Messaging for web and mobile with Apache ActiveMQ

By Bosanac Dejan
Bosanac Dejan?

- Senior Software Engineer at RedHat
- Apache ActiveMQ committer and PMC member
- Co-author of ActiveMQ in Action
- Blog – [http://sensatic.net](http://sensatic.net)
- Twitter – [http://twitter.com/dejanb](http://twitter.com/dejanb)
Agenda

• Challenges of web messaging
• REST vs Stomp
• Mobile messaging using MQTT
• In-browser messaging (Ajax vs Web Sockets)
• Striking the balance
Messaging For Web
Messaging for Web

• Connect from any web application backend (Ruby, PHP, Python, …)
• Connect directly from the browser (AJAX, Web Sockets)
• The main requirement is *simplicity*
What’s wrong with HTTP?

• Nothing at all!
• Ideal for simple request-reply communication
• Lacks semantics for publish-subscribe and point-to-point communication
Limitations

• Pull based protocol
• Easy simple producing
• There’s no concept of consumer or subscription
• There’s no concept of transactions
Pull Consuming

- HTTP techniques
  - Long polling
  - Comet
- Maintain a state
  - Session
  - ClientID
Push Consuming

- Web Hooks – http://webhooks.org
- Provide a callback (HTTP URL) to be called on event
- Trigger callback on every message
Camel HTTP component

```xml
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="activemq:topic:events"/>
    <to uri="http://mysite.com/events"/>
  </route>
</camelContext>
```

- **HawtIO** – [http://hawt.io](http://hawt.io)
- **Missing API for dynamically managing subscribers**
Stomp – what it is?

- [http://stomp.github.com](http://stomp.github.com)
- **Simple Text Orientated Messaging Protocol**
- HTTP for the messaging realm
Stomp – basics

- Very simple, so it’s easy to write clients and servers in practically any language
- A lot of client APIs in C, Java, Ruby, Python, JS, PHP
- Implemented by ActiveMQ, Apollo, HornetQ, RabbitMQ
Stomp - Protocol

- Text based headers, similar to HTTP
- Can transport binary bodies
- Frame command for every messaging concept, like CONNECT, MESSAGE, SUBSCRIBE, ACK, etc.

```
MESSAGE
subscription:0
message-id:007
destination:/queue/a
ccontent-type:text/plain

hello queue a^@
```
Stomp + ActiveMQ

- Available transports

```xml
<transportConnectors>
  <transportConnector name="stomp" uri="stomp://0.0.0.0:61613"/>
  <transportConnector name="stomp+nio" uri="stomp+nio://0.0.0.0:61614"/>
  <transportConnector name="stomp+ssl" uri="stomp+ssl://0.0.0.0:61615"/>
  <transportConnector name="stomp+nio+ssl"
      uri="stomp+nio+ssl://0.0.0.0:61615"/>
</transportConnectors>
```

- NIO implementation for better scalability
- SSL for secure communication
Stomp Java Client

- StompJMS - [https://github.com/fusesource/stompjms](https://github.com/fusesource/stompjms)
- APIs:
  - JMS
  - Blocking
  - Future
  - Callback
Stomp stomp = new Stomp("localhost", 61613);
Future<FutureConnection> future = stomp.connectFuture();
FutureConnection connection = future.await();

CONNECT
host:localhost
accept-version:1.1

CONNECTED
heart-beat:0,0
session:ID:vidra.local-56933-1369046267671-2:1
server:ActiveMQ/5.9-SNAPSHOT
version:1.1
StompFrame frame = new StompFrame(SEND);
frame.addHeader(DESTINATION, StompFrame.encodeHeader("/queue/test"));
frame.addHeader(MESSAGE_ID, StompFrame.encodeHeader("test"));
frame.content(new Buffer("Important Message").getBytes("UTF-8"));
Future<Void> sendFuture = connection.send(frame);

sendFuture.await();

SEND
message-id:test
destination:/queue/test
content-length:17

Important Message
StompFrame disconnect = new StompFrame(DISCONNECT);
Future<Void> disconnectFuture = connection.send(disconnect);
disconnectFuture.await();

DISCONNECT
Stomp stomp = new Stomp("localhost", 61613);
Future<FutureConnection> future = stomp.connectFuture();
FutureConnection connection = future.await();

CONNECT
host:localhost
accept-version:1.1

CONNECTED
heart-beat:0,0
session:ID:vidra.local-56933-1369046267671-2:1
server:ActiveMQ/5.9-SNAPSHOT
version:1.1
Future<StompFrame> receiveFuture = connection.receive();

StompFrame frame = new StompFrame(SUBSCRIBE);
frame.addHeader(DESTINATION, StompFrame.encodeHeader("/queue/test"));

