The Polyglot Rabbit

Adventures on OpenVMS with RabbitMQ, Erlang, and multi-protocol messaging

Brett Cameron
September 2014

Abstract

RabbitMQ (http://www.rabbitmq.com) is a popular 100% Erlang-based (http://www.erlang.org/) Open Source message queuing system that implements the Advanced Message Queuing Protocol (AMQP). It has been estimated that there are presently some 35,000+ production deployments of RabbitMQ across the globe, and this number is growing rapidly. Most of these deployments are business-critical, underpinning everything from internet-based pizza ordering systems through to providing the central nervous system for OpenStack-based cloud deployments. RabbitMQ natively supports versions 0.8.0 and 0.9.1 of AMQP; however, a somewhat overlooked capability of RabbitMQ is its ability to also readily provide support via a flexible plugin architecture for a variety other popular Open Source message queuing protocols, including STOMP, MQTT, ZeroMQ, and RESTful messaging via the RabbitHub plugin. Most good message queuing protocols share many features in common; however some are better suited to a particular set of use cases than others. This ability of RabbitMQ to be able to seamlessly receive and propagate messages simultaneously via multiple protocols is an extremely powerful facility, and one that affords great flexibility. For example, it means that it is possible to use the most appropriate protocol for a particular function or to simultaneously use different protocols to disseminate the same data to different types of users via the most appropriate protocol without having to develop and maintain any separate gateway components. As a consequence of work done by the speaker to port Erlang to OpenVMS and to develop OpenVMS-friendly client API's, it is possible to fully utilize this message queuing technology in an OpenVMS environment. In this talk the presenter will discuss the multi-protocol features of RabbitMQ, how the capabilities of Erlang have been utilised to implement the powerful RabbitMQ plugin architecture, and how these capabilities of RabbitMQ can be used to implement a robust and highly scalable multi-protocol Open Source-based messaging hub.

About me

Brett Cameron currently works as a senior architect with HP's corporate Cloud Services group, focusing on the design and implementation of message queuing and related integration services for customers and for internal use. Brett lives in Christchurch, New Zealand, and has worked in the software industry for some 22 years. In that time he has gained experience in a wide range of technologies, many of which have long since been retired to the software scrapheap of dubious ideas. In recent years Brett has specialized in systems integration, and the design and implementation of large distributed systems for HP's enterprise customers. This work has seen Brett get involved in the research and development of low-latency and highly scalable messaging solutions for the Financial Services sector running on HP platforms, and as a consequence of this work, Brett has been involved in several interesting Open Source projects, and he has been responsible (or should that be irresponsible) for porting various pieces of Open Source software to the HP OpenVMS platform. Brett holds a doctorate in chemical physics from the University of Canterbury, and still maintains close links with the University, working as a part time lecturer in the Computer Science and Electronic and Computer Engineering departments. In his spare time, Brett enjoys listening to music, playing the guitar, and drinking beer.

AGENDA

- **Introduction**
- AMQP
- RabbitMQ
- RabbitMQ plugins
- Multi-protocol Open Source messaging hub
- Summary
- Questions

Introduction

- Brief overview of RabbitMQ and AMQP
- RabbitMQ plugins
  - Overview (what, where, why, how…)
  - Plugin architecture
- Using RabbitMQ and various plugins to create a multi-protocol Open Source OpenVMS-based messaging hub
  - Setting up a RabbitMQ cluster on OpenVMS
  - Cluster performance/scalability
  - Multi-protocol use-cases/examples
  - Futures
- Wrap-up/questions

AGENDA

- Introduction
- **AMQP**
- RabbitMQ
- RabbitMQ plugins
- Multi-protocol Open Source messaging hub
- Summary
- Questions

