors limit**. RabbitMQ gives priority to the network operations - with growing load more descriptors are being used by sockets and less are available for files causing slower disk operations

## → **Erlang processes**

One million by default. Use **RABBITMQ\_MAX\_NUMBER\_OF\_PROCESSES** or **RABBITMQ\_SERVER\_ERL\_ARGS** environment variables. Increasing is not a good practice - scale out your cluster.

## → **Memory**

40% RAM by default. Use **vm\_memory\_high\_watermark** in rabbitmq.conf file.

## → **Free disk space**

50MB by default. Use **disk\_free\_limit** in rabbitmq.conf file.

# Alarms - Summary

- **Never too high, never too low**

Take into account type, size and volume of data which is processed by your cluster as well hardware limits

- **Don't increase erlang processes**

Default value of  $2^{20} = 1048576$  is already very high. Scale-out your cluster by adding more nodes (IoT workload)

- **Monitor alarms**

Better prevent than cure

# RabbitMQ Memory Alarms

As default RabbitMQ memory threshold is set to 40% of available RAM. More memory can be allocated, just by passing 40% RabbitMQ will start throttle (block connections).

Name	File descriptors	Socket descriptors	Erlang processes	Memory	Disk space	Uptime	Info	Reset stats	+/-
rabbit1@localhost	0 32768 available	26 29401 available	779 1048576 available	131MiB 130MiB high watermark	42GiB 48GiB low watermark	1m 10s	basic disc 1 rss	This node All nodes	
rabbit2@localhost	0 65536 available	0 58893 available	481 1048576 available	117MiB 13GiB high watermark	42GiB 48MiB low watermark	2h 17m	basic disc 1 rss	This node All nodes	
rabbit3@localhost									

vm\_memory\_high\_watermark.relative = 0.4

```
> rabbitmqctl set_vm_memory_high_watermark 0.6
> rabbitmqctl set_vm_memory_high_watermark absolute "4G"
```

Overview					Details				Net
Name	Node	User name	State	SSL / TLS	Protocol	Channels	Fro		
127.0.0.1:58930	rabbit1@localhost	guest	blocking	o	AMQP 0-9-1	1	0iB	perf-test-consumer-0	
127.0.0.1:58931	rabbit1@localhost	guest	blocked	o	AMQP 0-9-1	1	0iB	perf-test-producer-0	
127.0.0.1:58934	rabbit1@localhost	guest	blocked	o	AMQP 0-9-1	1	0iB	perf-test-producer-3	
127.0.0.1:58935	rabbit1@localhost	guest	blocked	o	AMQP 0-9-1	1	0iB	perf-test-producer-4	
127.0.0.1:58939	rabbit1@localhost	guest	blocked	o	AMQP 0-9-1	1	0iB	perf-test-producer-8	
127.0.0.1:58940	rabbit1@localhost	guest	blocked	o	AMQP 0-9-1	1	0iB	perf-test-producer-9	
127.0.0.1:58941	rabbit1@localhost	guest	blocked	o	AMQP 0-9-1	1	0iB	perf-test-producer-10	
127.0.0.1:58945	rabbit1@localhost	guest	blocked	o	AMQP 0-9-1	1	0iB	perf-test-producer-14	
127.0.0.1:58947	rabbit1@localhost	guest	blocked	o	AMQP 0-9-1	1	0iB	perf-test-producer-16	

# RabbitMQ Disk Alarms

As default RabbitMQ disk threshold is set to 50MB. When free disk space drops below, an alarm will be triggered and RabbitMQ will block connections.



Overview									Details			Net
Name	Node	User name	State	SSL / TLS	Protocol	Channels	Fro					
<b>127.0.0.1:58930</b> perf-test-consumer-0	rabbit1@localhost	guest	blocking	○	AMQP 0-9-1	1	0iB					
<b>127.0.0.1:58931</b> perf-test-producer-0	rabbit1@localhost	guest	blocked	○	AMQP 0-9-1	1	0iB					
<b>127.0.0.1:58934</b> perf-test-producer-3	rabbit1@localhost	guest	blocked	○	AMQP 0-9-1	1	0iB					
<b>127.0.0.1:58935</b> perf-test-producer-4	rabbit1@localhost	guest	blocked	○	AMQP 0-9-1	1	0iB					
<b>127.0.0.1:58939</b> perf-test-producer-8	rabbit1@localhost	guest	blocked	○	AMQP 0-9-1	1	0iB					
<b>127.0.0.1:58940</b> perf-test-producer-9	rabbit1@localhost	guest	blocked	○	AMQP 0-9-1	1	0iB					
<b>127.0.0.1:58941</b> perf-test-producer-10	rabbit1@localhost	guest	blocked	○	AMQP 0-9-1	1	0iB					
<b>127.0.0.1:58945</b> perf-test-producer-14	rabbit1@localhost	guest	blocked	○	AMQP 0-9-1	1	0iB					
<b>127.0.0.1:58947</b> perf-test-producer-16	rabbit1@localhost	guest	blocked	○	AMQP 0-9-1	1	0iB					

