Riviera JUG 2011/03/11
Soirée Messaging
HornetQ & The Web
How HornetQ Embraces the Web and Its Different Forms

Jeff Mesnil

Riviera JUG 2011/03/11
Who?

• Jeff Mesnil
• Software Developer
• former Red Hatter
• work for Bestofmedia Group
• jmesnil@gmail.com
• http://jmesnil.net/
• twitter: @jmesnil
What?

• Messaging/JMS Introduction
• What is HornetQ?
• HornetQ & The Web
• HornetQ & REST
• HornetQ & HTML5 WebSocket
• Pros & Cons
• Questions?
MOM
Message-Oriented Middleware

- software or hardware infrastructure to send and receive messages between distributed systems
- loosely-coupling
- reliability
- asynchronocity
JMS
Java Message Service

- `javax.jms`
- Java **API**
- Part of Java EE
- JMS 1.1 (March 2002)
- 2 messaging models
  - Point-to-point with *Queues*
  - Publish/Subscribe with *Topics*
JMS Resources

- **Producer** (sender/publisher)
  - send messages
- **Consumer** (receiver/subscriber)
  - receive messages
- **Session**
  - context for producing and consuming messages
- **Destination** (queue/topic)
  - «staging area» used to exchange messages
JMS Message

- Data exchanged by producer and consumer
- Standard Headers
  - JMSDestination
- Application-Specific Properties
  - SKU
- Opaque Body
  - byte[], String, Map,...
JMS Models

- Point-to-point (queues)
  - 1 producer - 1 consumer
- Publish/Subscribe (topics)
  - 1 producer - N consumers
JMS Features

- Message types
  - Stream, Map, Text, Object, Bytes
- Filters
  - headers, properties
- Acknowledgement
  - auto, client, dups_ok
- Transacted
  - XA
JMS Consumer

• pull:
  Message m = consumer.receive(timeout);

• push:
  session.setMessageListener(new MessageListener() {
      public void onMessage(Message m) {
          //...
      }
  });
• [http://jboss.org/hornetq/](http://jboss.org/hornetq/)
• JBoss community project
• Open Source (Apache License 2.0)
• Multi-protocol asynchronous messaging system
• Designed for *performance*
• Designed with *usability* in mind
Usability

• Ease of use (configuration, deployment)
• Simple API
• Documentation
• 90+ examples
  • A feature without documentation and example is useless
• Minimal third-party dependencies
Deployment

• Fully functional standalone messaging server
• Default messaging provider for JBoss AS 6
• Integrated with any Java EE application server using its JCA adaptor
• Use dependency-injection framework to embed it in your application
Generic Messaging Architecture

• Simple set of POJOs (JMS-agnostic)
• No hard dependencies on JMS, JNDI, JCA
• JMS is 1 way to use HornetQ
• 2 ways to use HornetQ on the Web
• Different messaging APIs / protocols in a single consistent messaging system
Performance

- Very high performance journal
  - No relational database
  - Java NIO
  - JNI interface to Linux asynchronous IO
- Very fast transport layer
  - Netty
- Don’t take my word for it!
Supported Protocols

- Core
- JMS
- STOMP
- REST
HornetQ Core Protocol

- Java
- Optimized
- Send messages to an *address*
- Consume messages from a *queue*
- *Bindings* between addresses and queues and queues and consumers define the messaging model
HornetQ Core API

Producer sends message to Address

? Queue

message

Consumer
JMS API

• Point-to-Point (queues)
  • 1 producer - 1 consumer
• Publish/Subscribe (topics)
  • 1 producer - N consumers
JMS Queue

Producer \(\rightarrow\) Address

Queue \(\rightarrow\) Consumer 1

Queue \(\rightarrow\) Consumer 2

Riviera JUG 2011/03/11
JMS Models

- JMS Queue
  - 1 address - 1 queue binding
  - Many consumers for the queue
- JMS Topic
  - 1 address - many queues binding
  - 1 single consumer per queue
STOMP

- [http://stomp.codehaus.org/](http://stomp.codehaus.org/)
- Text-orientated messaging protocol
- Interoperability
- Clients for many languages
- Simple and flexible (headers, ack, transactions)
STOMP

$ telnet localhost 61613
CONNECT
login: jmesnil
passcode: mypassword
^@

SEND
destination:/queue/myqueue

this is my message body
^@
REST API

• HTTP-Based RESTful API
• More on that in a minute...
Many Other Features

- Clustering, HA, Failover
- Bridges and Diverts
- Management (JMX, JMS)
- Dead Letters, Expiration
- ...
- [http://community.jboss.org/wiki/HornetQFeatures](http://community.jboss.org/wiki/HornetQFeatures)
But

HornetQ & The Web
HornetQ & The Web

- Web is everywhere
- Cloud Computing
- Integration
  - Enterprise environment
  - Heterogenous platforms
- End-user applications
  - Web Browsers
  - RIA
- More and more web-based applications needs messaging
Messaging and the Web