AMQP introduction

- AMQP (Advanced Message Queuing Protocol) is an open standard application layer protocol for message oriented middleware
  - It is an open protocol (like TCP, HTTP, SMTP, and so on)
- The defining features of AMQP are:
  - Message orientation
  - Queuing
  - Routing (including point-to-point and publish-and-subscribe)
  - Reliability
  - Security
- AMQP mandates the behaviour of the messaging provider and client
  - Implementations from different vendors are truly interoperable
  - Previous attempts to standardise middleware have focussed at the API level
    - This approach did not create interoperability
  - Instead of merely defining an API, AMQP defines a wire-level protocol
    - A wire-level protocol is a description of the format of the data that is sent across the network as a stream of octets
    - Any tool that can create and interpret messages that conform to the defined wire-level protocol can interoperate with any other compliant tool, irrespective of implementation language

History

- AMQP was originally designed to provide a vendor-neutral protocol for managing the flow of messages across an enterprise's business systems
- AMQP was developed from mid-2004 to mid-2006 by JPMorgan Chase and iMatix
  - iMatix also developed the original AMQP implementation (OpenAMQ)
  - See http://www.openamq.org
- JPMorgan Chase and iMatix documented the protocol and assigned it to a working group that included Red Hat, Cisco Systems, TWIST, IONA, and iMatix
- As of June 2014, the working group consisted of:
  - Bank of America, Bloomberg Finance, Credit Suisse, Deutsche Börse, JPMorgan Chase Bank, Informatica, LogMeIn, Microsoft, Red Hat, SITA, Software AG, US Department of Homeland Security, WSO2
  - Mixture of Financial Services companies and technology providers
  - See http://www.amqp.org
- Although AMQP originated in the financial services industry, it has general applicability to a broad range of middleware problems

History

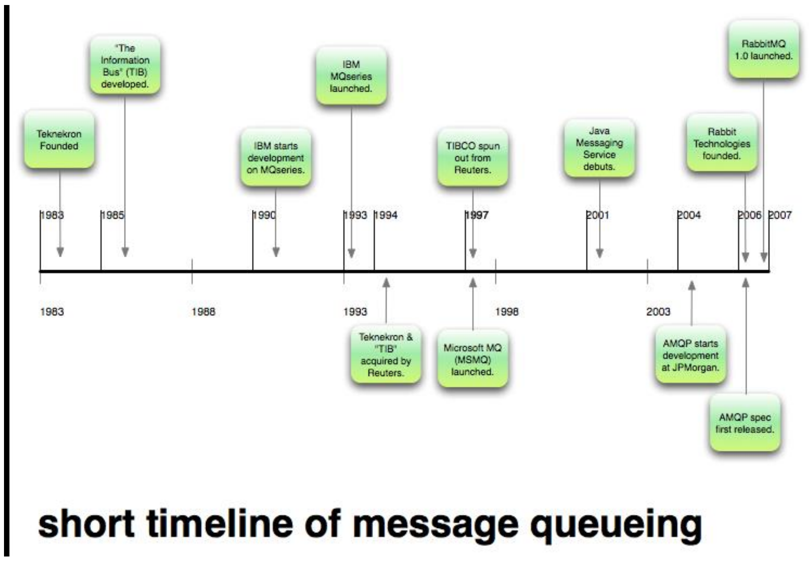

Teknekron Founded
"The Information Bus" (TIB) developed.
IBM starts development on MQseries.
IBM MQseries launched.
TIBCO spun out from Reuters.
Java Messaging Service debuts.
Rabbit Technologies founded.
RabbitMQ 1.0 launched.
1983
1985
1990
1993
1994
1997
2001
2004
2006
2007
1983
1988
1993
1998
2003
Teknekron & "TIB" acquired by Reuters.
Microsoft MQ (MSMQ) launched.
AMQP starts development at JPMorgan.
AMQP spec first released.
short timeline of message queueing