disk\_free\_limit.absolute = 1GB

```
> rabbitmqctl set_disk_free_limit 1GB
```

# RabbitMQ Flow Control

There is additional “Flow Control” feature related to connections which publish or consume too fast. RabbitMQ uses “Credit Flow Control” algorithm. It's not configurable, exchanges, queues, connections can be in “flow” state.

Overview				Details			Network	
Name	Node	User name	State	SSL / TLS	Protocol	Channels	From client	To client
127.0.0.1:61099 perf-test-producer-0	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61102 perf-test-producer-3	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61103 perf-test-producer-4	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61105 perf-test-producer-6	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61107 perf-test-producer-8	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61111 perf-test-producer-12	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61113 perf-test-producer-14	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61114 perf-test-producer-15	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61115 perf-test-producer-16	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61117 perf-test-producer-18	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61120 perf-test-producer-21	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
127.0.0.1:61123 perf-test-producer-24	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s
[::1]:61098 perf-test-consumer-0	rabbit1@localhost	guest	green running	o	AMQP 0-9-1	1	0iB/s	0iB/s
[::1]:61100	rabbit1@localhost	guest	yellow flow	o	AMQP 0-9-1	1	0iB/s	0iB/s

Overview						
Name	Exclusive	Parameters	Policy	State	Ready	
idle-q	D		idle	0		
persistent-q	D		yellow flow	117,215		
transient-q	D		green running	0		



Security

# RabbitMQ Security

- ✗ Network security
  - ✗ Firewalls
  - ✗ Proper hardware configuration
  - ✗ Load balancers
  - ✗ File permissions
  - ✗ Operation system security
- ✓ Information Security

## → Exposing sensitive data (data leak)

Messages may contain personal information, credit card numbers, passwords or strictly confidential data. Use access rights and encryption at-rest. RabbitMQ does not encrypt messages at-rest. You need to encrypt messages on producer side and decrypt on consumer side. In transit (SSL/TLS) is available.

## → System failure

Poison message can crash consumers or put them into infinite loop. More sophisticated techniques may be used for unauthorized access to other parts of your infrastructure



# Information Security



May 2017 - the largest coordinated ransomware attack by malware called **WannaCry**. Over **200 000 computers** infected in over **150 countries**. Individuals were forced to pay a ransom to decrypt encrypted files. Many public health services impacted (outdated operating systems). Cost is estimated in **many \$billions**

In 2018 **British Airways** was the victim of an attack which affected over **400 000 customers** (personal data and credit card data). **Fee: £183 million** for inadequate security control.



In 2017 about 163 milion personal data was stolen (social security numbers, birth dates, addresses and information used to conduct identity fraud) from **Equifax**. Cost: over **\$1.4 billion** to cover compensations, lawyers, fines and security investition.

# Information Security Regulations

## → **GDPR (EU)**

The GDPR aims primarily to give control to individuals over their personal data and to simplify the regulatory environment for international business by unifying the regulation within the EU

## → **BDSG (Germany)**

Governs the exposure of personal data, which are manually processed or stored in IT systems

## → **FDPIC (Switzerland)**

Responsible to advise, educate and ensure the protection of personal data

## → **ICCPR (New Zealand)**

Right to privacy but it tends to hold the status of a value or an interest, rather than a right

**and many other....**



The same software hosted in different regions may have different architecture due to local regulations

# Information Security in RabbitMQ

## → Permissions (access rights)

- ◆ Delete guest user or change it's password
- ◆ Separate user per application
- ◆ Good naming convention
- ◆ Use minimum permissions

