Building a Reliable Messaging Infrastructure with Apache ActiveMQ

Bruce Snyder
IONA Technologies
Do You JMS?
A Crash Course in Messaging

Loosely coupled vs. tightly coupled
- The loosely coupled exchange of messages
- Producers and consumers are not aware of one another
- Communication is indirect via destinations
- Synchronous or asynchronous message delivery
- Optional features
  -- Durability
  -- Persistence
  -- Transactionality
- Understanding message domains is important
Analogy: person-to-person email
- One-to-one
- One consumer per message
- Based on queues
- Once and only once delivery
- Queues retain messages until consumed or expired
- Good for load balancing messages
- Queue browsing
- Partial outages OK
Analogy: Mailing lists
- Based on topics
- Publishers and subscribers
- Publisher had no knowledge of subscribers
- Messages are delivered to all subscribers
- Business events, i.e., EDA
First Principle of Messaging

There is no absolute comparison to distinguish Message Oriented Middleware

- All situations are unique in some way
- Too many use cases
What Are Your Performance Objectives?

- Will you use synchronous or asynchronous messaging?
- Is performance more critical than QOS?
- Can your app handle duplicate messages?
- Can your app handle missing messages?
- Do you need messages to be received in order?
- Do you have any slow consumer situations?
- Do you know your message requirements?
- What is the average message size?
- How consistent is this size?
- How much can it vary?
- Are there attachments on the messages?
- Are the messages binary or text?
- Will your messages need compression?

-- Do all messages need compression?
What Trade-Offs Can You Accept?

- Trade-offs are a given
-- Speed and reliability are mutually exclusive
-- Each comes at a price
Common Trade-Offs

Synchronous vs. Asynchronous Messaging

- Synchronous vs. asynchronous
- Durability and persistence
- Transactionality
- Message consumption
Common Trade-Offs

Durability vs. Persistence

- Durability
  -- When subscriber goes offline messages are held
- Persistence
  -- Messages held in a persistent data store
Common Trade-Offs

Transactions

-- Using them in batches can actually speed up some operations
Second Principle of Messaging

Messaging is focused on reliability more than performance

- Bottlenecks are in the speed bumps, not message destinations
Messaging Use Cases Are Still Expanding...

- Integration
- Decoupling
  --- Level of indirection using messaging as the mediator
- Enterprise Service Buses
- High amount of message variability
- Intelligent routing
- Itinerary-based routing
- Content-based routing
- Orchestration (e.g., BPEL)
- Workflow/BPM and BAM
- Complex Event Processing
What Is Apache ActiveMQ?

- JMS 1.1 compliant
- Integration with:
  -- Geronimo, Spring, Tomcat, JBoss and any J2EE 1.4 container (e.g., WebLogic or WebSphere)
- Supported transports
  -- TCP, UDP, multicast, SSL, HTTP, Jabber (XMPP), JXTA, etc.
- Pluggable persistence and security
- Wildcards, selectors, composite destinations
- Fast and highly scalable
- Topologies supported:
  -- Clustering, peer-to-peer, federated network support
- Multi-language clients:
  -- Java, C/C++, .NET, Ruby, Perl, PHP, Python
Examples Demo

Easily send and receive messages using the default examples
Configuration

<xml />

(conf/activemq.xml)
ActiveMQ Uses URIs

<protocol>://<host>:<port>?<transport-options>
Example URIs

vm://localhost?broker.persistent=false

tcp://localhost:61616?jms.useAsyncSend=true

stomp://localhost:61613

failover:(tcp://host1:61616,tcp://host2:61616)?initialReconnectDelay=100
Two Types of Transports

- Client to broker communication
  -- The `<transportConnector>` element
- Broker to broker communication
  -- The `<networkConnector>` element
- High performance journal
- JDBC provider
- Kaha provider
STOMP Client Demo