Diagram adapted from http://www.manning.com/videla/

Motivation

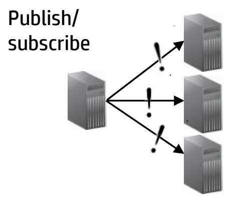
- AMQP was born of frustration…
- Message oriented middleware needs to be everywhere in order to be useful, but…
  - Traditionally dominant solutions are typically very proprietary
    - They are frequently too expensive for everyday use
    - They invariably do not interoperate
  - The above *issues* with proprietary solutions have resulted in numerous home-grown developments
    - Custom middleware solutions
    - Custom adaptors
  - The net result for a large enterprise is *middleware hell*
    - Hundred's of applications, thousand's of links
    - Every other connection is different
    - Massive waste of effort
      - Costly to implement
      - Costly and difficult to maintain

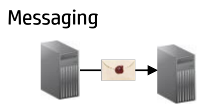
The AMQP model – key features

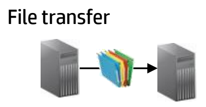
Key features of AMQP include:

- Queuing with strong delivery assurances
- Event distribution with flexible routing
- Large message capability (gigabytes)
- Global addressing scheme (email-like)
- Meets common requirements of mission-critical systems
- Service oriented

AMQP aims to become *the* solution for enterprise messaging: Any language, any model, any payload, any platform, reliable, interoperable, manageable, performant, scalable. Big aspirations!







Comparison with some other protocols

Protocol	Comments
SMTP	Unreliable, slow
HTTP	Synchronous, semi-reliable, no routing
XMPP	No delivery fidelity or queue management
FTP	Point to point, transient, does not work well with NAT/SSL
MQ	Exactly once
TCP	At least once, reliable but short lived, no application-level state management
UDP	Fast but has no delivery guarantees

AMQP can accommodate all of the above as use-cases… and switch between them (open, ubiquitous, and adaptable)

The AMQP model – key components

- **Message layer**
  - Exchanges
    - Software *switches* or *routers*
    - Message producers publish messages to exchanges
    - Three basic types:
      - Direct, fan-out, and topic
  - Queues
    - Are (essentially) FIFO containers that hold messages
    - Can be memory only or backed by disk
    - Have various convenience options on who can use them, automatic clean-up, and so on
  - Bindings
    - A binding associates an exchange with a queue
    - Messages published to an exchange are routed by the exchange to any bound queues
- **A peer-to-peer wire protocol**
- **Message broker**
  - Applications connect to a broker to participate in the AMQP network
  - The connection is used to establish a session
    - Sessions provide state between connections, establish identity, ease failover
  - Connections are further subdivided into channels
    - Multiple threads of control within an application can share one connection

Note that we are really only talking about AMQP 0.8 and 0.9.1 here. Things are a bit different with AMQP 1.0, which only defines the network wire-level protocol for the exchange of messages between two endpoints. The 1.0 standard supports exchanging messages in peer-to-peer fashion or via brokered topologies.

Scales of deployment

AMQP implementations can cover deployment at different levels of scale ranging from trivial to the mind-boggling… no job too big or too small.

Type of deployment	Possible scenario
• Developer/casual use	• 1 server, 1 user, 10 queues, 1 message per second
• Production application	• 2 servers, 10-100 users, 10-50 queues, 25 messages per second
• Departmental mission critical application	• 4 servers, 100-500 users, 50-100 queues, 250 messages per second
• Regional mission critical application	• 16 servers, 500-2,000 users, 100-500 queues and topics, 2,500 messages per second
• Global mission critical application	• 64 servers, 2K-10K users, 500-1000 queues and topics, 25,000 messages per second
• Market data (trading)	• 200 servers, 5K users, 10K topics, 250K messages per second

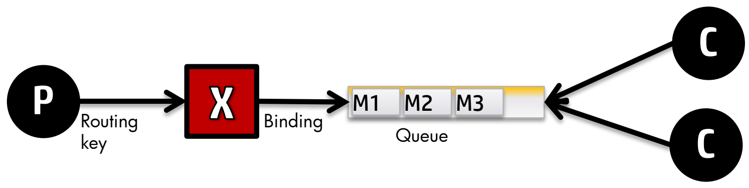
As well as volume, the latency of message transfer is often important; AMQP implementations can deliver messages with latencies of less than 200μs