## → Messages encryption

- ◆ At-rest encryption. RabbitMQ does not support at-rest encryption by default. You must encrypt messages on producer side and decrypt on consumer side.
- ◆ In-transit encryption. Use **SSL/TLS**. Make sure producer, consumer, nodes and entire clusters use encrypted connections



## Security - Permissions

# Permissions - Core

## → Authentication

User can be authenticated via password, X.509 certificate or external system,

## → Authorization

Permissions for user are set by **regular expressions** for **configure**, **write** and **read** operations

- ◆ **read** operations on every entity (resource)  
retrieve messages from a resource
- ◆ **write** operations on every entity (resource)  
inject messages into a resource
- ◆ **configure** operations every entity (resource)  
create or destroy resources, or modify their behaviour



RabbitMQ management plugin uses tagging for authorization (administrator, monitoring, policymaker, management, impersonator)

# Permissions - Core

AMQP 0-9-1 Operation		configure	write	read
exchange.declare	(passive=false)	exchange		
exchange.declare	(passive=true)			
exchange.declare	(with AE)	exchange	exchange (AE)	exchange
exchange.delete		exchange		
queue.declare	(passive=false)	queue		
queue.declare	(passive=true)			
queue.declare	(with DLX)	queue	exchange (DLX)	queue
queue.delete		queue		
exchange.bind			exchange (destination)	exchange (source)
exchange.unbind			exchange (destination)	exchange (source)
queue.bind			queue	exchange
queue.unbind			queue	exchange
basic.publish			exchange	
basic.get				queue
basic.consume				queue
queue.purge				queue

# Permissions - Management UI

## → **Management**

User can access the management plugin (front-end & back-end)

## → **Policymaker**

User can access the management plugin and manage policies and parameters for the vhosts he has access to

## → **Monitoring**

User can access the management plugin and see all connections and channels as well as node-related information.

## → **Administrator**

User can do everything. Monitor cluster and metrics, manage users, vhosts and permissions. Close other user's connections, manage policies and parameters for all vhosts

# Permissions - Topics

In version 3.7.0 RabbitMQ introduced feature called topic permissions. Feature is used by topic exchanges only to define to what topic user can publish and from what topic user can read messages. It's an additional layer on top of existing checks for publishers (all protocols) and consumers (MQTT and STOMP protocols only).

Routing-key is matched against a regular expression to decide whether the message can be routed downstream or not.

- **write to topic**

routing-key of a published message is matched against a regular expression to decide whether the message can be routed downstream or not)

- **read from topic**

for consumers; for the topic-oriented protocols such as MQTT and STOMP (not applies to AMQP)

# RabbitMQ Permissions

General permissions can be granted to:

- **read** operations on every entity (resource)  
retrieve messages from a resource
- **write** operations on every entity (resource)  
inject messages into a resource
- **configure** operations every entity (resource)  
create or destroy resources, or alter their behaviour

Topic permissions (available from version 3.7.0) can be granted to:

- **write to topic**  
routing-key of a published message is matched against a regular expression to decide whether the message can be routed downstream or not)
- **read from topic**  
for consumers; for the topic-oriented protocols such as MQTT and STOMP (not applies to AMQP)

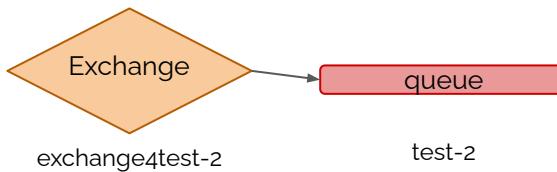
# RabbitMQ Permissions

'`^$`', i.e. matching nothing but the empty string (empty string '`''` can be used as a synonym)

'`^(amq\.gen.*|amq\.default)$`' gives a user access to server-generated names and the default exchange

'`^test-.*`' gives a user access to server-generated names prefixed by "test-", but not to default exchange  
(not possible to publish messages)

Can message be published to test-2 queue?



'`^(amq\.default)$`'

YES (via default exch.)

'`^(amq\.default|test-.*$)`'

YES (via default exch.)

'`^(exchange4test.*$)`'

YES (via exchange4test)

'`^(exchange4test-.*|test-2$)`'

YES (via default exch.)

