AGENDA

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

RabbitMQ plugins

RabbitMQ provides a highly flexible plugin system

- Provides a powerful facility for extending or supplementing the capabilities of the RabbitMQ broker
- Provides an interface into core RabbitMQ functionality

What can you do with plugins?

- Add support for other messaging protocols
  - STOMP
  - WebSockets
  - MQTT
  - HTTP/REST
  - ZeroMQ
- Alternative authentication mechanisms
  - LDAP
  - SSL
  - Kerberos
  - Could create something that hooks into `SYSUAF`
- Implement alternative message stores
- Add extra functionality to the RabbitMQ broker
  - RabbitMQ Management plugin
  - RabbitMQ Shovel plugin
  - Federation plugin
- Create new exchange types
  - Random exchange
  - Consistent hash exchange
  - Riak exchange
  - Global fanout exchange
  - Recent history exchange
  - Sharding exchange (partition messages across sharded queues)

See http://www.rabbitmq.com/plugins.html for a more complete list of official and community-developed plugins.

Why write a plugin?

- Enable your application to access internal RabbitMQ functionality that is not exposed via the AMQP interface
- Plugins run in the same Erlang VM as the RabbitMQ broker
  - This may improve performance for certain applications
- Can simplify deployment
  - Plugins can be packed with the broker as a single deployable unit

Plugins must be written in Erlang and run in the same Erlang VM as the broker.

Comments on writing plugins

- Not going to go into the gory details
  - The RabbitMQ team provides the *RabbitMQ Public Umbrella* to assist with plugin development
  - See http://www.rabbitmq.com/plugin-development.html and http://manning.com/videla/
  - Can also do *rebar*-based build for some plugins (see https://github.com/brc859844/rabbithub for an example of this approach)
    - Rebar is a powerful Erlang build tool (see https://github.com/basho/rebar)
  - Study and borrow code from existing plugins!
- Desirable to have a reasonable knowledge of RabbitMQ internals
  - API's
  - Boot steps (order in which core RabbitMQ components and plugins are started)
  - ...
- Beware of minor changes to internals between versions
  - Can break plugins
  - Far less of a problem with newer versions of RabbitMQ, but be careful

AGENDA

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

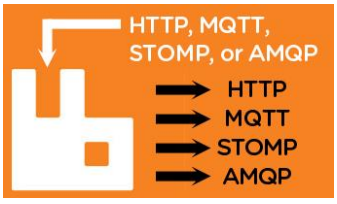
Multi-protocol Open Source messaging hub

- Multi-protocol messaging
- Cluster configuration
- Multi-protocol messaging examples

Multi-protocol messaging

Via its powerful plugin mechanism the RabbitMQ broker is able to support other message queuing protocols in addition to AMQP

- What does this mean and why should you care?



Flexibility

The notion of a "one-size-fits-all" messaging protocol is flawed

- Some messaging protocols are optimized for low-latency transmission of small messages
  - Others are designed for efficient transmission of large messages
- Some messaging protocols sacrifice performance for reliability
  - Others favor performance at the risk of occasional message loss
- …

The world is not uniform

- There are clearly advantages to being able to communicate with people in multiple languages
  - In an analogous fashion there are advantages to message queuing software being able to accommodate and seamlessly map between different message queuing protocols

Any attempt to accommodate all possible messaging scenarios within the scope of a single ubiquitous protocol must entail some level compromise

- The resultant protocol will be less well-suited to one or more messaging scenarios over a less generic protocol that is optimally designed for a specific use case

Adaptability

Most good message queuing protocols share many features in common

- However some are better suited to a particular task than others

Change is constant

- What may seem like a good idea today may not be such a good idea tomorrow
- If software cannot readily (and potentially rapidly) adapt to change then it will become extinct

Other protocols supported by RabbitMQ

- STOMP
  - http://stomp.github.com
  - https://www.rabbitmq.com/stomp.html (supports STOMP 1.0, 1.1, and 1.2)
- MQTT (MQ Telemetry Transport)
  - http://mqtt.org/
  - http://www.rabbitmq.com/blog/2012/09/12/mqtt-adapter/
- ZeroMQ
  - http://www.zeromq.org/
  - https://github.com/rabbitmq/rmq-0mq/wiki
  - Plugin code needs some updating to work with current RabbitMQ versions
- HTTP/REST via the RabbitHub plugin
  - https://github.com/tonyg/rabbithub
  - Recently updated to be compatible with current RabbitMQ versions
- Web-STOMP (STOMP + WebSockets)
  - http://www.rabbitmq.com/blog/2012/05/14/introducing-rabbitmq-web-stomp/
- ... and a few others

Whilst there are some limitations in terms of fully mapping some of the protocols to the AMQP 0.9.1 model, these limitations are generally insignificant compared to the capabilities that the plugins provide.

