

# Event Based Approaches

Oxford University  
Software Engineering  
Programme  
December 2019



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons  
Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Why Asynchronous?



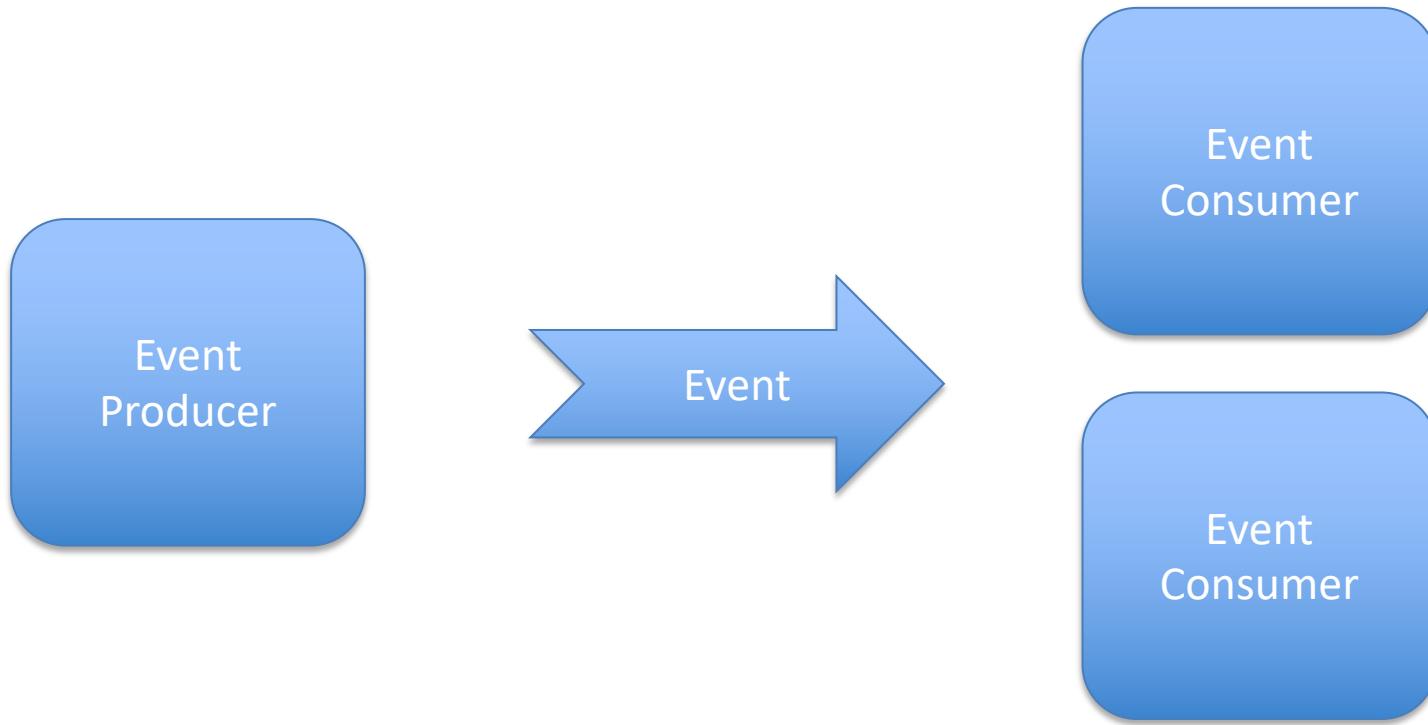
© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Loose coupling



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Event Driven Architecture



# Loose coupling in EDA

- Location
  - Logical addresses
- Time
  - Asynchronous, Store/Forward, Replay
- Message
  - JSON, XML, ProtoBuf, etc
- Pattern
  - Pub/Sub, Queue, 1-1, 1-many, many-many, request-reply

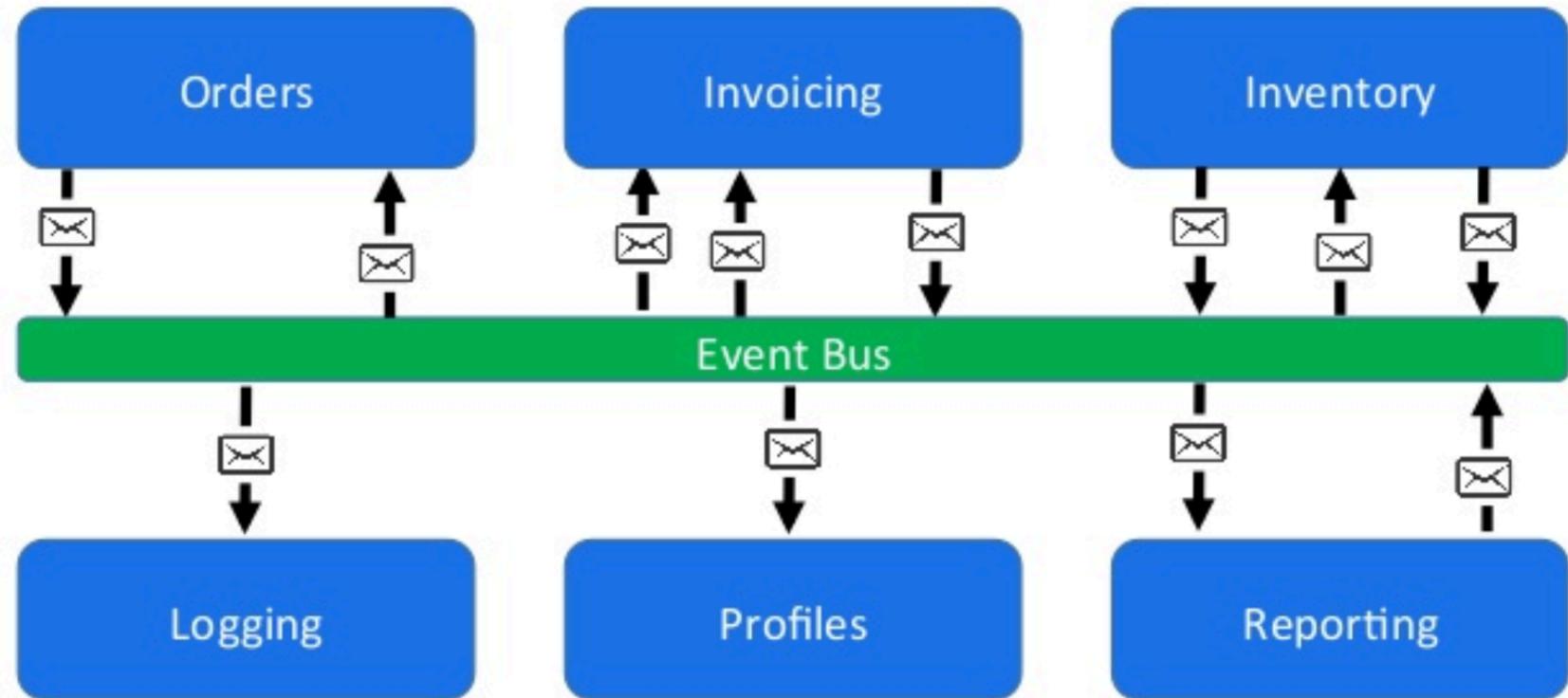


# EDA



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Loose coupling via event bus