'`^(exchange4test-.*|test-2$)`'

YES (via both)

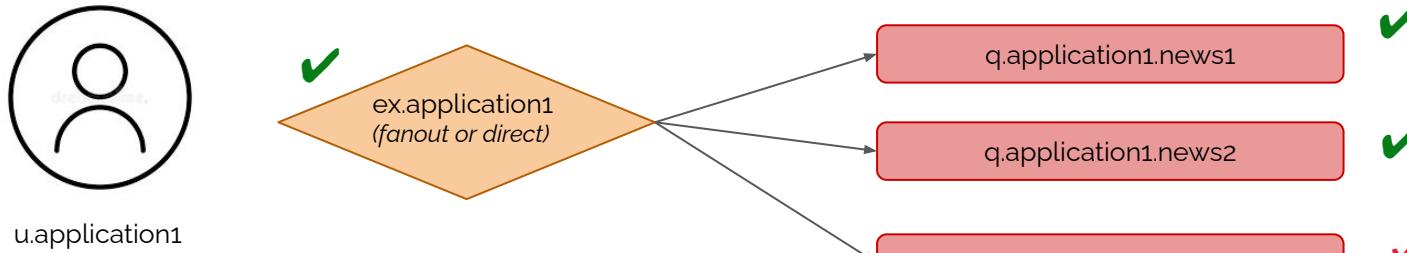
'`^(exchange4test-.*|test-2| !amq\.default$)`' YES (via exchange4test-2 only)



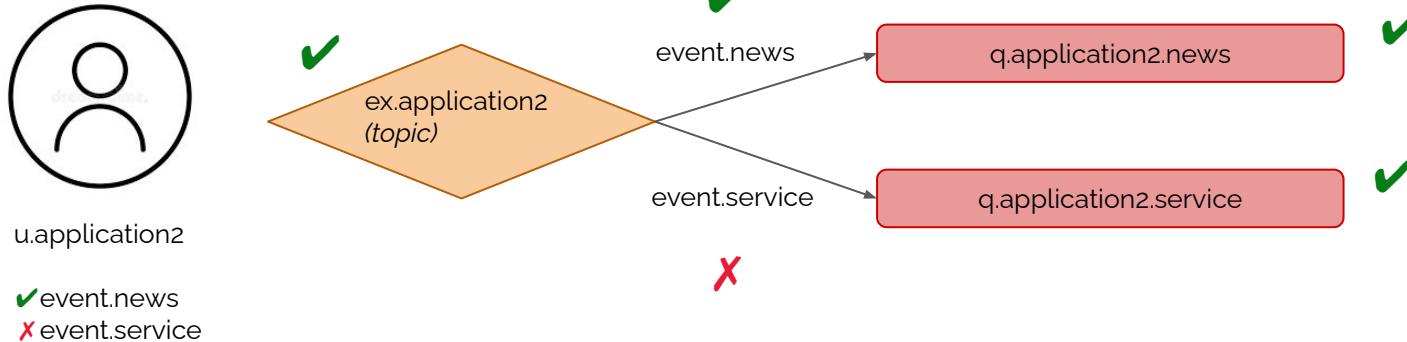
Define naming convention for exchanges and queues

# RabbitMQ Permissions - HandsOn

Publish/Consume permissions



Topic permissions



# Permissions - Summary

## → Explicitly grant user to exchanges and queues

Never leave any permissions widely open. For instance, leaving write permissions widely open allows user publish messages to any queue using for instance default (nameless) exchange. Good naming convention of exchanges and queues helps a lot!

## → Minimum permissions (least privilege)

Don't set permissions in advance. When you create permissions, follow the standard security advice of **granting least privilege**, or granting only the permissions required to perform a task

## → Examples

`'^$'`, matching nothing but the empty string (empty string `' '` can be used as a synonym). This regular expression restricts permissions.

`'ex\.test1'` gives a user access to ex.test1 and all names which includes ex.test1

`'^(ex\.test1|amq\.default)$'` gives a user access to ex.test1 and the default exchange

`'^(amq\.gen.*|amq\.default|ex\.test[0-9]+| !ex\.test1)$'` gives a user access to server-generated names, the default exchange and all resources prefixed by ex.test and followed by numbers except ex.test1

# RabbitMQ Users and Permissions

**Users**

▼ All users

Filter:   Regex ?

Name	Tags	Can access virtual hosts	Has password
guest	administrator	/	•

?

▼ Add a user

Username:  \*

Password:  .....  ..... \*

(confirm)

Tags:  ?