AsciiBuffer id = connection.nextId();
frame.addHeader(ID, id);
Future<StompFrame> response = connection.request(frame);
response.await();

SUBSCRIBE
receipt:2
destination:/queue/test
id:1

RECEIPT
receipt-id:2
StompFrame received = receiveFuture.await();
System.out.println(received.content());

MESSAGE
message-id:ID:vidra.local-56933-1369046267671-2:1:-1:1:1
destination:/queue/test
timestamp:1369046474700
expires:0
subscription:1
content-length:17
priority:4

Important Message
StompFrame unsubscribe = new StompFrame(UNSUBSCRIBE);
unsubscribe.addHeader(ID, id);
Future<Void> unsubscribeFuture = connection.send(unsubscribe);
unsubscribeFuture.await();

UNSUBSCRIBE
id:1
StompFrame disconnect = new StompFrame(DISCONNECT);
Future<Void> disconnectFuture = connection.send(disconnect);
disconnectFuture.await();

DISCONNECT
Advanced Stomp

- Ack modes
- Transactions
- Reliable messaging
- Protocol Negotiations
- Heart-beating
Stomp and ActiveMQ

• Queues and Topics
• Reliable Messaging
• Temporary destinations
• Durable topic subscribers
• Destination wildcards
• Message selectors
Stomp and ActiveMQ

- Message expiration
- Composite destinations
- Priority consumers
- Exclusive consumers
Messaging For Mobile
MQTT
Messaging for Mobile

- Different set of requirements
- Low bandwidth network
- Small footprint
- Low power usage
MQTT

• IoT (Internet of Things) protocol
• Efficient binary protocol
• Developed by IBM for embedded devices telemetry
MQTT Features

• Low bandwidth
  • Smallest frame 2 bytes
• Unreliable networks
• Small footprint
MQTT for mobile

- Ideal for native mobile applications
- Use case: Facebook messenger
  - Phone-to-phone delivery in milliseconds, rather than seconds
  - Without killing battery life
MQTT

- Publish/subscribe protocol – topics only
- 3 QoS Options:
  - At Most Once – message loss might occur
  - At Least Once – duplicates might occur
  - Exactly Once – guaranteed delivery
MQTT + ActiveMQ

- Available transports

```
<transportConnectors>
  <transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883"/>
  <transportConnector name="mqtt+nio" uri="mqtt+nio://0.0.0.0:1884"/>
  <transportConnector name="mqtt+ssl" uri="mqtt+ssl://0.0.0.0:1885"/>
  <transportConnector name="mqtt+nio+ssl"
    uri="mqtt+nio+ssl://0.0.0.0:1886"/>
</transportConnectors>
```

- NIO implementation for better scalability
- SSL for secure communication
MQTT client

- mqtt-client
  - https://github.com/fusesource/mqtt-client
- APIs:
  - Blocking
  - Callback
  - Future
MQTT Example

```java
MQTT mqtt = new MQTT();
mqtt.setHost("localhost", 1883);
final CallbackConnection connection = mqtt.callbackConnection();
```
MQTT Example

```java
connection.connect(new Callback<Void>() {
    public void onSuccess(Void value) {
        connection.publish("test", "Important Message!".getBytes(), QoS.AT_LEAST_ONCE, false, null);
    }

    public void onFailure(Throwable value) {
        connection.disconnect(null);
    }
});
```
final Promise<Buffer> result = new Promise<Buffer>();

connection.listener(new Listener() {
    public void onConnected() {}
    public void onDisconnected() {}
    public void onPublish(UTF8Buffer topic, Buffer body, Runnable ack) {
        result.onSuccess(body);
        ack.run();
    }
    public void onFailure(Throwable value) {
        result.onFailure(value);
        connection.disconnect(null);
    }
});

LOG.info("Received: " + result.await(5, TimeUnit.MINUTES));
MQTT Example

```java
connection.connect(new Callback<Void>() {
    public void onSuccess(Void aVoid) {
        Topic[] topics = {
            new Topic(utf8("test"), QoS.AT_LEAST_ONCE)
        };
        connection.subscribe(topics, null);
    }
    public void onFailure(Throwable value) {
        connection.disconnect(null);
    }
});
```
MQTT Android Example

https://github.com/jsherman1/android-mqtt-demo/
In-Browser Messaging
In-browser Messaging

• Use JavaScript to produce and consume messages directly from the browser
• We need to leverage existing web technologies like Ajax and Web Sockets
• We need a web server that’s able to communicate with the broker
Ajax