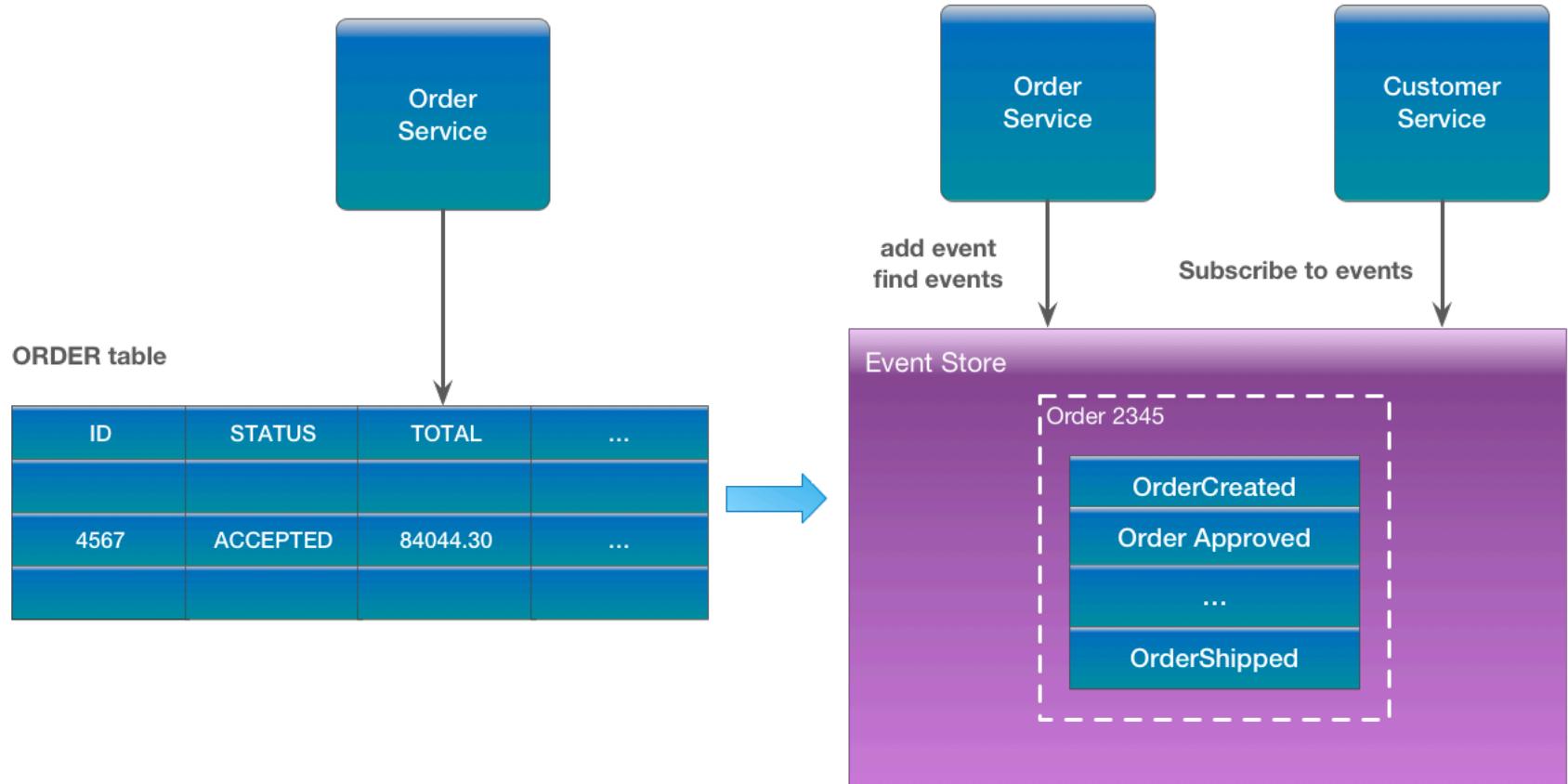


<https://www.slideshare.net/CentricConsulting/eventdriven-architecture-57613466>



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Event Sourcing



<https://eventuate.io/whyeventsourcing.html>



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Message distribution systems

- NATS
- MQTT
- STOMP
- AMQP / RabbitMQ
- Kafka



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# NATS

- Simple text based protocol
- Multiple patterns
  - Pure Pub/Sub
  - Request-Reply
  - Queuing
- Clustered servers
  - Distributed queue across clusters
  - Cluster aware clients



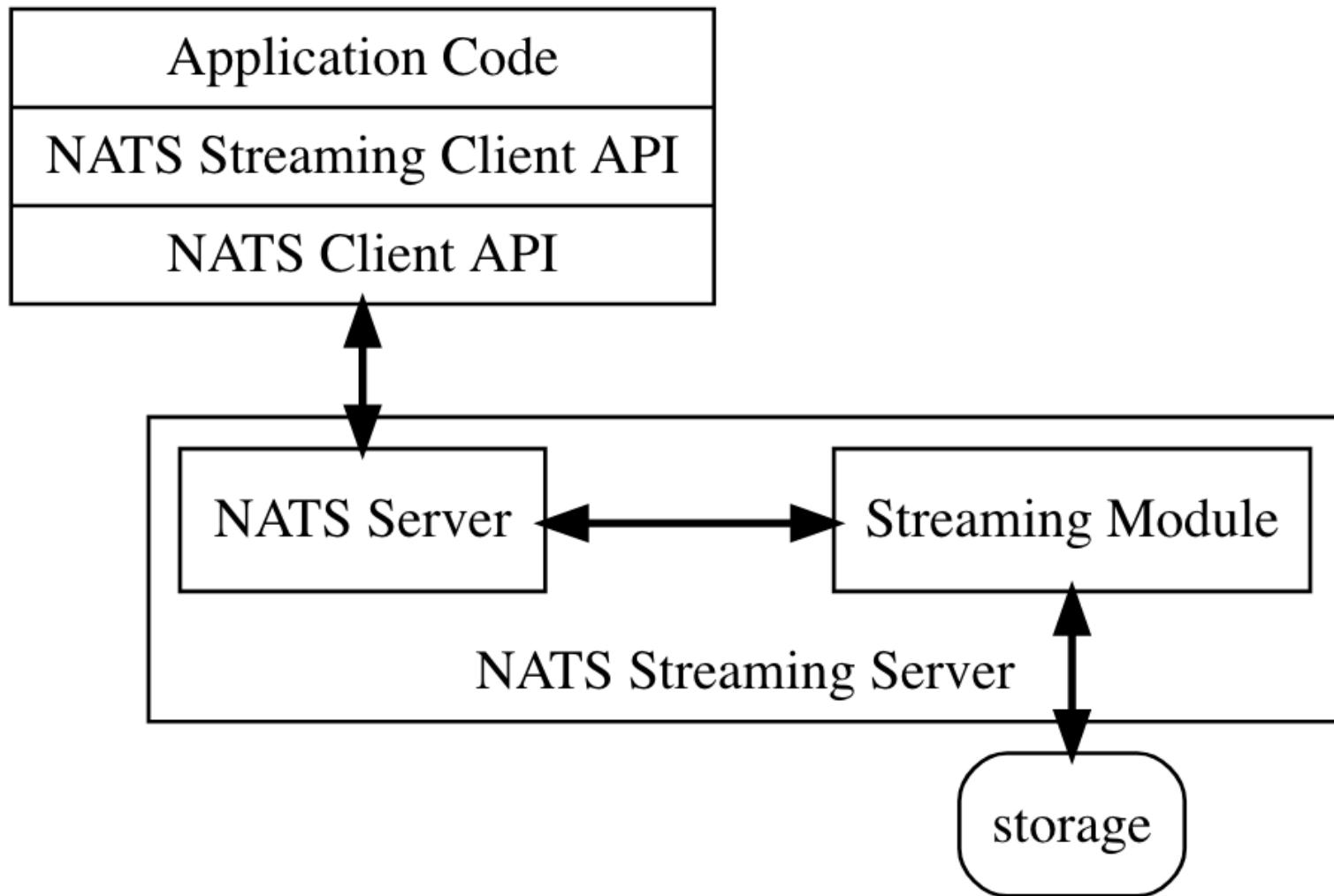
© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# NATS simple demo

```
Pauls-MacBook-Pro-3:~ paul$
```



