Presentation Outline

● Overview of AMQ7
● Technical Discussion of AMQ 7 Operation
● Cloud-Messaging Demonstration
Overview of AMQ7
AMQ7. GA. Today.
AMQ7 At A Glance

AMQ Broker

AMQ Interconnect

AMQ Clients
AMQ7 Broker

- High performance general-purpose message broker
- Asynchronous core with thread pooling for improved scale and performance
- Support for multiple protocols
  - Legacy “Core” protocol
  - Legacy “Openwire” protocol
  - Standard AMQP
  - Standard MQTT
  - STOMP
AMQ7 Interconnect

- All new message router for AMQP
- Separates message routing from message storage
- Integrates clients and brokers in flexible, scalable networks
- Provides direct/brokerless message delivery
- Provides security
- Leverages the extensive capabilities of the AMQP protocol
AMQ7 Clients

- Standard JMS2
- Reactive AMQP clients for better integration
  - C/C++
  - Python
  - Javascript (browser and Node.js)
  - .NET
- Legacy clients
  - AMQ6 (ActiveMQ5)
  - HornetQ
  - MRG-M
A Word about Performance

Broker Performance

SpecJMS - Transaction rate, Durable message rates, filtering, etc.

Router Performance

Raw latency and throughput
AMQ7 is Next-Generation Messaging for Enterprise, Cloud, and IoT
AMQP Anatomy
Message Producer

Process

Messaging System
Message Producer

```
on_start():
    conn = container.connect(hostname)
```
on_start():
    conn = container.connect(hostname)
    sender = container.create_sender(conn, "Service")
on_start():
    conn = container.connect(hostname)
    sender = container.create_sender(conn, “Service”)

on_sendable(event):
    msg = Message(headers, body)
    sender.send(msg)
on_start():
    conn = container.connect(hostname)
    sender = container.create_sender(conn, "Service")

on_sendable(event):
    msg = Message(headers, body)
    sender.send(msg)

onaccepted(event):
    # message delivery confirmed
Message Consumer

Messaging System

Process
on_start():
    conn = container.connect(hostname)
on_start():
    conn = container.connect(hostname)
    receiver = container.create_receiver(conn, "Service")
on_start():
    conn = container.connect(hostname)
    receiver = container.create_receiver(conn, "Service")

on_message(event):
    Process(event.message)
on_start():
    conn = container.connect(hostname)
    receiver = container.create_receiver(conn, “Service”)

on_message(event):
    Process(event.message)
    container.accept(event.delivery)
AMQP Protocol Features

- Full-Duplex and Asynchronous
- Message encoding: Body and Headers/Annotations
- Settlement and Disposition
  - Settlement: Best Effort; At-Least-Once; Exactly-Once
  - Disposition: Accepted, Rejected, Released
- Flow Control
  - Message Credit
  - Session Frames
- Multiplexing
- Addressing
Brokered Messaging

Producer

Consumer

Consumer

Consumer
Non-Brokered Messaging
Scaling Out
Scaling Out

PHX

ATL

BOS
Scaling Out
Hybrid Cloud Demonstration
The AMQ Network

RDU Data Center

- Enmasse Cluster (AMQ in openshift)
- ON-PREM Router

AWS

- AWS Router

Azure

- AZURE Router

20
Security Configuration

ON-PREM, AWS, and AZURE routers mutually authenticate using a dedicated x.509 Certificate Authority

Connection roles are explicit. Inter-router connections are separate from normal (client access) connections.

Client access to the cloud routers is not exposed outside the cloud provider.
The Application

- Service is an internal Enterprise application service
- All services are hosted inside the enterprise (in openshift)
- Service and SubService.B are also hosted in the public cloud for overflow
- SubService.A uses sensitive data and is not deployed outside the enterprise
The AMQ Network

RDU Data Center

OpenShift Cluster (AMQ in openshift)

ON-PREM Router

client-once

20

AWS Router

SubService.B

AZURE Router

Service

SubService.B

20

client-once
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHatNews

#redhat #rhsummit
LEARN. NETWORK. EXPERIENCE OPEN SOURCE.