Some typical usage patterns

- Direct exchanges
  - Simple point-to-point queue delivery
  - Abstracted point-to-point queue
  - Load-balanced point-to-point queue delivery
  - RPC
- Fan-out and topic exchanges
  - Dynamic (non-persistent) publish/subscribe delivery
  - Durable (persistent) publish/subscribe delivery
- Inter-network connectivity

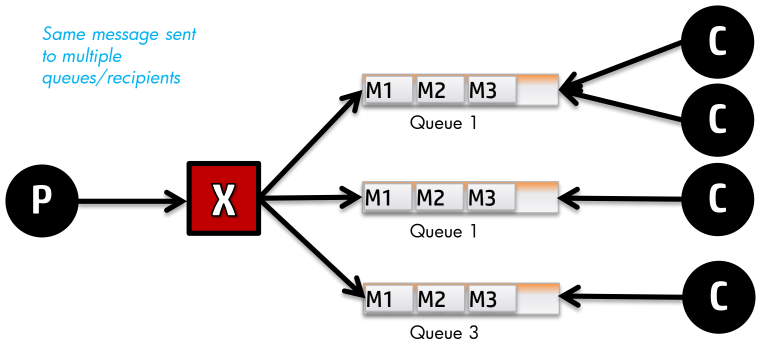
Let's briefly look at the three primary exchange types…

Direct exchanges



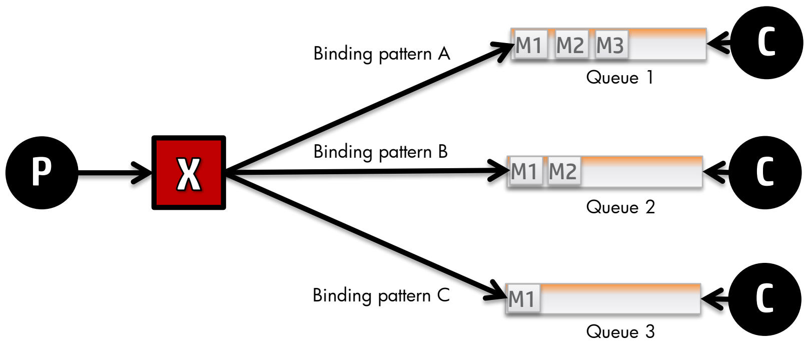
- Point to point messaging
- A single message that needs to be sent to a single recipient
- Can have multiple consumers (load will be balanced across them)

Fan-out exchange



- A fan-out exchange is a "publish-subscribe" mechanism
- Used to send a single message to multiple recipients (very similar to an email distribution list)
- Any queue bound to a fan-out exchange will receive any message sent to the exchange
- Message consumption of each queue will be dependent on the number of consumers and speed of message consumption

Topic exchange



- Content-based routing mechanism
- Messages posted to queues based on binding pattern (PCRE used)
- Very powerful mechanism

AGENDA

- Introduction
- AMQP
- **RabbitMQ**
- RabbitMQ plugins
- Multi-protocol Open Source messaging hub
- Summary
- Questions

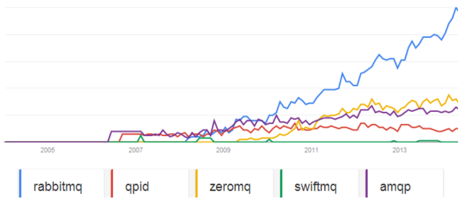
What is RabbitMQ?