# NATS Streaming



# NATS Streaming

- At least once delivery
- Publisher rate limiting
- Subscriber rate limiting
- Message Replay
- Durable Subscriptions



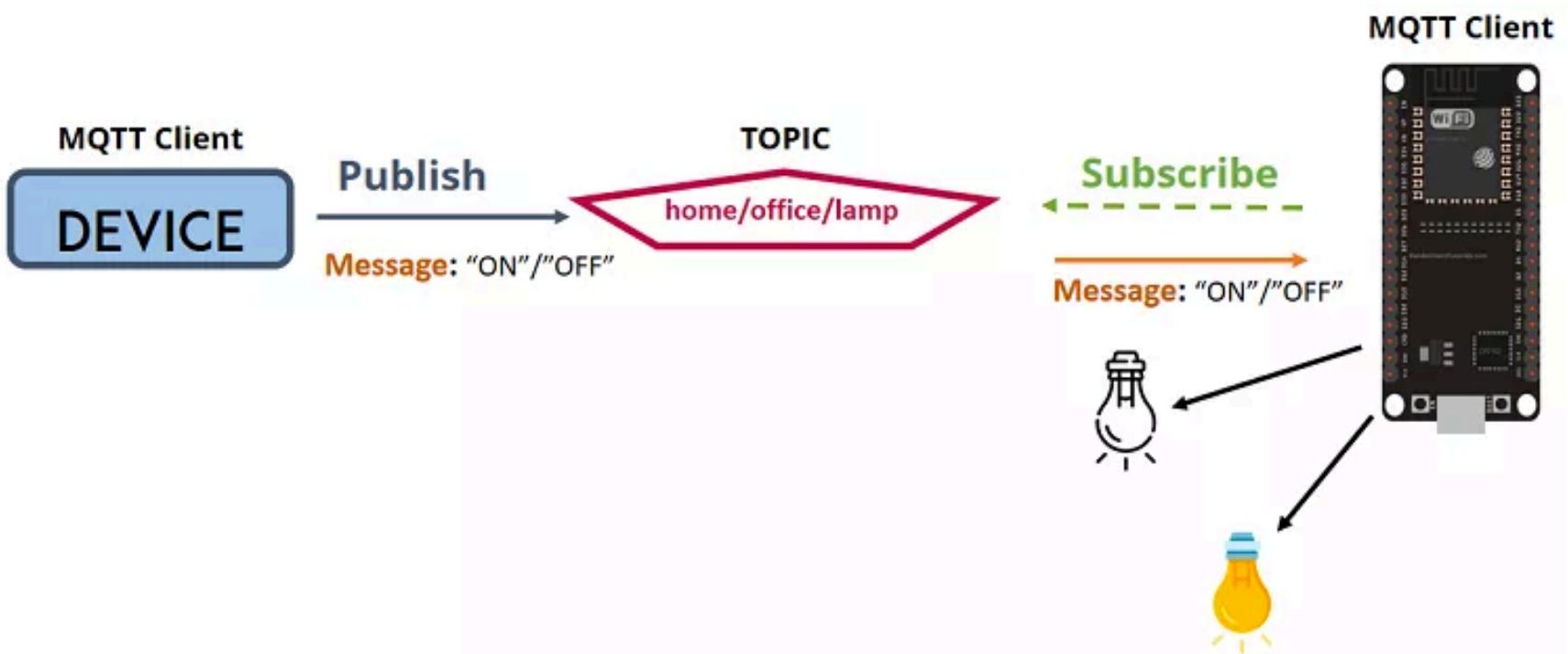
© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# MQTT

- Very lightweight binary protocol
  - 2-byte overhead
- Widely used in IoT scenarios
- Pub-sub only until MQTT5
- QoS levels
  - Fire and forget QoS0
  - At least once QoS1
  - Exactly once QoS2

