



How to: Emulate Amazon Simple Queue Service (SQS) for Testing Purposes

A guide to creating a dynamic wiremock for SQS



Miro Barsocchi

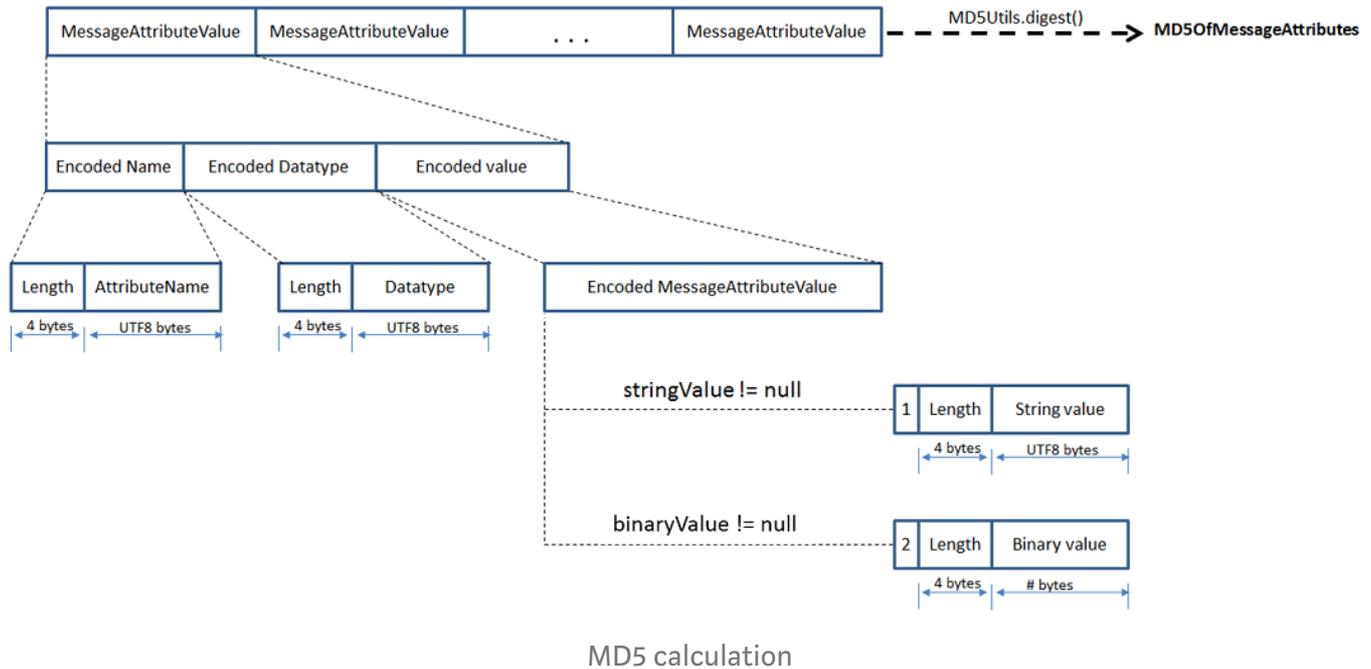
Jul 13, 2018 · 3 min read

Amazon Simple Queue Service (**SQS**) is a fully managed message queuing service from Amazon.

If you want to integrate it, you may also want **to test** if it works. SQS has probably a sandbox to test it but if you don't want to get stuck searching for it, or you don't want to get crazy with login procedures, you can use a mock service that acts as an SQS queue* .

Why do we need this? Why can't we use a simple Wiremock that always responds "200, OK"?

You can't do it because the response of SQS is a dynamic response dependent from the request, and it's based on some MD5 calculation.



What we have done is **an extension of Wiremock** that **always** responds something like "200, OK", with all the fields correctly calculated for SQS. It was both easy and not-so-easy, because every easy task it's never a "It's just ...".

In this case, it was more of a "it's just not enough to..." — just *spinning a docker container up* or *adding a class to the classpath* would simply not do.