Set Admin | Monitoring | Policymaker  
Management | Impersonator | None

Add user

HTTP API Server Docs Tutorials Community Support Community Slack Commercial

Comma-separated list of tags to apply to the user. Currently **supported by the management plugin**:

**management**  
User can access the management plugin

**policymaker**  
User can access the management plugin and manage policies and parameters for the vhosts they have access to.

**monitoring**  
User can access the management plugin and see all connections and channels as well as node-related information.

**administrator**  
User can do everything monitoring can do, manage users, vhosts and permissions, close other user's connections, and manage policies and parameters for all vhosts.

Note that you can set any tag here; the links for the above four tags are just for convenience.

Close



Name	Tags	Can access virtual hosts	Has password
guest	administrator	/	•
testuser	administrator	No access	•

EXERCISE

# RabbitMQ Users and Permissions

The screenshot shows the RabbitMQ Management UI. On the left, there's a sidebar for 'User: testuser' with sections for Overview (Tags: administrator), Permissions (Current permissions: Virtual host /, Configure regexp: ^test-\*, Write regexp: ^\$, Read regexp: ^test-\*), Set permission (Virtual Host: /, Configure regexp: .\*, Write regexp: .\*, Read regexp: .\*), and Topic permissions (Current topic permissions: ... no topic permissions ...). A red arrow labeled 'EXERCISE' points to the 'Set permission' section.

In the center, there's a 'Channels' tab selected. A 'Publish message' dialog is open, showing fields for Delivery mode (1 - Non-persistent), Headers, Properties, and Payload (aaa). A yellow warning box displays the error message: '403 ACCESS\_REFUSED - access to exchange 'amq.default' in vhost '/' refused for user 'testuser''. A 'Close' button is at the bottom of the dialog.

On the right, there's a section titled 'Manage privileges from command line' with three terminal commands:

- > rabbitmqctl add\_user <user id> <password>
- > rabbitmqctl set\_user\_tags <user id> administrator
- > rabbitmqctl set\_permissions -p / <user id> ".\*" ".\*" ".\*"

# RabbitMQ Topic Permissions

Reuse previous example with topic\_logs exchange

**Exchanges**

▼ All exchanges (12, filtered down to 1)

Pagination

Page 1 of 1 - Filter: topic\_

Name	Type	Features	Message rate in	M
topic_logs	topic	D		

▼ Bindings

This exchange

↓

To	Routing key	Arguments	
logs_topic_queue	*.logs.error		Unbind

Set topic permission

Virtual host	Exchange	Write regexp	Read regexp
/	topic_logs	^application1.*	*

Virtual Host: /

Exchange: topic\_logs

Write regexp: ^application1.\*

Read regexp: .\*

**Set topic permission**

▼ Publish message

Routing key: application2.logs.error

Headers: ?

Properties: ?

Payload: test

**Publish message**



403 ACCESS\_REFUSED - access to topic 'application2.logs.error' in exchange 'topic\_logs' in vhost '/' refused for user 'guest'

**Close**

**EXERCISE**



Reliable system

# Issues with reliable delivery

Messaging-based systems are distributed by definition and **can fail at any moment**. Node as well server and client applications can experience **hardware failure** or they can crash due to **any reason at any time**. Additionally, even if client applications keep running, logic errors can cause channel leaks or connection errors impacting reliable message delivery. We can distinguish three groups of possible issues:

1. **Network issues**

Most common and unpredictable

2. **Hardware issues**

Less common. They impact RabbitMQ nodes, producers, consumers.

3. **Software issue**

Bugs or poor code quality

# 5 points for reliable delivery

## 1. Client's ACKs

Only manual ACKs guarantee message consumption

## 2. Transactions & Producer Confirms (aka Publisher Confirms)

Sending different type of data may require different code logic and different mechanisms like transactions or publisher conforms or both - use async or bulks together with complex retry logic

## 3. Messages to survive server restart

Durable exchange, durable queues and persistent messages

## 4. Availability

RabbitMQ Clustering and Federations with Classic HA queues, Quorum queues or Streams to distribute messages across nodes

## 5. Well designed application logic

Depends on the type the data, its volume, size of individual message, daily volume, daily size and the business requirements. Features like: rejections, alternate exchanges, TTLs, DLX, retry custom logic, consumer tag tracking etc.



