ENTERPRISE MESSAGING AND JBOSS A-MQ

Jakub Knetl
jknetl@redhat.com
LECTURE OUTLINE

- Messaging systems
  - JMS specification
  - JMS API
- JBoss A-MQ
  - JBoss A-MQ and Apache ActiveMQ
  - Protocols
  - topologies
- Apache Artemis
ENTERPRISE MESSAGING

- data exchange between applications
- Message oriented middleware (MoM)
  - Message broker servers as mediator between communicating parties
BENEFITS OF MESSAGING SYSTEM

- Asynchronous communication
- Loose coupling
- Scalability
- Reliability
- Message routing and transformation
JMS SPECIFICATION

• goal is to:
  - provide messaging functionality to the java applications
  - maximizes portability between messaging products
  - define common messaging concepts

• JMS is not messaging system!
• JMS provides API for messaging products
• JMS does not address:
  - load balancing
  - fault tolerance
  - administration
  - wire protocol
  - security
BASIC CONCEPTS

- JMS provider
- (non) JMS client
  - producer/consumer
- JMS domains
- JMS destination
- JMS Message
MESSAGE STRUCTURE

- Headers
  - key value pairs
  - two types (differs only semantically):
    - default headers
    - custom properties
## MESSAGE HEADERS LIST

<table>
<thead>
<tr>
<th>Header name</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSDestination</td>
<td>destination on the broker</td>
</tr>
<tr>
<td>JMSDeliveryMode</td>
<td>persistent or nonpersistent delivery mode</td>
</tr>
<tr>
<td>JMSExpiration</td>
<td>message will not be delivered after expiration</td>
</tr>
<tr>
<td>JMSMessageID</td>
<td>Identification of the message</td>
</tr>
<tr>
<td>JMSPriority</td>
<td>Number 0 - 9 (0-4 low, 5-9 high priority). Advise only</td>
</tr>
<tr>
<td>JMSTimestamp</td>
<td>Time when the message is handed to provider for send</td>
</tr>
<tr>
<td>JMSCorrelationID</td>
<td>links message to another one</td>
</tr>
<tr>
<td>JMSReplyTo</td>
<td>Destination for reply</td>
</tr>
<tr>
<td>JMSRedelivered</td>
<td>Contains true if the message was likely redelivered</td>
</tr>
</tbody>
</table>
JMS PROPERTIES

- Custom
  - Used for application specific data
- JMS defined
  - JMSX prefix in the name (e.g. JMSXAppID, JMSXConsumerTXID, ...)
- Provider specific
  - JMS_<vendor_name>
  - typically used in non-JMS clients
MESSAGE SELECTORS

- Message filtering based on properties
- Condition based on subset of SQL92

PRICE <= 1000 AND COLOR = 'RED'

FLIGHT_NUMBER LIKE 'N14%'
MESSAGE CONTENT

- There are several types of message defined in JMS:
  - TextMessage
  - MapMessage
  - BytesMessage
  - StreamMessage
  - ObjectMessage
COMMUNICATION DOMAINS

- Communication type:
  - Point-to-point (PTP)
  - Publish/Subscribe (Pub/Sub)
POINT TO POINT COMMUNICATION

- Destination is a queue
- One of multiple consumers gets message
- Load is distributed across consumers
- Message is stored until some consumers receives it
PUBLIS SUBSCRIBE DOMAIN

- Destination is a topic
- message is delivered to all subscribers
- message is thrown away if there is no subscription
- Durable subscriber
  - if durable subscriber disconnects broker is obliged to store all messages for later delivery
JMS API

- JMS 1.0 (2001)
  - Different APIs for pub/sub and PTP communication
- JMS 1.1 (2002)
  - Classic API - unified API for pub/sub and PTP
- JMS 2.0 (2013)
  - Classic API
    - it is not deprecated and will remain part of JMS indefinitely
  - Simplified API
    - less code needed
    - AutoCloseable resources -> Java 7 needed
    - no checked exceptions
CLASSES OF CLASSIC API

- ConnectionFactory - administered object
- Connection
- Session
- MessageProducer
- MessageConsumer
- Destination - administered object
- Message
DEVELOPMENT MODE

- determines level of delivery reliability
  - Persistent (default)
    - provider should persist the message
    - message must be delivered once and only once even in case of provider failure
  - Nonpersistent
    - provider is instructed not to persist the message
    - message must be delivered at most once
    - message is usually lost on provider failure
    - better performance
MESSAGE ACKNOWLEDGEMENT

- Delivery between broker and client is not considered successful until message is acknowledged.
- acknowledgement modes:
  - DUPS_OK_ACKNOWLEDGE
  - AUTO_ACKNOWLEDGE
  - CLIENT_ACKNOWLEDGE
TRANSACTIONS

- Session can be transacted
- multiple messages handled as atomic unit
- transaction is completed by calling commit() or rollback() on session
- commit also acknowledges message
- Support for distributed transaction is not required by JMS
  - but still many providers implement distributed transactions
  - JMS recommends support using JTA XAResource API
CODE EXAMPLES

- Classic API
  - synchronous send
  - asynchronous send
  - synchronous receive
  - asynchronous receive
- Simplified API
PART II