RabbitMQ
Open Source Enterprise Messaging

- A powerful Open Source message broker (message-oriented middleware)
  - The leading (and arguably the most popular) implementation of AMQP
  - Provides a robust and flexible messaging platform designed to interoperate with other messaging systems
  - See http://www.rabbitmq.com for more information
- A multi-protocol broker engine implemented in Erlang for exceptional scalability and fault-tolerance
  - 100% Erlang/OTP (an inspired move)
- RabbitMQ essentially comprises the following components:
  - The RabbitMQ broker (supports AMQP 0.8 and 0.9.1, and supports AMQP 1.0 via a plugin)
  - Adapters for HTTP, ZeroMQ, STOMP, MQTT, and other protocols
  - AMQP client libraries for Erlang, Java, .NET, and C/C++
    - AMQP clients for numerous other languages are available from other vendors and/or the Open Source community
    - Python, Ruby, PHP, Clojure, Node.js, Go, … just about any language you can think of
  - Large assortment of useful plugins

Very brief RabbitMQ history

- 2006
  - Rabbit Technologies Ltd. founded and the first version of RabbitMQ was born
  - RabbitMQ source code is released under the Mozilla Public License
- 2006 – today
  - Rapid adoption (good product, catchy name, excellent support), incremental rollout of new features
- April 2010
  - Rabbit Technologies Ltd. was acquired in April 2010 by VMware
- April 2013
  - VMware and EMC spin out some of their cloud-related computing projects - including RabbitMQ - into an autonomous company, Pivotal Software, Inc. (http://www.gopivotal.com/)



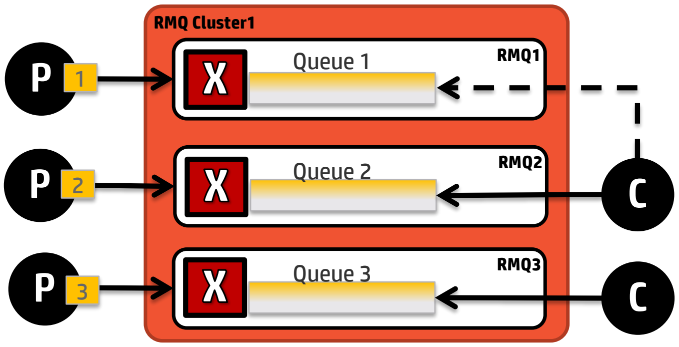
RabbitMQ business-critical features

In the next few slides we'll have a quick look at:

- Scalability through clustering
- High-availability
- Management and monitoring
- Security
- T-shirt sizes to suit all requirements

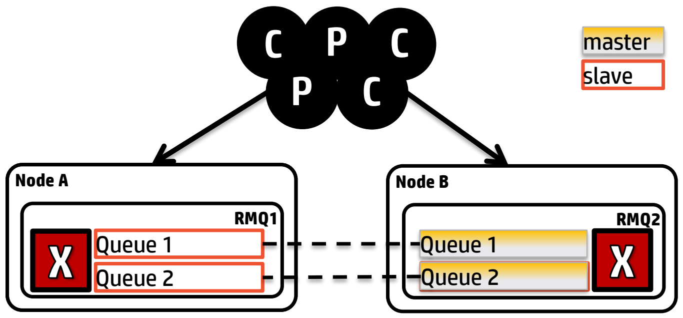
RabbitMQ clustering

Scalability and fault tolerance.



- Clustering can be used to elastically scale the messaging infrastructure
- Producers and consumers can send and retrieve messages by connecting to any broker in the cluster, allowing load to be spread evenly and avoid having a single point of failure for all queues
- A queue only lives on one cluster node (actually, with HA queue replication, it's not quite that simple… see next slide)
- Consumers are able to retrieve any message from any queue in the cluster, even if they are not connected to the server that owns the queue
- By using clustering, you can add new queues on new brokers without re-configuring clients