Demonstrate using ActiveMQ with the Ruby STOMP client library
Federated network of brokers
- Many brokers acting as a single, logical broker
- Use network connectors between each other
- Store and forward strategy
- Brokers use static or discovery based routing
- Clients can also use static or discovery based routing
Topology Example
Topology Example
Topology Example
Topology Example
- Pure Master/Slave
  -- Shared nothing, fully replicated topology
  --- Does not depend on shared filesystem or database
  -- Slave broker consumes all message states from the Master broker (messages, acks, tx states)
  -- Slave does not start any networking or transport connectors
  -- Master broker will only respond to client when a message exchange has been successfully passed to the slave broker
- Shared Filesystem
  -- Uses directory on shared filesystem (SAN)
- JDBC Master/Slave
  -- Uses a shared database
Message Prefetch

```
queue = new ActiveMQQueue("TEST.QUEUE?
    consumer.prefetchSize=10");
consumer = session.createConsumer(queue);
```
Asynch Dispatch

- Asynchronous message delivery
- Configurable on ConnectionFactory, Connection and Consumer
- Mostly used for slow consumers
Wildcards on Subscriptions

Price.>
Price.Stock.>
Price.Stock.NASDAQ.*
Price.Stock.*.IBM

. separates names in a path
* matches any name in the path
> matches recursively downward in the path
- Logical destinations that map onto one or more physical destinations
- Works around the issue of JMS durable subscribers
  -- Only one thread can be active per clientID and subscriber name
  -- This works around that by using queue semantics on topics
Exclusive Consumers

- Anytime more than one consumer consuming from a queue, message order is lost
- This feature allows a single consumer to consume all messages on a queue to maintain message ordering
- Uses the JMSXGroupID property to define which message group a message belongs
- Provides:
  -- Guarantees ordered processing of related messages across a single queue
  -- Load balancing of message processing across multiple consumers
  -- HA/failover if consumer goes down
Total Ordering

- Ensures that each consumer will see the same total order of messages on a topic
-- This feature is *per broker*
queue = new ActiveMQQueue("TEST.QUEUE?
    consumer.priority=10");
consumer = session.createConsumer(queue);

- Just like it sounds
--- Gives consumer priority for message delivery
- Allows for the weighting of consumers to optimize network traversal for message delivery
Retroactive Consumer

```
queue = new ActiveMQQueue("TEST.QUEUE?
    consumer.retroactive=true");
consumer = session.createConsumer(queue);
```

- Normal JMS Consumer with a kicker
- Message replay at start of a subscription
  -- At the start of every subscription, send any old messages that the consumer may have missed
- Configurable via timed or fixed size recovery

Mirrored Queues
- jconsole
- Hyperic
- IONA Fuse HQ
- ActiveMQ web console
What is Apache Camel?

- Lightweight implementation of EIP
What is EIP?

- The bible of integration patterns
Example Pattern: Content Based Router

RouteBuilder builder = new RouteBuilder() {
    public void configure() {
        from("seda:a").choice().when(header("foo").isEqualTo("bar")).to("seda:b").when(header("foo").isEqualTo("cheese")).to("seda:c").otherwise().to("seda:d");
    }
};

- The Java version of the Camel CBR
Example Pattern:
Content Based Router

```
<camelContext id="buildSimpleRouteWithChoice"
xmlns="http://activemq.apache.org/camel/schema/spring">
  <route>
    <from uri="seda:a"/>
    <choice>
      <when>
        <predicate>
          <header name="foo"/>
          <is EqualTo value="bar"/>
        </predicate>
        <to uri="seda:b"/>
      </when>
      <when>
        <predicate>
          <header name="foo"/>
          <is EqualTo value="cheese"/>
        </predicate>
        <to uri="seda:c"/>
      </when>
      <otherwise>
        <to uri="seda:d"/>
      </otherwise>
    </choice>
  </route>
</camelContext>
```

- The XML version of the CBR
Thank You for Attending

Please fill out your surveys