APACHE ACTIVEMQ
APACHE ACTIVEMQ

- Opensource MoM
- JMS 1.1 compliant
- Supports many protocols and clients
- other features:
  - High availability
  - scalibility
  - management
  - security
JBOSS A-MQ

- Open-source messaging platform
- Messaging system based on Apache ActiveMQ
- Runs on OSGI container
- Enable easy deployment
- Provides web based management console
JBoss A-MQ

**Red Hat JBoss A-MQ**

**Development and tooling**
- JBoss Developer Studio including JBoss Fuse IDE

**Reliable messaging**
- Apache ActiveMQ

**Container**
- Apache Karaf + Fuse Fabric

**Management and monitoring**
- JBoss Operations Network
- JBoss Fabric Management Console
CONFIGURATION

- XML file
- most of things work out of the box

- Default configuration example
MESSAGE STORES

- kahaDB
- multi kahaDB
- levelDB
- JDBC

```xml
<persistenceAdapter>
  <kahaDB directory="${activemq.data}/kahadb"/>
</persistenceAdapter>

<persistenceAdapter>
  <jdbcPersistenceAdapter dataSource="#derby-ds"/>
</persistenceAdapter>

<!-- Embedded Derby DataSource Sample Setup -->
<bean id="derby-ds" class="org.apache.derby.jdbc.EmbeddedDataSource">
  <property name="databaseName" value="derbydb"/>
  <property name="createDatabase" value="create"/>
</bean>
```
CONNECTION TO BROKER

- Transport connectors
  - For client to broker connections
- Network connectors
  - For broker to broker connections
- Many transport protocols supported:
  - tcp, udp, nio, ssl, http/https, vm
WIRE PROTOCOLS

- Openwire
- STOMP
- AMQP
- MQTT
OPENWIRE

- Binary format developed for ActiveMQ purposes
- default wire format
- very efficient
- complex implementation
- Native clients for java, c/c++, c#
- advanced features:
  - flow control
  - client load balancing
STOMP

- Streaming text oriented message protocol
- very simple
- easy to implement
- worse performance
MQTT

- Client server publish/subscribe messaging protocol
- ultra lightweight
- easy to implement
- supports only topics (no PTP messaging)
AMQP

- binary protocol
- open standard
- support both ptp and pub/sub
- advanced features:
  - flow control
TRANSPORT CONFIGURATION

- in transport connection section
- Connection type and options are defined by URI

<transportConnectors>
  <transportConnector name="mqtt"
    uri="mqtt://localhost:1883?wireFormat.maxFrameSize=100000"/>
  <transportConnector name="openwire"
    uri="tcp://0.0.0.0:61616?maximumConnections=1000&wireFormat.maxFrameSize=104857600"
    discoveryUri="multicast://default"/>
</transportConnectors>
HIGH LEVEL PROTOCOL URIS

- typically uses composite URI
- failover
- static

failover:(tcp://primary:61616,tcp://secondary:61616)?randomize=false
HIGH AVAILABILITY (HA)

- Messaging systems usually processes business critical data
- broker must be accessible 24/7
- ActiveMQ provides various mechanisms to ensure HA
HA IN ACTIVEMQ

- Group of brokers forms logically one broker
- Master broker
  - communicates with clients
- Slave brokers
  - Passive (all connectors are stopped)
- election mechanisms
- client reconnects in case of failure (failover)
- Message acknowledgment after the message is stored safely
MASTER SLAVE FOR HA

- Shared JDBC master/slave
- Shared file system master/slave
- Replicated levelDB master/slave
**SHARED JDBC**

- Shared database as persistence storage required
- Election mechanism: Locking tables in database
- Acknowledgment: After message is safely stored in database
- Single point of failure
**SHARED FILESYSTEM**

- Usually faster than JDBC
- Need file system with reliable locking mechanisms
- Persistent storage is located on shared file system
- Election mechanism: Locking file
- Acknowledgment: after message is stored on shared filesystem
- Single point of failure
REPLICATED LEVELDB

- Shared nothing
- No single point of failure
- Servers coordinates themselves by exchanging messages (replication protocol)
- Needs zookeeper server(s)
  - For master election only
- Replication protocol
  - Slave brokers connect to the master
  - Messages are replicated to slaves
  - Message is acknowledged after replicating at least to quorum of brokers.
NETWORK OF BROKERS

- connections between broker
- message forwarding
- enables massive scalability
- requires careful configuration
NETWORK CONNECTOR

Broker A:

```xml
<networkConnectors>
  <networkConnector uri="static:(tcp://B:61617)"/>
</networkConnectors>
```
NETWORK OF BROKERS

- duplex connections
- destination filtering
- dynamic vs static forwarding
- AdvisoryMessages
- network consumer priority
- networkTTL
HIERARCHIES OF NETWORKS

- concentrator topology
- hub and spokes topology
- tree topology
- mesh topology
- complete graph
OTHER ACTIVEMQ FEATURES

- exclusive consumers
- message groups
- wildcards (. * >)
- virtual topic
- DLQ
**APACHE ACTIVEMQ ARTEMIS**

- new Apache MoM
- non-blocking architecture => great performance
- merges codebase with JBoss HornetQ
- JMS 2.0 compliant
- Support for:
  - ActiveMQ clients
  - AMQP
  - STOMP
  - HornetQ clients