STOMP

Simple (or Streaming) Text Orientated Messaging Protocol (http://stomp.github.com/)

- A simple to use and easy to implement protocol
- Designed for asynchronous message passing between clients via mediating servers
- Defines a text-based wire format for messages passed between clients and servers
- Lightweight alternative to other open messaging protocols applicable to simple messaging scenarios
- Maps very well to AMQP 0.9.1
- Clients available for numerous languages

- Have developed a STOMP client API for OpenVMS that can be used with any OpenVMS 3GL (click here to look at some code examples)

MQTT

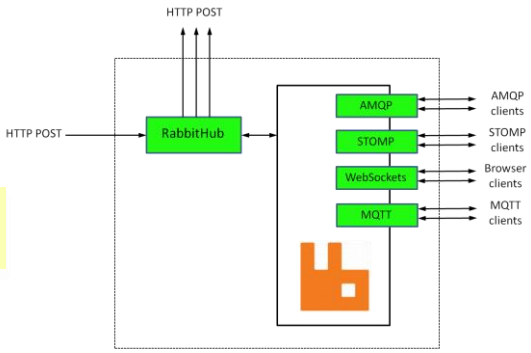
- Designed to facilitate the transfer of telemetry-style data to a server or message broker from pervasive devices over high-latency or otherwise constrained networks
  - Sensors and actuators
  - Mobile phones
  - Embedded systems on vehicles
  - Laptops and other computing devises
  - See http://www.mqtt.org
- Invented by Andy Stanford-Clark of IBM, and Arlen Nipper of Cirrus Link Solutions
- Maps well to AMQP 0.9.1
- Clients available in various languages
  - The Paho project (C and Java clients; see http://www.eclipse.org/paho/)
  - Paho C client ported to OpenVMS 8.4
    - Not so easy to port to earlier versions as code uses certain CRTL features available only on 8.4
    - Have also ported the Mosquito MQTT broker (http://mosquitto.org/) to OpenVMS…

MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. For example, it has been used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers, and in a range of home automation and small device scenarios. It is also ideal for mobile applications because of its small size, low power usage, minimized data packets, and efficient distribution of information to one or many receivers…

http://www.mqtt.org

The RabbitHub plugin

- A simple web-hook-based pub/sub mechanism that provides an HTTP-based interface to RabbitMQ
- Gives every AMQP exchange and queue hosted by the broker two URL's:
  - One to use for delivering messages to the exchange or queue
  - One to use to subscribe to messages forwarded on by the exchange or queue
- Subscriptions supply a callback URL to RabbitHub that is used to deliver messages via HTTP POST
- Provides a generally useful, easy to use, and efficient means of interacting with RabbitMQ in a RESTful manner



When used in combination with other protocol plugins such as those for STOMP, MQTT, and WebSockets, the result is an extremely powerful and flexible messaging solution.

Subscribing to a feed with RabbitHub

- Subscribing to a feed (topic) is straightforward:

```
$ curl -d -
"hub.mode=subscribe&hub.callback=http://localhost:4567/sub&hub.topic=foo&hub.lease_seconds=600" -
http://guest:guest@localhost:15670/subscribe/x/amq.direct
```

  - Any messages published to the exchange "`amq.direct`" with routing key "`foo`" will be forwarded by RabbitHub to the callback URL `http://localhost:4567/sub`
  - This subscription will expire after 600 seconds
  - Subscriptions can be to a queue or to an exchange
    - When subscribing to an exchange RabbitHub creates an exclusive queue for the subscriber and binds it to the exchange using the supplied binding details (namely the specified topic)
    - It is possible to subscribe to any RabbitMQ exchange type
  - Messages can be published using other protocols supported by RabbitMQ (AMQP, STOMP, MQTT, …)
    - RabbitHub does not care how messages get into queues from which it is servicing subscribers
    - More on this later…

For more information on RabbitHub see http://www.youtube.com/watch?v=09sFFI91NSc

Open standards

AMQP, HTTP, ZeroMQ, MQTT, STOMP

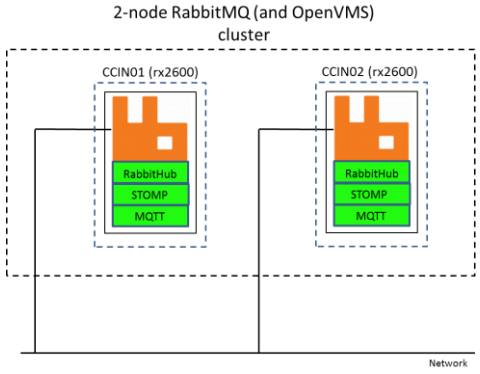
- All are essentially Open Standards

One of the key motivations behind the creation of AMQP was to escape the "*middleware hell*" (John O'Hara, father of AMQP)

