REST

ActiveMQ implements a RESTful API to messaging which allows any web capable device to publish or consume messages using a regular HTTP POST or GET.

If you are interested in messaging directly from web browsers you might wanna check out our Ajax or WebSockets support or try running the REST examples

Mapping of REST to JMS

To publish a message use a HTTP POST. To consume a message use HTTP DELETE or GET.

ActiveMQ has a Servlet that takes care of the integration between HTTP and the ActiveMQ dispatcher.

NOTE: The example below requires servlet mapping on the URL. For posting without the servlet mapping, see examples further down.

You can map a URI to the servlet and then use the relative part of the URI as the topic or queue name. e.g. you could HTTP POST to

```
http://www.acme.com/orders/input
```

which would publish the contents of the HTTP POST to the orders.input queue on JMS.

Similarly you could perform a HTTP DELETE GET on the above URL to read from the same queue. In this case we will map the MessageServlet from ActiveMQ to the URI

```
http://www.acme.com/queue
```

and configure it to accept the URI as a queue destination. We can do similar things to support topic destinations too.

We can use the HTTP session to denote a unique publisher or consumer.

Note that strict REST requires that GET be a read only operation; so strictly speaking we should not use GET to allow folks to consume messages. Though we allow this as it simplifies HTTP/DHTML/Ajax integration somewhat.

For a more cleaner mapping of a simple transfer protocol to different languages, you might wish to take a look at Stomp.

Default configuration

Until version 5.8, REST API was part of the Web Samples and was mapped to http://localhost:8161/demo/message url. From 5.8 onwards, the API is available by default at http://localhost:8161/api/message url. Also, starting with 5.8, web server is secured by default (see Web Console for more information), so have that in mind when trying to use it. Examples below will assume new api location and secured web server.

Producing

You can produce by sending a POST request to the server, like

```
curl -u admin:admin -d "body=message" http://localhost:8161/api/message/TEST?type=queue
```

NOTE: If no type parameter is specified, the default is to create a topic. To change the default to queue, initialize the servlet with an init param in the `webapps/demo/WEB-INF/web.xml`

As shown below:

```
<servlet>
  <servlet-name>MessageServlet</servlet-name>
  <servlet-class>org.apache.activemq.web.MessageServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
  <init-param>
     <param-name>topic</param-name>
     <param-value>false</param-value>
  </init-param>
</servlet>
```

Alternate Producing Syntax

An alternative syntax for posting is supported, using the destination URL-encoded parameter; here are some examples:

```
# Send to queue orders.input:
curl -XPOST -d "body=message" http://admin:admin@localhost:8161/api/message?destination=queue://orders.input

# Send to topic orders.input:
curl -XPOST -d "body=message" http://admin:admin@localhost:8161/api/message?destination=topic://orders.input
```

Timeouts

When reading from a queue we might not have any messages. We can use a timeout query parameter to indicate how long we are prepared to wait for a message to arrive. This allows us to poll or block until a message arrives.

Couple this with HTTP 1.1 keep-alive sockets and pipeline processing we can have efficient access to JMS over HTTP.

Obviously if your client is Java then using ActiveMQ's JMS API is the fastest and most efficient way to work with the message broker; however, if you are not using Java or prefer the simplicity of HTTP then it should be fairly efficient, especially if your HTTP client supports keep-alive sockets and pipeline processing.

Consuming

When consuming messages using the REST API, you have to keep session alive between GET requests, or you'll create a separate consumer for every request and due to prefetch limit your succeeding call will hang.

For example, you can use `wget` to consume messages, like this:

```
wget --user admin --password admin --save-cookies cookies.txt --load-cookies cookies.txt --keep-session-
cookies  http://localhost:8161/api/message/TEST1?type=queue
```

Also, if you plan to have multiple consumer using REST, it's advisable to set prefetch size to 1 so all consumers have an equal chance of getting the message. You can do that by passing a special parameter to the `MessageServlet`

```
    <servlet>
        <servlet-name>MessageServlet</servlet-name>
        <servlet-class>org.apache.activemq.web.MessageServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
        <init-param>
                <param-name>destinationOptions</param-name>
                <param-value>consumer.prefetchSize=1</param-value>
        </init-param>
    </servlet>
```

in the `webapps/demo/WEB-INF/web.xml`

Alternate Consuming Syntax

As with producing, an alternative syntax for consuming messages is also supported, using the destination URL-encoded parameter; here are some examples:

```
# Send to queue orders.input:
curl -XGET http://admin:admin@localhost:8161/api/message?destination=queue://orders.input

# Send to topic orders.input:
curl -XGET http://admin:admin@localhost:8161/api/message?destination=topic://orders.input
```

Consuming without sessions

Since 5.2.0 you can use `clientId` parameter to avoid storing actual JMS consumer in the request session. When using this approach, you don't need to keep sessions alive between requests, you just need to use the same `clientId` every time.

```
wget --user admin --password admin http://localhost:8161/api/message/test?type=queue&clientId=consumerA
```

Every such call will use the same JMS consumer and deliver messages send to it by the broker.