• Old-school way
• Comes bundled with ActiveMQ distribution
Ajax – explained

• Requires additional servlet as an intermediary between broker and clients
• POST to send messages
• Jetty continuations to receive messages
<script type="text/javascript" src="js/jquery-1.4.2.min.js"></script>
<script type="text/javascript" src="js/amq_jquery_adapter.js"></script>
<script type="text/javascript" src="js/amq.js"></script>

var amq = org.activemq.Amq;
amq.init(
    uri: 'amq',
    logging: true,
    timeout: 20
);
</script>
Ajax – Example

```javascript
amq.sendMessage("queue://TEST", "Important Message!");

var myHandler =
{
    rcvMessage: function(message)
    {
        console.log("Received message: "+ message);
    } 
};

amq.addListener("myListener", "queue://TEST", myHandler.rcvMessage);
```
WebSocket

- Evolution over Ajax and Comet
- Defines a “socket” – permanent duplex connection – between browser and server
- Server and browser can exchange messages
WebSocket

• Fully standardized and part of HTML5 spec
  • Protocol – standardized by IETF
  • API – standardized by W3C
• Supported by most modern web servers and browsers
var connection = new WebSocket("ws://localhost:8161");

connection.onopen = function() {
    console.log("Connection opened!");
};

connection.onmessage = function(msg) {
    console.log("Received Message: " + msg.data);
};

connection.onerror = function(error) {
    console.log("Error occurred: " + error);
}

connection.onclose = function(evt) {
    console.log("Connection closed!");
};

connection.send("Important Message!");
connection.close();
WebSocket + ActiveMQ

- WebSocket is a plain socket – like a raw TCP
- We need a protocol on top of it to use all concepts of messaging and connect to broker
- WebSocket+Stomp ideal for standard web clients!
- WebSocket+MQTT ideal for mobile web clients!
WebSocket + ActiveMQ

- New **ws** and **wss** transports

```xml
<transportConnectors>
  <transportConnector name="websocket" uri="ws://0.0.0.0:61613"/>
  <transportConnector name="secure_websocket" uri="wss://0.0.0.0:61614"/>
</transportConnectors>
```

- **wss** transport needs SSL context configuration
WebSocket + ActiveMQ

• Stomp supported since 5.4.0
• MQTT supported since 5.9.0
• You can use both over the same connector
• Connector detects the protocol when connection is initialized
**stomp-websocket**

- Client side library stomp-websocket
  - [http://github.com/jmesnil/stomp-websocket](http://github.com/jmesnil/stomp-websocket)
- Supports Stomp 1.1
- Not a “pure” Stomp as it requires WebSocket handshake
stomp-websocket Example

```javascript
var client = Stomp.client("ws://localhost:61614");
var connected = false;
client.connect("admin", "admin", function() {
    connected = true;
    client.subscribe("/queue/test", function(message) {
        console.log("Received message " + message);
    });
});

if (connected) {
    client.send("/queue/test", {priority: 9}, "Important Message!");
}

if (connected) {
    client.disconnect();
}
```

MQTT WebSocket client

• Eclipse Paho JavaScript Client
  • http://www.eclipse.org/paho/

• Demo available at
  • http://localhost:8161/demo/mqtt
MQTT Websocket Example

```javascript
var client = new Messaging.Client("localhost", "61614", "myClient");
var connected = false;

client.onConnect = function() {
    connected = true;
    client.subscribe("test");
}

client.onMessageArrived = function(message) {
    console.log("Received message " + message);
}

client.connect();

if (connected) {
    var message = new Messaging.Message("Important Message!");
    message.destination = "test";
    client.send(message);
}

if (connected) {
    client.disconnect();
}
```
Striking The Balance
Striking the Balance

• Lots of possibilities, how to choose right?
• Native mobile apps should consider MQTT
• Do you need live updates in your browser?
• WebSockets ideal for HTML5 apps with limited number of users that needs instant update
• For everyone else, there's backend messaging
Stomp pitfall

• Short-lived connections
• Every page view, open a new connection to the broker
• Puts heavy load on the broker
• Eliminates all advance messaging mechanisms – message prefetches, producer flow control, etc.
Stomp configuration

<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry queue="" producerFlowControl="false">
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>

<transportConnectors>
  <transportConnector name="stomp+nio"
    uri="stomp+nio://0.0.0.0:61613?
    transport.closeAsync=false"/>
</transportConnectors>
Conclusion

• Messaging is not the thing of the enterprise anymore
• Things want to get integrated
• We have technology to do that TODAY!
AMA

• Links
  • Stomp
    • http://stomp.github.com
    • https://github.com/fusesource/stompjms
  • MQTT
    • http://mqtt.org
    • https://github.com/fusesource/mqtt-client
• Blog: http://sensatic.net
• Twitter: http://twitter.com/dejanb