- Multiple proprietary message queuing products scattered across the enterprise
  - License costs
  - Different proprietary protocols
  - Gateways and adapters required to integrate between them
    - Development costs
    - Costly on-going maintenance and support
    - …

There is a significant difference between developing gateways and adapters to bridge proprietary messaging technologies and having a core open messaging technology such as RabbitMQ that can be readily extended via plugins to facilitate multi-protocol communication amongst a set of Open Standards-based messaging protocols.

Simple cluster configuration

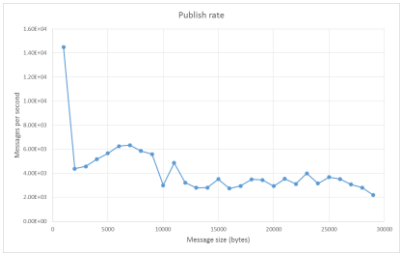


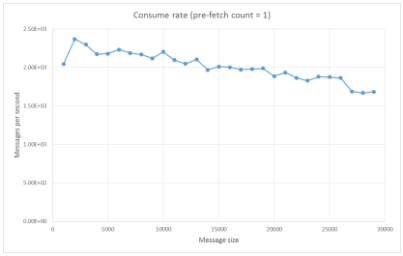
- A simple 2-node RabbitMQ cluster
- OpenVMS nodes also happen to be clustered, but this is not a requirement
- OpenVMS 8.4
- Erlang/OTP 16A
- RabbitMQ 3.3.1
- RabbitMQ STOMP 3.3.1 plugin
- RabbitMQ MQTT 3.3.1 plugin
- RabbitHub plugin (version 0.0.1)

A small 2-node RabbitMQ cluster was created for experimentation purposes using two old rx2600 servers running OpenVMS 8.4 and a rough port of Erlang 16A. Whilst such a configuration is not required in order to experiment with the different protocol plugins, using a cluster with plugins enabled on multiple nodes serves as a means of demonstrating the perhaps obvious fact that messages published via one protocol to one cluster node can be consumed via different protocols from another cluster node. Cluster configuration is largely as per standard clustering instructions at http://www.rabbitmq.com/clustering.html , using DCL versions of the UNIX shell scripts.

Cluster performance/scalability

- RabbitMQ cluster performance on OpenVMS is currently "satisfactory"
  - Mileage may vary, depending on messaging use-case
  - Largely limited by performance of Erlang/OTP TCP/IP driver code on OpenVMS
  - Needs work
- AMQP message rates on the order of several thousand messages per second are readily achievable
  - For both publish and consume operations
  - Non-persistent messages
  - Strong function of message size
  - Illustrated results obtained using pair of old rx2600 servers
  - Need to watch pre-fetch counts and memory
- Keep in mind that not all protocols and clients are born equal
  - For example HTTP is inherently request/response





Examples of multi-protocol messaging

The key point is that you can mix and match

- You can publish messages via one protocol and consume them via another (or indeed simultaneously via several others).
- The 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

Some simple examples…

- Publish via AMQP; consume via STOMP
  - Default exchange
  - Fanout exchange
  - Add in an HTTP consumer
- Publish via HTTP and STOMP; consume via AMQP, STOMP, and HTTP
- Publish via MQTT; consume via STOMP

- And of course other combinations are also possible
  - See http://assortedrambles.blogspot.co.nz/2012/11/the-polyglot-rabbit.html for more information and examples

Publish via AMQP; consume via STOMP

RabbitMQ Tcl client (Tcl commands on top of libRabbitMQ-c)

```
set ch [RMQ::attach amqp://16.156.32.108:5672]
set msg "Hello from Tcl client!"

set rv [RMQ::publish $ch "" "stomp" 0 0 0 $msg]

if {$rv == -1} {
   puts [RMQ::strerror]
   RMQ::abort
}

RMQ::detach $ch
```

Simple example; publishes messages via AMQP to the default exchange with routing key “stomp”, which maps to “stomp” queue created by the consumer…

Uses https://github.com/igb/Erlang-STOMP-Client

```
-module(stomp_consumer_queue).
-export([start/0]).

start() ->
    MyFunction=fun([_, _, {_, X}]) -> io:fwrite("~s~n", [X]) end,
    Conn = stomp:connect("ccin02", 61613, "guest", "guest"),
    stomp:subscribe("/queue/stomp", Conn, []),
    stomp:on_message(MyFunction, Conn).
```

Publish via AMQP; consume via STOMP

As per the previous example, but this time we'll publish messages to the built-in fanout exchange (`amq.fanout`). Messages will be routed to any queue bound to the exchange.

```
set ch [RMQ::attach amqp://16.156.32.108:5672]
set msg "Hello from Tcl client!"

set rv [RMQ::publish $ch "amq.fanout" "" 0 0 0 $msg]

if {$rv == -1} {
   puts [RMQ::strerror]
   RMQ::abort
}

RMQ::detach $ch
```

And let's have a couple of different consumers...