In 5.4.1 it's also possible to unsubscribe the client. It's done by sending a POST call with `clientId` and `action=unsubscribe` parameters to the server, like

```
http://localhost:8161/demo/message/test?clientId=consumerA&action=unsubscribe
```

Consuming with selectors

As of ActiveMQ 5.4.0, you can use selectors when consuming using REST protocol. To do that, just specify the appropriate header with selector. To define a selector for the consumer, you have to provide it in an appropriate HTTP header. By default selector header name is `selector`, so the following example

```
wget  --user admin --password admin --save-cookies cookies.txt --load-cookies cookies.txt --keep-session-
cookies  --header="selector: test=2" http://localhost:8161/api/message/test?type=queue
```

should consume only messages that have `test` property set to 2.

You can change the name of the selector header using the `org.apache.activemq.selectorName` Servlet context property in `WEB-INF/web.xml`, such as

```
    <context-param>
        <param-name>org.apache.activemq.selectorName</param-name>
        <param-value>activemq-selector</param-value>
    </context-param>
```

For more info, take a look at RestTest

Consuming with One Shot Consumers

One shot consumption allows a REST call to receive a single message and then immediately close the associated consumer.

All of the examples so far lead to the servlet creating and holding on to consumers of the destination across multiple HTTP requests. Without care, these consumers could easily lead to confusion as messages are dispatched to them but then sit unused because the consuming HTTP session, or clientId, fails to connect to continue requesting the messages. One way around that problem is the use of one-shot consumers. Simple add the `?oneShot=true` option and the consumer is removed once the consumption completes; as follows:

```
curl -XGET http://admin:admin@localhost:8161/api/message?destination=queue://orders.input&oneShot=true
```

Note that interrupting the call while the consumer is waiting for a message, the consumer may remain until the server times out the HTTP request, or until a message finally arrives.

Content Types

By default messages are sent to the consumers with `text/xml` content type. Your REST-based application may expect JSON response instead of XML one. In that case, you can configure the servlet to send responses back by adding something like this

```
    <servlet>
        <servlet-name>MessageServlet</servlet-name>
        <servlet-class>org.apache.activemq.web.MessageServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
        <init-param>
                <param-name>defaultContentType</param-name>
                <param-value>application/json</param-value>
        </init-param>
    </servlet>
```

to your `WEB-INF/web.xml`.

A default content type can also be overridden using request headers. Specifying `xml=true` or `json=true` URL parameter you'll get a response with the desired content type.

```
wget --user admin --password admin http://localhost:8161/api/message/TEST?type=queue\&clientId=A\&json=true
```

Security

Since 5.7.0 release REST API can connect to the secured brokers. The API uses basic authentication header format to get username and password information.

For example, with curl you can do something like

```
curl -u system:manager -d "body=message" http://localhost:8161/demo/message/TEST?type=queue
```

Also, you might want to enable `ssl` for your connections. To do that, just uncomment SecureConnector in `conf/jetty.xml`

```
                <bean id="SecureConnector" class="org.eclipse.jetty.server.ssl.SslSelectChannelConnector">
                    <property name="port" value="8162" />
                    <property name="keystore" value="file:${activemq.conf}/broker.ks" />
                    <property name="password" value="password" />
                </bean>
```

Rest Management

Starting with version 5.8 we provide a REST management API for the broker. Using Jolokia JMX-HTTP bridge it's possible to access all broker metrics (like memory usage) and execute management operations (like purging queues) using REST API. By default the management API is exposed at http://localhost:8161/api/jolokia/ URL. So you can for example get basic broker data with

```
wget --user admin --password admin --auth-no-challenge http://localhost:8161/api/jolokia/read/org.apache.
activemq:type=Broker,brokerName=localhost
```

or to be more specific, total consumer count with

```
wget --user admin --password admin --auth-no-challenge http://localhost:8161/api/jolokia/read/org.apache.
activemq:type=Broker,brokerName=localhost/TotalConsumerCount
```

For more information on Jolokia protocol, see its reference manual. An API like this makes it easy to script monitoring and management operations against the broker, see also How can I monitor ActiveMQ?

Gotcha's and other trivia

1. Missing "body" parameter

In the curl POST examples above, the use of "body=..." is critical. If this is not specified in front of the message body contents, the web servlet will attempt to read the body from the request (rather than from the -d parameter), and this will result in the following exception:

```
java.lang.IllegalStateException: STREAMED
at org.eclipse.jetty.server.Request.getReader(Request.java:898)
at org.apache.activemq.web.MessageServletSupport.getPostedMessageBody(MessageServletSupport.java:347)
at org.apache.activemq.web.MessageServlet.doPost(MessageServlet.java:126)
...
```

However, one option in this case would be to specify the content type explicitly:

```
curl -u admin:admin -d "hello world $(date)" -H "Content-Type: text/plain" -XPOST http://localhost:8161/api
/message?destination=queue://abc.def
```