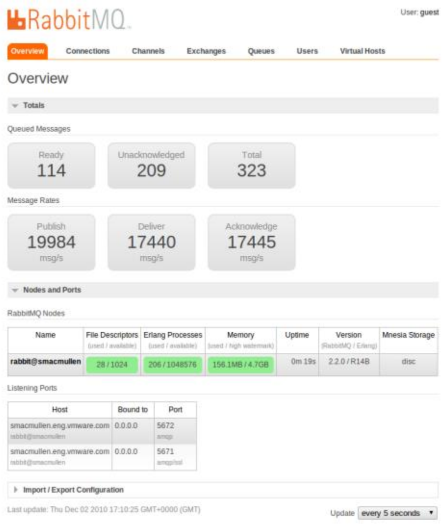
High availability



- Highly available, mirrored queues (and replication policies)
- Potentially better performance and less downtime
- Downside is more network traffic inside the RabbitMQ cluster as messages are replicated to slave queues on message send
- Need to add a little code to some clients to make them HA cluster aware (so they will try a backup server when a primary connection fails)

Management and monitoring

- The RabbitMQ Management plugin provides an HTTP-based RESTful API for management and monitoring the RabbitMQ broker
- Two tools are provided that use this API:
  - A powerful browser-based UI
  - A powerful command line tool (`rabbitmqadmin`)
  - `rabbitmqadmin` can perform the same actions as the web-based UI, and is convenient for use when scripting
- Or you can use the API to develop your own tools…



Overview

Queued Messages

Ready	Unacknowledged	Total
114	209	323

Message Rates

Publish	Deliver	Acknowledge
19984 msg/s	17440 msg/s	17445 msg/s

RabbitMQ Nodes

Name	File Descriptors (used / available)	Erlang Processes (used / available)	Memory (used / high watermark)	Uptime	Version (RabbitMQ / Erlang)	Mnesia Storage
rabbit@smacmullen	28 / 1024	206 / 1048576	156.1MB / 4.7GB	0m 19s	2.2.0 / R14B	disc

Listening Ports

Host	Bound to	Port
smacmullen.eng.vmware.com rabbit@smacmullen	0.0.0.0	5672 amqp
smacmullen.eng.vmware.com rabbit@smacmullen	0.0.0.0	5671 amqp/ssl

Last update: Thu Dec 02 2010 17:10:25 GMT+0000 (GMT)

Security

Virtual hosts (vhosts)

- When an AMQP client establishes a connection to an AMQP server, it specifies a virtual host within which it intends to operate
- A first level of access control is enforced at this point, with the server checking whether the user has any permissions to access the virtual hosts, and rejecting the connection attempt otherwise
- Vhosts are essentially independent RabbitMQ message brokers within the RabbitMQ process with their own queues, exchanges, permissions, and so on

Users and permissions

- A second level of access control is enforced when certain operations are performed on resources
  - Resources (exchanges and queues) are named entities inside a particular virtual host
- RabbitMQ distinguishes between *configure*, *write,* and *read* operations on a resource
  - *Configure* operations create or destroy resources, or alter their behavior
  - *Write* operations inject messages into a resource
  - *Read* operations retrieve messages from a resource

Some other AMQP implementations

- OpenAMQ (http://www.openamq.org)
  - Original Open Source implementation of AMQP, written in C by iMatix
    - No longer supported (sadly)
  - Runs on Linux, AIX, Solaris, Windows, OpenVMS, HP-UX
  - Includes broker, APIs in C/C++ and Java JMS, remote administration shell, scripting, federation, failover, and AMQP-over-HTTP via the RestMS protocol
- Apache Qpid (http://qpid.apache.org/)
  - A project in the Apache Foundation
  - Includes broker and APIs that support C++, Ruby, Java, JMS, Python and .NET; AMQP 1.0 support
- Red Hat Enterprise MRG (http://www.redhat.com/mrg/)
  - Rich set of features including management, federation, Active-Active clustering, and APIs for C++, Ruby, Java, JMS, Python .NET; AMQP 1.0 support
- SwiftMQ (http://www.swiftmq.com/)
  - An enterprise-grade JMS messaging product with full support for AMQP 1.0
- StormMQ (http://stormmq.com/)
  - A cloud-hosted messaging service based on AMQP