Publish via AMQP; consume via STOMP

Python AMQP consumer (using https://code.google.com/p/py-amqplib on OpenVMS):

```
from amqplib import client_0_8 as amqp

conn = amqp.Connection(host="16.156.32.108:5672 ", userid="guest", password="guest", virtual_host="/", insist=False)
chan = conn.channel()
q = chan.queue_declare("", durable=False, auto_delete=True, exclusive=True)
q = q[0]
chan.queue_bind(queue=q, exchange="amq.fanout", routing_key="anything")

def my_callback(msg):
        print msg.body
        chan.basic_ack(msg.delivery_tag)

chan.basic_consume(queue=q, callback=my_callback)

while True:
        chan.wait()
```

Both consumers will receive copies of any messages published to the `amq.fanout` exchange, regardless of consumer protocol.

Erlang STOMP consumer:

```
-module(stomp_consumer_exchange).
-export([start/0]).

start() ->
    MyFunction=fun([_, _, {_, X}]) -> io:fwrite("~s~n", [X]) end,
    Conn = stomp:connect("ccin02", 61613, "guest", "guest"),
    stomp:subscribe("/exchange/amq.fanout", Conn, []),
    stomp:on_message(MyFunction, Conn).
```

Add in an HTTP consumer

Using the RabbitHub plugin we can add into the mix an HTTP consumer. This can be readily implemented using the Mongoose embedded web server (see https://code.google.com/p/mongoose/), which we've ported to OpenVMS.

Click here to see HTTP consumer code!

And the following cURL command may then be used to register our Mongoose-based webserver callback with RabbitHub:

```
$ curl :== $curl$root:[bin]curl.exe
$ curl -vd -
"hub.mode=subscribe&hub.callback=http://ccin02:4321/test&hub.topic=foo&hub.verify=sync&hub.lease_seconds=86400" -
http://guest:guest@16.156.32.108:15670/subscribe/x/amq.fanout
$ exit
```

Now, in addition to going to our STOMP and AMQP consumers, any messages posted to the `amq.fanout` exchange will also be received via HTTP by our Mongoose-based consumer.

Publish via STOMP and HTTP

Further extending the fanout example, we in addition to consuming messages via multiple protocols, we can also publish via multiple protocols.

Via STOMP:

```
-module(stomp_publish_fanout).
-export([start/0]).

start() ->
    Conn = stomp:connect("ccin02", 61613, "guest", "guest"),
    stomp:send(Conn, "/exchange/amq.fanout", [], "This is a test"),
    stomp:disconnect(Conn).
```

Via HTTP/RabbitHub:

```
$ curl :== $curl$root:[bin]curl.exe
$ curl -vd "Hello via HTTP" http://guest:guest@ccin02:15670/endpoint/x/amq.fanout?hub.topic=anything
$ exit
```

Publish via MQTT; consume via STOMP

Paho-C MQTT publisher code:

Click here to see client code!

Erlang STOMP consumer:

```
-module(stomp_consumer_mqtt).
-export([start/0]).

start() ->
    MyFunction=fun([_, _, {_, X}]) -> io:fwrite("~s~n", [X]) end,
    Conn = stomp:connect("ccin02", 61613, "guest", "guest"),
    stomp:subscribe("/topic/MQTT Example", Conn, []),
    stomp:on_message(MyFunction, Conn).
```

53

AGENDA

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

Summary

- RabbitMQ is a humble and modest polyglot
  - It does not boast about its ability to speak multiple languages and different dialects of the same language
    - But perhaps it should
    - The ability to be able to seamlessly receive and propagate data via multiple protocols affords great flexibility
    - 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 mechanism without having to develop and maintain separate gateway components
  - Polyglot nature/capabilities heavily underpinned by Erlang
- Open Source
  - Large and active communities
  - Client APIs available in numerous languages for all of the messaging protocols mentioned
- Open Standards

... and it all works on OpenVMS ... some work is still required on the Erlang port

AGENDA

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


Thank you