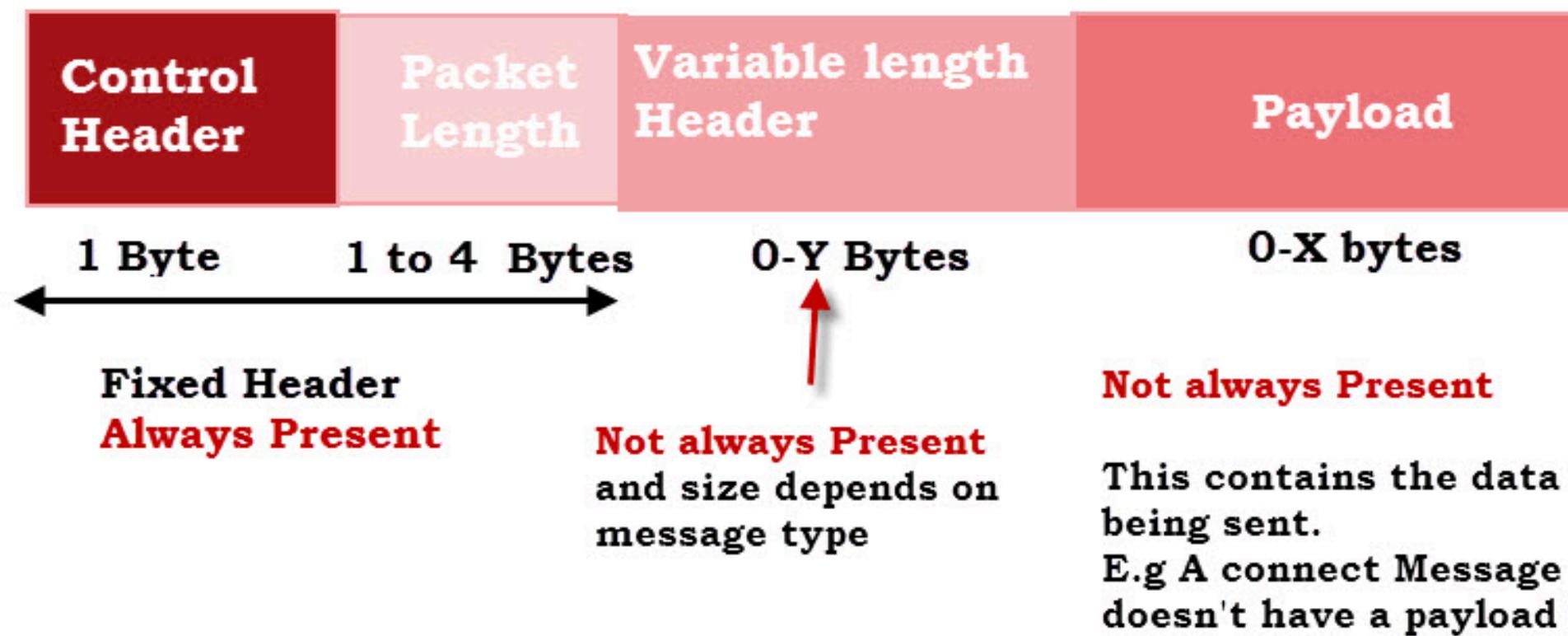


# MQTT



# MQTT Packets

<http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/>



## MQTT Standard Packet Structure



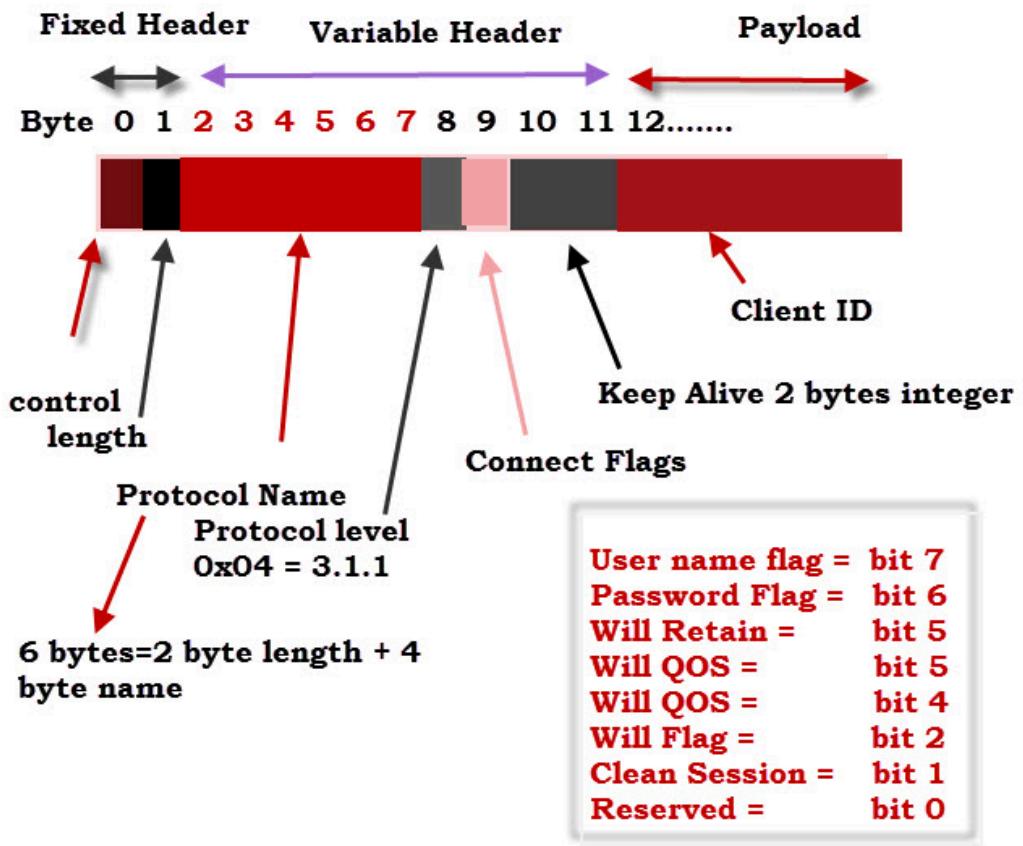
© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# MQTT Message Types and Hex Codes

# Message types	
CONNECT = 0x10	=16 decimal
CONNACK = 0x20	
PUBLISH = 0x30	
PUBACK = 0x40	
PUBREC = 0x50	
PUBREL = 0x60	
PUBCOMP = 0x70	
SUBSCRIBE = 0x80	=128 decimal
SUBACK = 0x90	
UNSUBSCRIBE = 0xA0	
UNSUBACK = 0xB0	
PINGREQ = 0xC0	
PINGRESP = 0xD0	
DISCONNECT = 0xE0	=224 decimal



# MQTT Connect Message Structure



`python_test`

**Connection message code example**

```
connecting client = "python_test", clean session = True  
sending command 0x10 sending flags = 0  
sending bytearray(b'\x10\x17\x00\x04MQTT\x04\x02\x00<\x00\x0bpyth  
on_test')
```