- Asynchronous
- Loosely-coupled
- Distributed
HornetQ and the Web

- HornetQ Core/JMS over HTTP
- REST API over HTTP
- STOMP over HTML5 WebSocket
HornetQ Over HTTP

- Use HTTP instead of TCP as the transport layer
- Work around firewalls
- Not Web-friendly
  - Requires HornetQ Java client library
  - HornetQ Core wireformat encoded
HornetQ REST API

• RESTful API to do messaging over HTTP
• http://en.wikipedia.org/wiki/Representational_State_Transfer
• Based on RESTEasy
RESTEasy

- [http://www.jboss.org/resteasy/](http://www.jboss.org/resteasy/)
- JBoss community project
- Open Source (Apache license 2.0)
- Fully certified JAX-RS implementation
- Portable to any application server/web container
- Rich set of providers (XML, JSON, YAML, Atom, etc.)
- JAXB marshalling into XML, JSON, etc.
- Wrappers for maps, arrays, lists, and sets of JAXB Objects.
- GZIP content-encoding
- Rich interceptor model
- EJB, Seam, Guice, Spring, and Spring MVC integration
Demo

- Application to track the location of users
  - Users will use their mobile phones to broadcast their position (REST API)
  - Web application to monitor the users’ positions (STOMP over HTML5 WebSocket)
  - Application to keep a log of everybody’s moves (JMS)
JMS Topic

JMS Monitoring Application
HornetQ Server Configuration

hornetq-jms.xml

<configuration>
    <topic name="trackers">
        <entry name="/topic/trackers" />
    </topic>
</configuration>

hornetq-configuration.xml

<acceptor name="jms">
    <factory-class>org.hornetq.core.remoting.impl.netty.NettyAcceptorFactory</factory-class>
</acceptor>
<acceptor name="websockets">
    <factory-class>org.hornetq.core.remoting.impl.netty.NettyAcceptorFactory</factory-class>
    <param key="protocol" value="stomp_ws" />
    <param key="port" value="61614" />
</acceptor>
RESTEasy Configuration

WEB-INF/web.xml

```xml
<listener>
  <listener-class>org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap</listener-class>
</listener>

<listener>
  <listener-class>org.hornetq.rest.integration.HornetqBootstrapListener</listener-class>
</listener>

<listener>
  <listener-class>org.hornetq.rest.integration.RestMessagingBootstrapListener</listener-class>
</listener>

...

<context-param>
  <param-name>rest.messaging.config.file</param-name>
  <param-value>rest-config.xml</param-value>
</context-param>
```
RESTEasy Configuration

rest-config.xml

<rest-messaging>
  <server-in-vm-id>0</server-in-vm-id>
  <use-link-headers>false</use-link-headers>
  <default-durable-send>false</default-durable-send>
  <dups-ok>true</dups-ok>
  <topic-push-store-dir>topic-push-store</topic-push-store-dir>
  <queue-push-store-dir>queue-push-store</queue-push-store-dir>
  <producer-session-pool-size>10</producer-session-pool-size>
  <session-timeout-task-interval>1</session-timeout-task-interval>
  <consumer-session-timeout-seconds>300</consumer-session-timeout-seconds>
  <consumer-window-size>-1</consumer-window-size>
</rest-messaging>
Demo
HornetQ REST API

- RESTful messaging specification
  - [http://www.jboss.org/reststar/specifications/messaging.html](http://www.jboss.org/reststar/specifications/messaging.html)
- Simple REST/HTTP interface to access HornetQ
- Usable by any programming language that has an HTTP client library
- Zero client footprint (no libraries to download, install and deploy)
- No envelope (à la SOAP) or feed (à la Atom) format requirements
  - the HTTP request/response is the message
- Lightweight interoperability
- Pub/Sub (topics) and Point-to-point (queues)
HornetQ REST API

- Follow the headers
- Duplicate Detection
- Persistent messages ?durable=true
Send a Message [1/2]

=> Query the resource URL

HEAD /topics/jms.topic.trackers HTTP/1.1
Host: example.com

--- Response ---
HTTP/1.1 200 Ok
msg-create: http://example.com/topics/jms.topic.trackers/create
msg-create-with-id: http://example.com/topics/jms.topic.trackers/create/{id}
Send a Message [2/2]

POST /topics/jms.topic.trackers/create
Host: example.com
Content-Type: application/json

{}
  "alias": "jmesnil",
  "position": {
    "coords": {
      "longitude": 65.735344466666666,
      "latitude": 35.198200083333326
    },
    "timestamp": 1285943662645
  }
}

--- Response ---
HTTP/1.1 201 Created
msg-create-next: http://example.com/topics/jms.topic.trackers/create/002
Receive a Message - Pull [1/4]