Backup & Restore

# Backup & Restore

Every RabbitMQ node stores two type of data inside /data directory. Both of them can be backup and restored separately.

## 1. Schema definition (topology)

definitions for Users, vhosts, queues, exchanges, bindings, permissions, runtime parameters etc. Definitions are stored in an internal database and replicated across all nodes in the cluster, so every node has its own replica of all definitions. Definitions can only be backed up from a **running node** - active partitions impact this data in the same way messages in queues

## 2. Message store data

messages content. Messages can be replicated between nodes (HA queues, Quorum queues, Streams). Messages can only be backed up from a **stopped node** (to avoid inconsistent snapshot of the data). When using HA (mirrored) queues or Quorum queues, **entire cluster should be stopped**.



From RabbitMQ 3.7.0,  
messages are grouped in  
subfolders per virtual host

# Schema Definition Export and Import

All the nodes in the cluster store definitions for Users, vhosts, queues, exchanges, bindings, permissions, runtime parameters etc. Such group of definitions we call a **Schema Definition** (or just Schema or Topology). Schema Definitions are stored in an internal database and **replicated across all nodes**. Every node in a cluster has its **own replica of all definitions**. When a part of definitions changes, the update is performed on all nodes in a single transaction. Definitions can be exported from any cluster node with the same result.

Definitions can be exported to a file and then imported into another cluster or used for schema backup or resources monitoring purposes.

```
> # Does not require management plugin to be enabled, new in RabbitMQ 3.8.2  
> rabbitmqctl export_definitions /path/to/definitions.json
```



Use RESTful API alternatively:  
GET /api/definitions

```
> # Requires management plugin to be enabled  
> rabbitmqadmin export /path/to/definitions.json
```

# Schema Definition Export and Import

Definition import can be used for pre-configuring nodes at deployment time.

Manual import:

```
> # Does not require management plugin to be enabled, new in RabbitMQ 3.8.2  
> rabbitmqctl import_definitions /path/to/definitions.file.json
```

```
> # Requires management plugin to be enabled  
> rabbitmqadmin import /path/to/definitions.file.json
```

When node boots (in rabbitmq.conf file):

```
# Does not require management plugin to be enabled, new in RabbitMQ 3.8.2  
load_definitions = /path/to/definitions/file.json  
  
# Requires management plugin to be enabled at the time of node boot  
management.load_definitions = /path/to/definitions/file.json
```

# Messages Export and Import

Make a copy of “**message store data**” pointed by **RABBITMQ\_MNESIA\_DIR**. While restoring, make sure schema definitions are already imported.

Name	Date modified	Type	Size
coordination	6/22/2022 1:04 PM	File folder	
msg_stores	6/22/2022 1:04 PM	File folder	
quorum	6/22/2022 1:04 PM	File folder	
cluster_nodes.config	6/22/2022 1:06 PM	CONFIG File	1 KB
DECISION_TABLE.LOG	6/22/2022 1:33 PM	Text Document	1 KB
LATEST.LOG	6/22/2022 1:33 PM	Text Document	0 KB
nodes_running_at_shutdown	6/22/2022 1:06 PM	File	1 KB
rabbit_durable_exchange.DCD	6/22/2022 1:06 PM	DCD File	2 KB
rabbit_durable_exchange.DCL	6/22/2022 1:33 PM	DCL File	4 KB
rabbit_durable_queue.DCD	6/22/2022 1:06 PM	DCD File	1 KB
rabbit_durable_queue.DCL	6/22/2022 1:33 PM	DCL File	1 KB
rabbit_durable_route.DCD	6/22/2022 1:04 PM	DCD File	1 KB
rabbit_runtime_parameters.DCD	6/22/2022 1:06 PM	DCD File	1 KB
rabbit_runtime_parameters.DCL	6/22/2022 1:15 PM	DCL File	1 KB
rabbit_serial	6/22/2022 1:06 PM	File	1 KB
rabbit_topic_permission.DCD	6/22/2022 1:04 PM	DCD File	1 KB
rabbit_user.DCD	6/22/2022 1:06 PM	DCD File	1 KB
rabbit_user_permission.DCD	6/22/2022 1:06 PM	DCD File	1 KB
rabbit_user_permission.DCL	6/22/2022 1:33 PM	DCL File	1 KB
rabbit_vhost.DCD	6/22/2022 1:06 PM	DCD File	1 KB
rabbit_vhost.DCL	6/22/2022 1:33 PM	DCL File	1 KB
schema.DAT	6/22/2022 1:06 PM	DAT File	31 KB
schema_version	6/22/2022 1:04 PM	File	1 KB