Total length = 23 decimal

connect flags clean  
session is True

keep alive = 60 seconds  
Ascii < =60

length of client id 0xb = 11 decimal

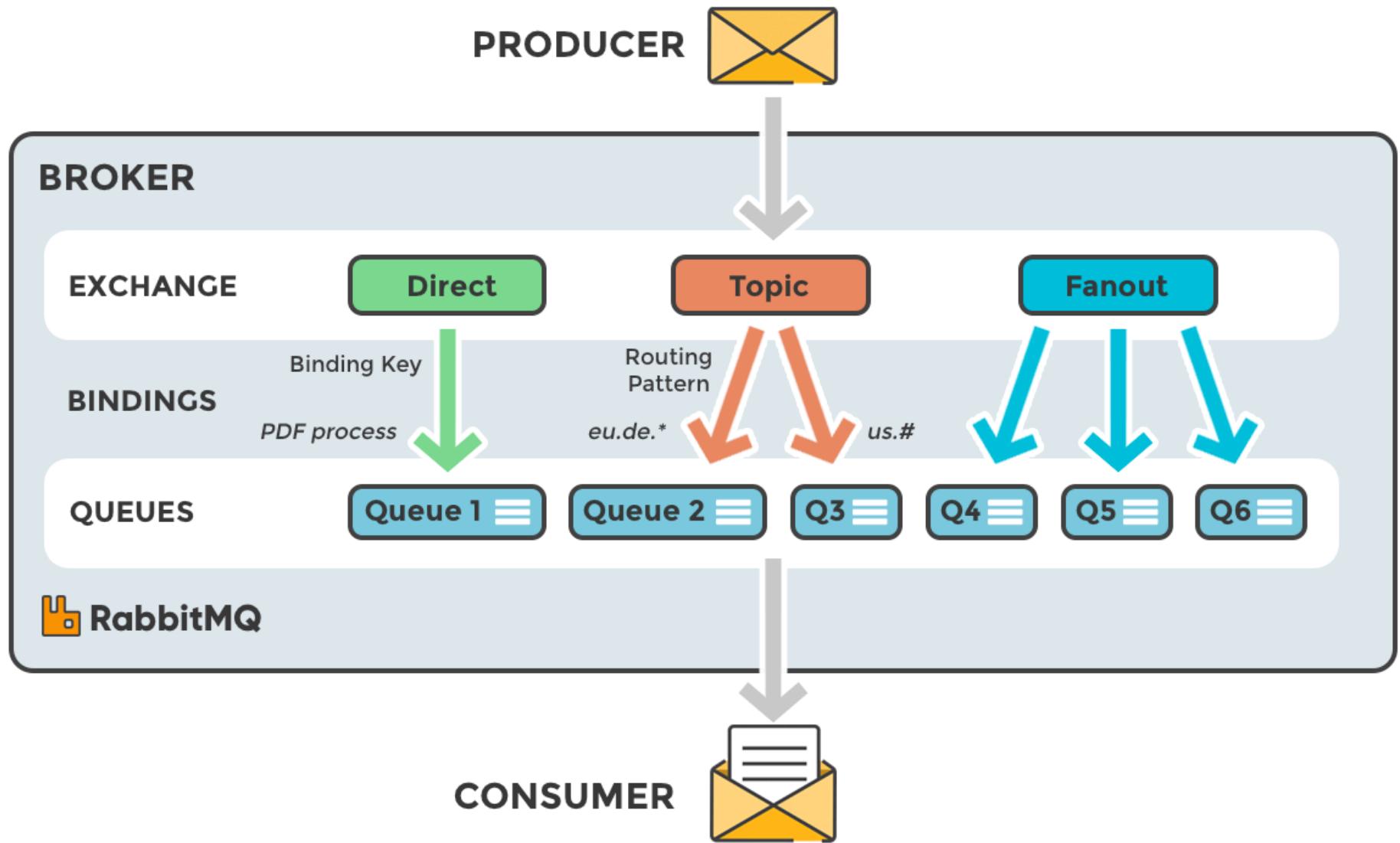


© Paul  
Attrib  
See [ht](#)

# AMQP / RabbitMQ

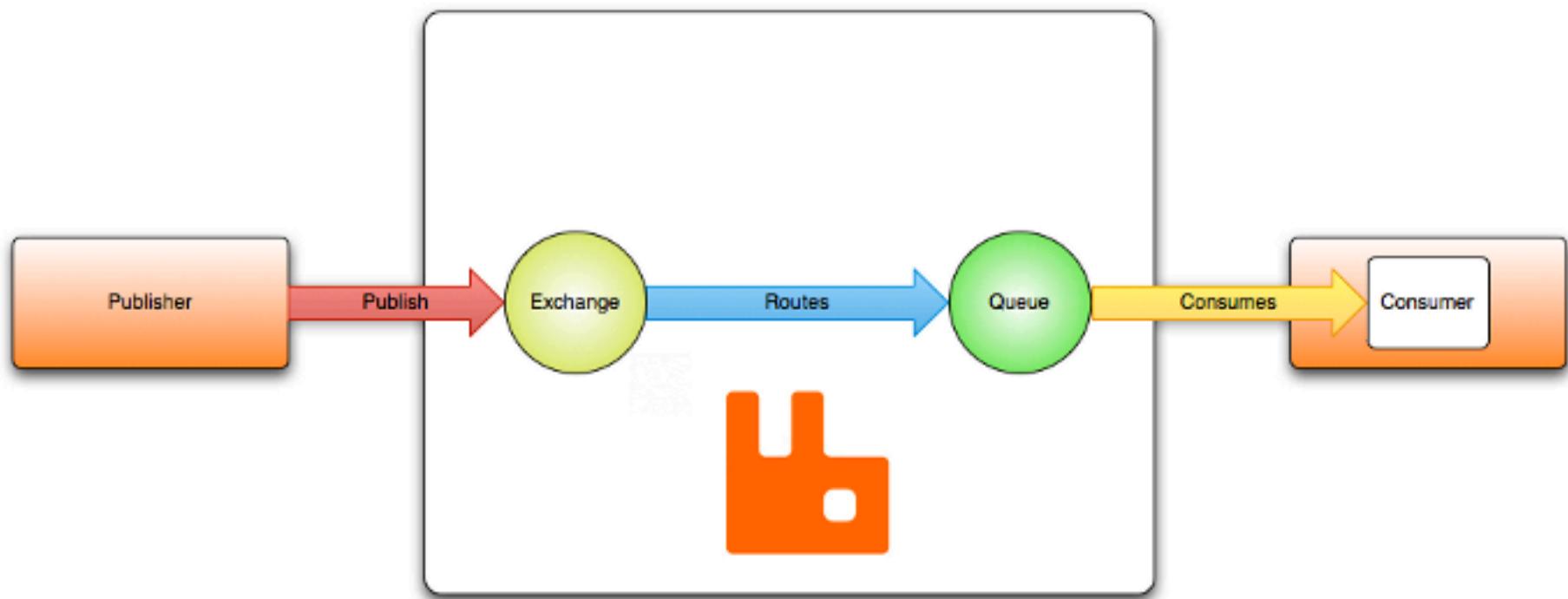
- AMQP is an advanced messaging protocol
  - Designed to meet more enterprise needs
  - Emerged from JP Morgan attempting to decouple from proprietary systems
- Standardised in OASIS
  - Although many implementations prefer 0-91 to 1-00