=> query the resource URL

HEAD /topics/jms.topic.trackers HTTP/1.1
Host: example.com

--- Response ---
HTTP/1.1 200 Ok
msg-create: http://example.com/topics/jms.topic.trackers/create
msg-pull-subscriptions: http://example.com/topics/jms.topic.trackers/pull-subscriptions
msg-push-subscriptions: http://example.com/topics/jms.topic.trackers/push-subscriptions
Receive a Message - Pull [2/4]

=> create a subscription by doing an empty POST on the msg-pull-subscriptions URL

POST /topics/jms.topic.trackers/pull-subscriptions HTTP/1.1
Host: example.com

--- Response ---
HTTP/1.1 201 Created
Location: http://example.com/topics/jms.topic.trackers/pull-subscriptions/auto-ack/222
msg-consume-next: http://example.com/topics/jms.topic.trackers/pull-subscriptions/auto-ack/222/consume-next-1
Receive a Message - Pull [3/4]

=> consume the message by POSTing on the subscription

POST /topics/jms.topic.trackers/pull-subscriptions/auto-ack/222/consume-next-1
Host: example.com

--- Response ---
HTTP/1.1 200 Ok
Content-Type: application/json

```json
{
    "alias": "jmesnil",
    "position": {
        "coords": {
            "longitude": 65.735344446666666,
            "latitude": 35.198200083333326,
            ...
        },
        "timestamp": 1285943662645
    }
}
```
Receive a Message - Pull [4/4]

=> if there is no message to consume, will return 503

POST /queues/jms.queue.bar/pull-consumers/consume-next-2
Host: example.com

--- Response ---
Http/1.1 503 Service Unavailable
Retry-After: 5
Receive a Message - Push[1/2]

=> query the Topic resource URL

HEAD /topics/jms.topic.trackers HTTP/1.1
Host: example.com

--- Response ---
HTTP/1.1 200 Ok
msg-create: http://example.com/topics/jms.topic.trackers/create
msg-pull-subscriptions: http://example.com/topics/jms.topic.trackers/pull-subscriptions
msg-push-subscriptions: http://example.com/topics/jms.topic.trackers/push-subscriptions
Receive a Message - Push[2/2]

=> create a push subscription

POST /topics/jms.topic.trackers/push-subscriptions
Host: example.com
Content-Type: application/xml

<push-topic-registration>
  <link rel="template" href="http://somewhere.com/resources/{id}/messages" method="POST"/>
</push-topic-registration>

--- Response ---
HTTP/1.1 201 Created
Location: http://example.com/topics/jms.topic.trackers/push-subscriptions/1-333-1212

=> new messages will be POSTed to http://somewhere.com/resources/{id}/messages URL
Create a Destination

POST /topics
Host: example.com
Content-Type: application/hornetq.jms.topic+xml

<topic name="testTopic" />

--- Response ---
HTTP/1.1 201 Created
Location: http://example.com/topics/jms.topic.testTopic
HTML5

- http://www.w3.org/TR/html5/
- Revision of HTML markup language
- JavaScript APIs
  - canvas, geolocation, video, audio, etc.
  - WebSocket!
HTML5 WebSocket

- «TCP for the Web»
- bidirectional communication between the browser and the web server
- same-origin security model
- JavaScript API in the browser
  - http://dev.w3.org/html5/websockets/
- WebSocket protocol on the server
STOMP Over WebSocket

• WebSocket instead of TCP
• STOMP as the messaging protocol
• JavaScript libraries to send and receive STOMP messages
  • http://jmesnil.net/stomp-websocket/doc/
  • http://github.com/krukow/stomple
• ActiveMQ, RabbitMQ(?)
• Talks about adding it to STOMP 2.0 specification
• Work with text only (no bytes)
• Receive message by push-only
Demo
STOMP Over WebSocket

- JavaScript-based
- Looks like other STOMP clients
- Send and receives text-only messages
Create a Client

// create a client
var url = "ws://example.com:61614/stomp";
var client = Stomp.client(url);

// connect to the STOMP server
client.connect(login, passcode, function() {
  // called back after the client is connected
  // and authenticated to the Stomp server
});

...

client.disconnect();
Send and Receive Messages

// send a message
var tracker = ...; // retrieved from Geolocation API
client.send("jms.topic.trackers", {}, JSON.stringify(tracker));

// subscribe to a destination
var sub_id = client.subscribe("jms.topic.trackers", function(message) {
    // called back every time a message is sent to the destination
    var tracker = JSON.parse(message.body);
    ...
});

// unsubscribe from a destination
client.unsubscribe(sub_id);
REST or WebSocket?

• Different usage and timeline
• WebSocket
  • (+) TCP-like
  • (+) push data from the server to the browser
  • (-) limited support now, will likely expand
  • (-) disabled in browsers, security issues!
• REST
  • (+) works everywhere now
  • (-) no push messages in the browser
Demo Source

• http://github.com/jmesnil/hornetq/
Conclusion

- Different ways to do messaging on the Web
- Integrate with the whole Java / Java EE ecosystem
- Embraces REST principles
- Integration & interoperability
- ... Performance(?)
Questions?
Merci!