Upgrade

# RabbitMQ Upgrade

## 1. In-place upgrade

Upgrade node by node with existing data on the disk

## 2. Blue-Green

Create new cluster and migrate existing data

	In-place	Blue-Green
Risk of failure	High	Low
Extra hardware	n/a	Yes

# RabbitMQ In-place upgrade

## 1. Prepare

- Check **version compatibility**,
- **Erlang version** requirements,
- Release notes,
- Features that do not support in-place upgrades, and known caveats.
- Make sure there are no ongoing synchronisation operations (i.e. HA queues), no alarms - generally node has to be in good shape.

## 2. Backup & Stop the node

## 3. Upgrade files

Upgrade RabbitMQ + Erlang

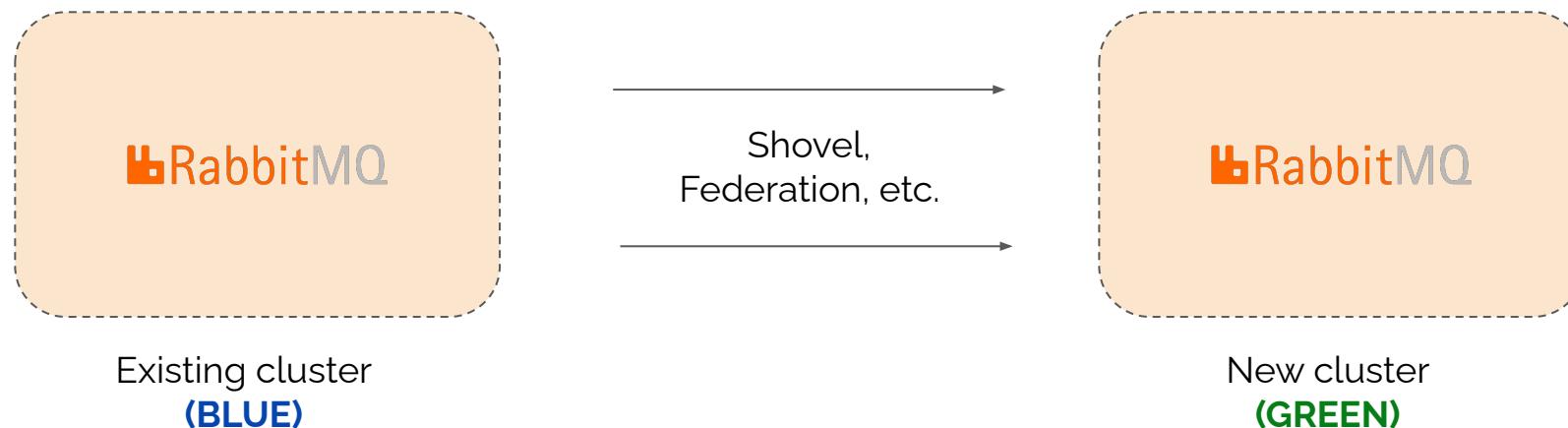
## 4. Start node

From	To
3.6.x	3.8.x
3.6.x	3.7.x
3.5.x	3.7.x
=< 3.4.x	3.6.16

# RabbitMQ Blue-Green upgrade

Safer, but more costly approach.

Data can be copied in various ways - the easiest one is to configure Shovel or Federation.





Tuning

# Tuning

## 1. Review queues configuration

Consider lazy queues, keep queues short (if possible), use TTL etc.

## 2. Review architecture

Review publishers and consumers (i.e enable NIO, change frame size, prefetch, use transient messages over persistent, use multiple queues and customers, not to use manual acks and publisher confirms), more nodes, replication, network etc.

## 3. Play with configuration

Disable plugins you are not using, review features used by RabbitMQ (CPU/MEM). Most popular quick-wins as well deprecated approaches are on the next slides.



Queues are single-threaded in RabbitMQ. Use multiple queues and customers



Persistent messages are written to disk as soon as they arrive. Use transient messages to increase the throughput.

# Tuning - config

**HiPE** support (High-Performance Erlang) has been dropped in Erlang 22. RabbitMQ no longer supports HiPE precompilation.