# RabbitMQ

"Hello, world" example routing



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# RabbitMQ

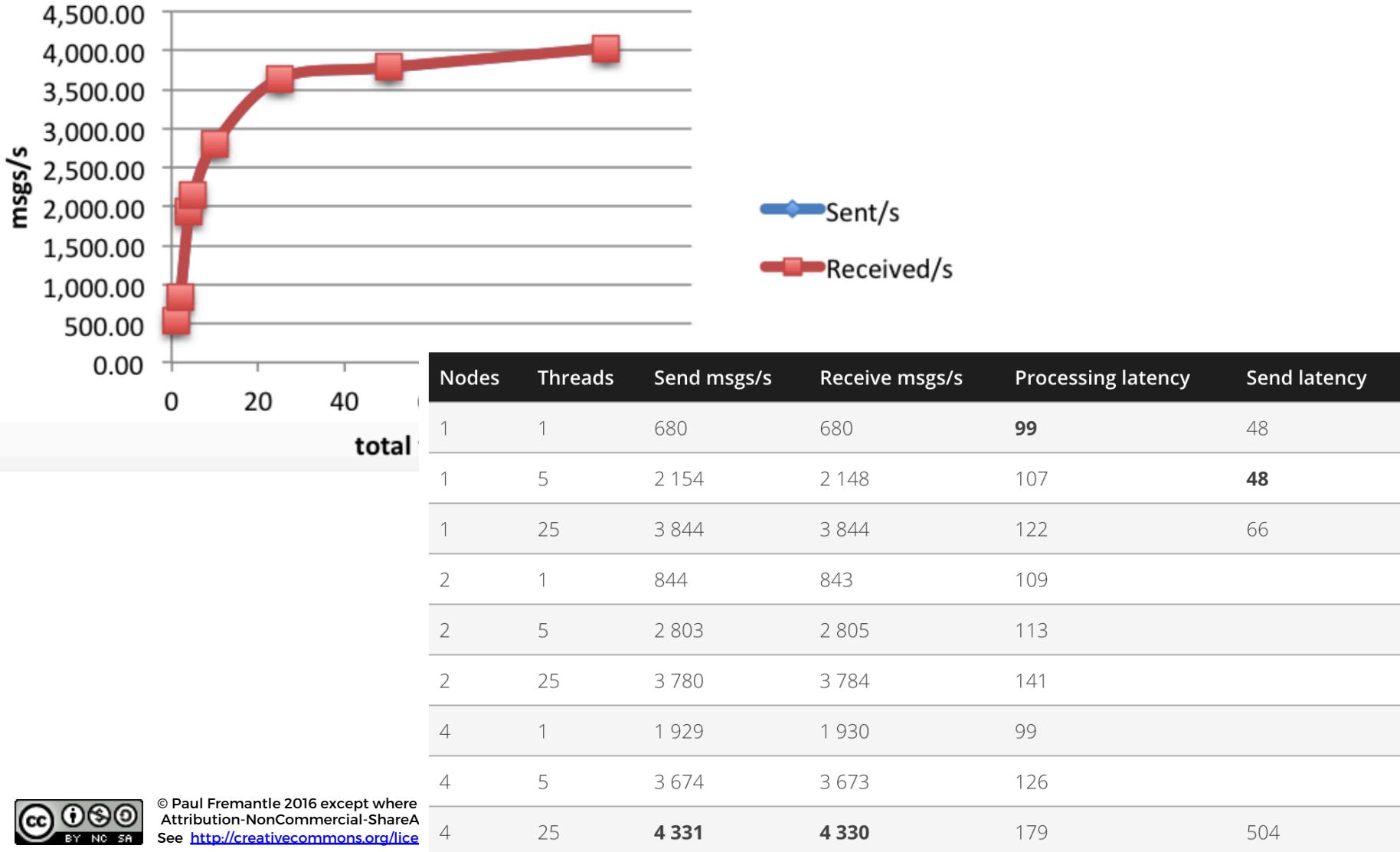
- Written in Erlang
- Designed to implement the AMQP 0-91 spec
  - Now extended to support STOMP, MQTT, AMQP 1.0 and HTTP
- Clusterable and high-performance
- Transactional



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# RabbitMQ performance

<https://softwaremill.com/mqperf/>



© Paul Fremantle 2016 except where  
Attribution-NonCommercial-ShareAlike  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# >1m msgs/sec

<https://content.pivotal.io/blog/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine>

User: queue  
RabbitMQ 3.2.3, Erlang R15B01 [Log out](#)

[Overview](#) [Connections](#) [Channels](#) [Exchanges](#) [Queues](#) [Admin](#)

## Overview

[▼ Totals](#)

Queued messages (chart: last minute) (?)

Ready: 2,343 msg  
Unacknowledged: 0 msg  
Total: 2,343 msg

Message rates (chart: last minute) (?)

Publish: 1,345,531/s  
Deliver (noack): 1,413,840/s

Global counts (?)

Connections: 12690 Channels: 12690 Exchanges: 194 Queues: 186 Consumers: 10304

[▼ Nodes](#)

Name	File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Uptime	Type
rabbit@b-rabbitmq-queue-1te2	445 262144 available	395 235837 available	4594 1048576 available	252MB 12GB high watermark	6.2GB 48MB low watermark	1h 33m	RAM



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# ActiveMQ / Artemis

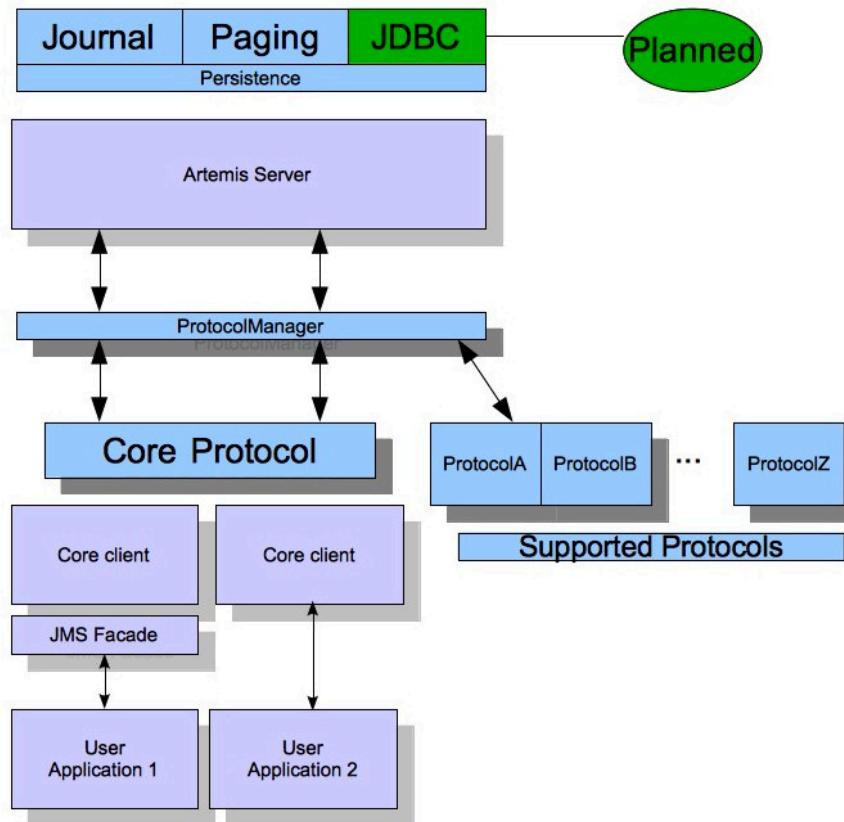


Figure 3.1 Artemis High Level Architecture



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Artemis

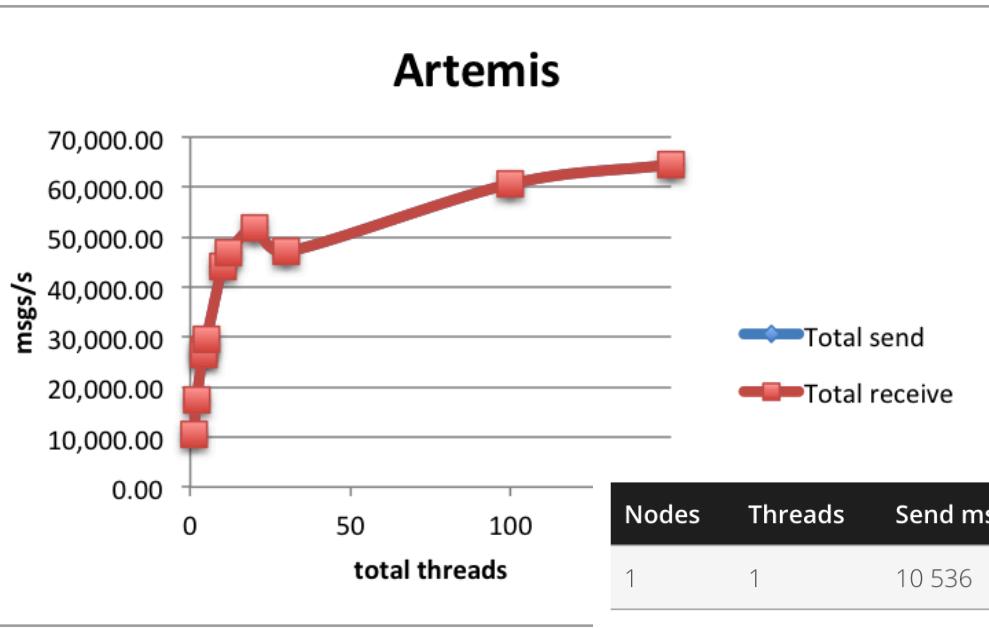
- **Supports multi-protocols:**
  - “JMS”, AMQP, STOMP, OpenWire, MQTT, REST
  - Highly available and clusterable
  - Written in Java



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Artemis Performance

<https://softwaremill.com/mqperf/>

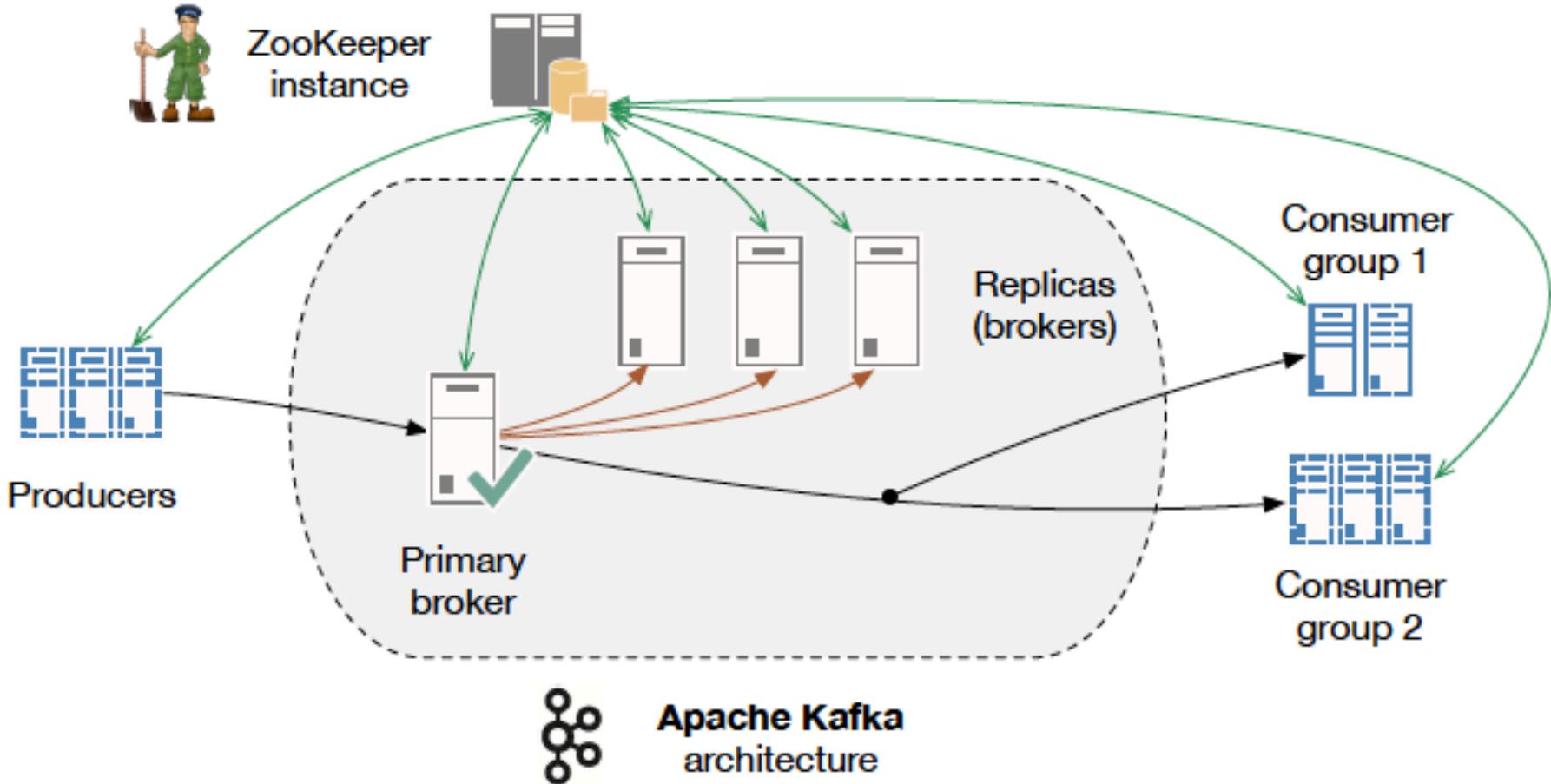


Nodes	Threads	Send msgs/s	Receive msgs/s	Processing latency	Send latency
1	1	10 536	10 536	48	45
1	5	29 476	29 476	48	47
2	1	17 515	17 515	46	46
2	5	44 003	44 003	46	47
4	1	27 197	27 197	47	47
4	5	51 724	51 720	46	47
4	25	60 619	60 619	62	48
6	5	47 078	47 082	<b>47</b>	48
6	25	<b>64 485</b>	<b>64 487</b>	122	<b>48</b>