There were various opinion about enabling HiPE to speed up RabbitMQ. Fact that HiPE has been officially dropped, means we should not take this option into account while tuning our cluster.

`hipec_compile=true`



# Tuning - config

By default the RabbitMQ emits metrics events every 5 seconds. The message rate values shown in the management plugin are calculated over this period.

- Increase the statistics interval to i.e. 30s
- Eventually disable message rates calculation for Management plugin

Both above reduce CPU consumption.

Overview				Messages			+/-
Name	Type	Features	State	Ready	Unacked	Total	
perf_queue-name	classic	AD	running	0	0	0	



```
management.rates mode = none  
collect_statistics_interval = 30000
```

```
> rabbitmqctl eval 'application:set_env(rabbit, collect_statistics_interval, 30000).' 
```

APPROVED

# Tuning - maintenance

Prior to version 3.6.7 stats database was stored on a single node. Starting from version 3.6.7, each node has its own statistics database. The statistics database is stored in the memory, so to save some RAM, it can be reset from time to time.

RabbitMQ < 3.6.2

```
> rabbitmqctl eval 'exit(erlang:whereis(rabbit_mgmt_db), please_terminate).'
```

RabbitMQ > 3.6.2 and RabbitMQ < 3.6.7

```
> rabbitmqctl eval 'supervisor2:terminate_child(rabbit_mgmt_sup_sup,  
rabbit_mgmt_sup), rabbit_mgmt_sup_sup:start_child().'
```

RabbitMQ >= 3.6.7 (current node)

```
> rabbitmqctl eval 'rabbit_mgmt_storage:reset().'
```

RabbitMQ >= 3.6.2 (all nodes)

```
> rabbitmqctl eval 'rabbit_mgmt_storage:reset_all().'
```



Use RESTful API alternatively:  
`DELETE /api/reset`  
`DELETE /api/reset/:node`

**APPROVED**

# Tuning - config

When node starts, up to 16384 messages are loaded into RAM from the every single queue (including lazy queues).

Deep dive: 16384 messages < 4096 bytes are loaded into RAM from each queue

- Decrease value from 4096 (default) to reasonable smaller value, i.e. 512 bytes

```
queue_index_embed_msgs_below = 512
```

APPROVED

# Tuning - config

When messages are big, increase frame\_size

```
frame_size = 1000000
```

APPROVED

# Tuning - queue configuration

Quorum Queues store content on disk (per Raft requirements) as well in memory by default, use:

- x-max-in-memory-length
- x-max-in-memory-bytes

to limit.

RabbitMQ 3.10  
and newer



RabbitMQ 3.9.x  
and earlier



# Tuning - applications, config

We need to make sure our applications are well designed (both producers and consumers). Most common issue is wrong connection/channels usage

1. **Use long-lived connections and reasonable number of connections**

Establishing connection is time consuming. Avoid "High Churn" issues. Issues with batch scripts, PHP etc. Too many connections = waste of resources on both sides (RabbitMQ and App).

2. **Separate connection for publisher and consumer**

Sometimes the same process acts as a publisher and consumer. When too many messages are being produced, server might not receive the acknowledgments from the client = lower consume speed + server will be overloaded

3. **Don't share channels between threads**

Channel consumes a relatively small amount of memory on the client size in comparison to connection.. Share connection if needed, but use one channel per thread in your application

```
channel_max = 2047  
connection_max = infinity
```



# Tuning - applications

We need to make sure our applications are well designed (both producers and consumers).  
Most common issues with queues

- 1. Use relatively small queues**

Avoid too long queues (number of messages) and too big queues (in terms of bytes).

Long queues increase memory usage, sync time between nodes, increase node start time. Management interface collects and stores more stats. Use multiple queues instead of single huge one.

- 2. Use valid queue attributes (adopt queue attr to business requirements)**

Lazy queues, auto-delete, durability, TTL, max-length etc.

- 3. Make sure to use up-to-date versions**

Keep RabbitMQ and Erlang up-to-date



# Tuning - config

Increase RABBITMQ\_IO\_THREAD\_POOL\_SIZE (default is 128) when many CPUs are available

Increase RABBITMQ\_MAX\_NUMBER\_OF\_PROCESSES (default is 1mln) when more than hundreds of queues or in environment with high number of connections

**APPROVED**  
But rarely used