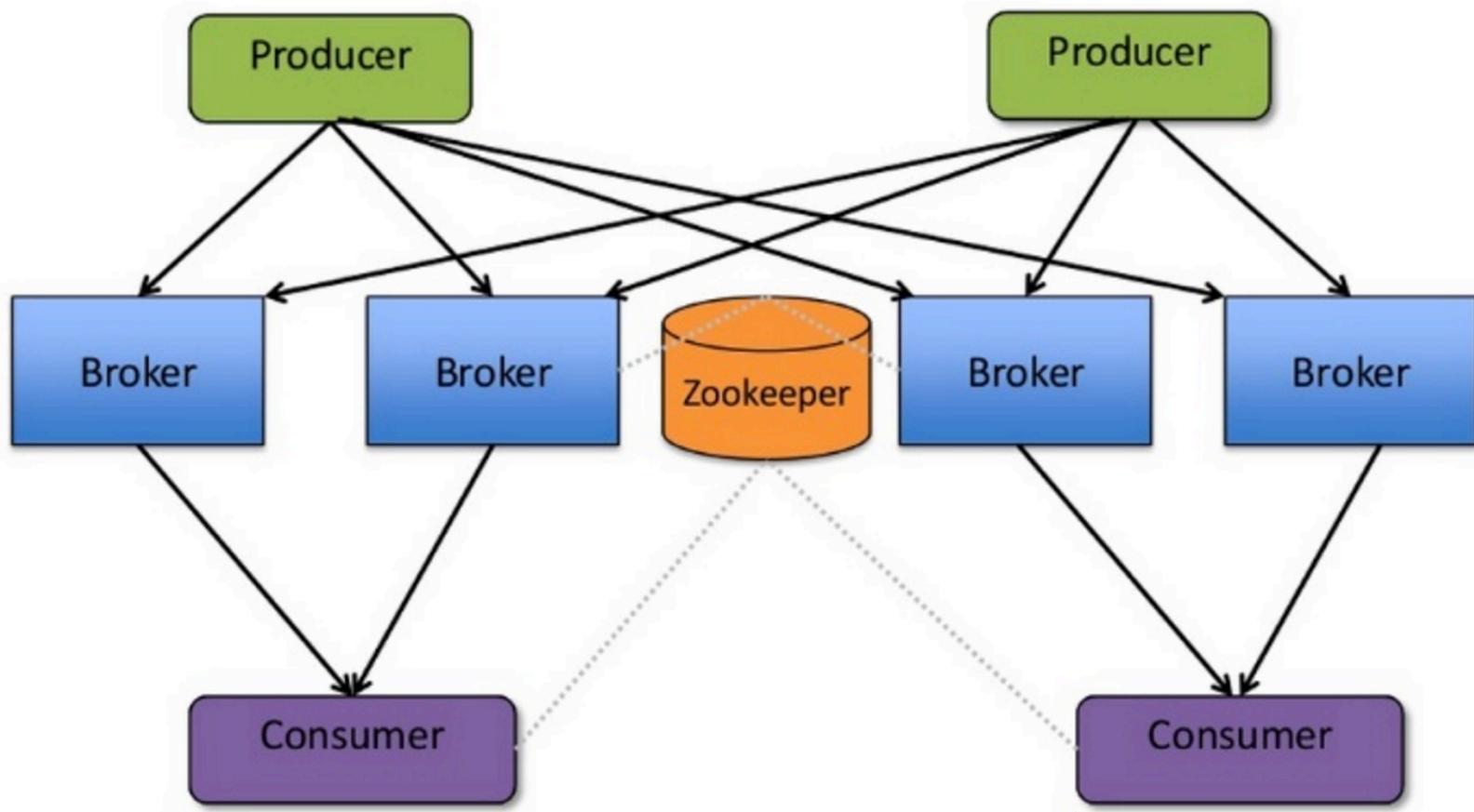


© Paul Fremantle 2016 except where cre  
Attribution-NonCommercial-ShareAlike  
See <http://creativecommons.org/licenses/>

# Apache Kafka



# Apache Kafka



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons  
Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Source: <http://www.slideshare.net/charmalloc/>

# Kafka

- Applying “big data” approaches to messaging:
  - Partitioning
  - Multiple brokers
  - Elastically scalable
  - Supports clusters of co-ordinated consumers
  - Automatic re-election of leaders



# Kafka exactly-once semantics



**Mathias Verraes**

@mathiasverraes

Follow

There are only two hard problems in distributed systems:  
2. Exactly-once delivery 1.  
Guaranteed order of messages  
2. Exactly-once delivery

RETWEETS LIKES

**6,775** **4,727**



10:40 AM - 14 Aug 2015



69



6.8K

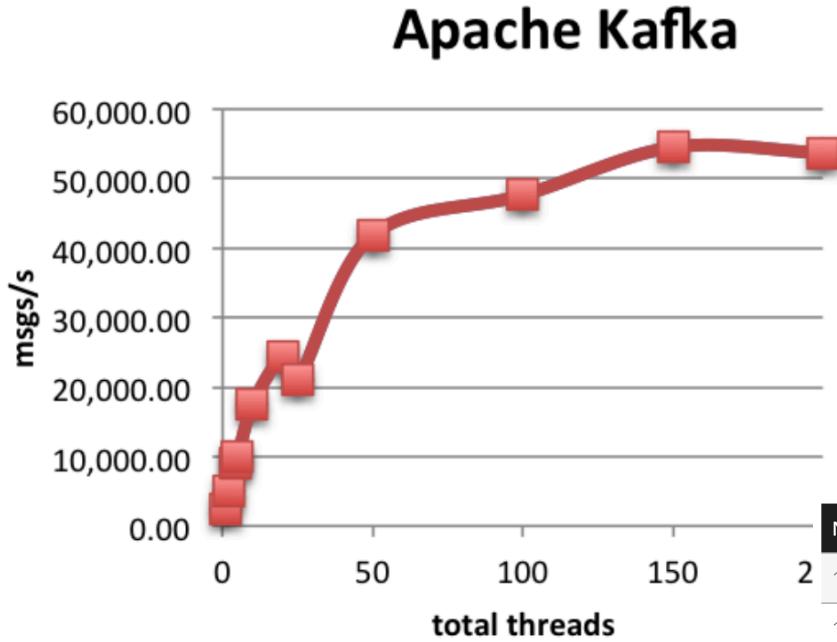


4.7K



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Kafka Performance



Nodes	Threads	Send msgs/s	Receive msgs/s	Processing latency	Send latency
1	1	2 391	2 391	48	48
1	5	9 917	9 917	48	48
1	25	20 982	20 982	46	48
2	1	4 957	4 957	47	
2	5	17 470	17 470	47	
2	25	41 902	41 901	45	48
4	1	9 149	9 149	47	
4	5	24 381	24 381	47	48
4	25	47 617	47 618	47	
6	25	<b>54 494</b>	<b>54 494</b>	<b>47</b>	<b>48</b>
8	25	53 696	53 697	47	48



# Questions?



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>