How To, Step by step guide

First of all we need to define a class that extends the `ResponseDefinitionTransformer` class with two easy to implement methods:

```
1 public class SQSMock extends ResponseDefinitionTransformer {
2   @Override
3     public boolean applyGlobally() {
4       return true;
5     }
6   @Override
7     public String getName() {
8       return this.getClass().getSimpleName();
9     }
10  [.....]
```

applyGlobally that returns true, means that “all the requests will receive this response”.

getName it's a method that Wiremock needs. Then you must implement the real method that responds correctly, the method is **transform**

```
1  @Override
2  public ResponseDefinition transform(Request request, ResponseDefinition responseDefinition) {
3      String xmlString = null;
4      try {
5          xmlString = this.calculateResponse(request);
6      } catch (TransformerConfigurationException ex) {
7          Logger.getLogger(SQSMock.class.getName()).log(Level.SEVERE, null, ex);
8      } catch (TransformerException | ParserConfigurationException | URISyntaxException ex) {
9          Logger.getLogger(SQSMock.class.getName()).log(Level.SEVERE, null, ex);
10     }
11     return new ResponseDefinitionBuilder()
12         .withHeader("Content-Type", "application/xml")
13         .withStatus(200)
14         .withBody(xmlString)
15         .build();
16 }
```

transform hosted with ❤️ by GitHub

[view raw](#)

calculateResponse does the dirty job, and it's a dirty method (I know it could be refactored...)

```
1  private String calculateResponse(Request request) throws ParserConfigurationException, TransformerException {
2      String md5ofBody = "";
3      String md5ofAttributes = "";
4      String valueDecoded = "";
5      MessageDigest md = MessageDigest.getInstance("MD5");
6      String[] split = request.getBodyAsString().split("&");
7      Map<String, String> map = new HashMap<String, String>();
8
9      Arrays.asList(split).stream().filter(el -> el.startsWith("Message")).forEach(
10         el -> {
11             String[] splitAtEqual = el.split("=");
12             map.put(splitAtEqual[0], splitAtEqual[1]);
13         });
14     if (map.get("MessageBody") != null) {
15         valueDecoded = java.net.URLDecoder.decode(map.get("MessageBody"), CHARACTER_ENCODING);
16         md.update(valueDecoded.getBytes());
17         md5ofBody = DatatypeConverter.printHexBinary(md.digest());
18         md.reset();
19         map.remove("MessageBody");
20     }
21 }
```

```

20     List<SimplifiedMessageAttribute> simplifiedArrayOfAttributes = new ArrayList<Simp
21     if (!map.isEmpty() && map.size() % 3 == 0) {
22         int numberOfAttributes = map.size() / 3;
23         for (int i = 1; i <= numberOfAttributes; i++) {
24             String name = java.net.URLDecoder.decode(map.get(MESSAGE_ATTRIBUTE + i +
25             String value = java.net.URLDecoder.decode(map.get(MESSAGE_ATTRIBUTE + i
26             String dataType = java.net.URLDecoder.decode(map.get(MESSAGE_ATTRIBUTE +
27             simplifiedArrayOfAttributes.add(new SimplifiedMessageAttribute(name, valu
28         }
29     simplifiedArrayOfAttributes.sort((o1, o2) -> {
30         return o1.getName().compareTo(o2.getName());
31     });
32     for (SimplifiedMessageAttribute attr : simplifiedArrayOfAttributes) {
33         updateLengthAndBytes(md, attr.getName());
34         updateLengthAndBytes(md, attr.getDataType());
35         md.update(String_Type_Field_Index);
36         updateLengthAndBytes(md, attr.getValue());
37     }
38     md5fAttributes = DatatypeConverter.printHexBinary(md.digest());
39 }
40 [.....]
41 }

```

gistfile1.txt hosted with ❤️ by GitHub

[view raw](#)

This method gets the body of the request, splits it because it's an url-like body, and then calculates the MD5 of all the attributes, types and values according to [SQS documentation](#).

updateLengthAndBytes it's a method to update the MD5 calculation.

```

1 private static void updateLengthAndBytes(MessageDigest digest, String str) throws Unsuppo
2     byte[] utf8Encoded = str.getBytes(CHARACTER_ENCODING);
3     ByteBuffer lengthBytes = ByteBuffer.allocate(INTEGER_SIZE_IN_BYTES).putInt(utf8Er
4     digest.update(lengthBytes.array());
5     digest.update(utf8Encoded);
6 }

```

updateLengthAndBytes hosted with ❤️ by GitHub

[view raw](#)

To be honest, we also created an additional class **SimplifiedMessageAttribute**:

```

1 public class SimplifiedMessageAttribute {
2     final String name;
3     final String value;
4     final String dataType;
5     public SimplifiedMessageAttribute(String name, String value, String dataType) {

```

```

6         this.name = name;
7         this.value = value;
8         this.dataType = dataType;
9     }
10    public String getName() {
11        return name;
12    }
13    public String getValue() {
14        return value;
15    }
16    public String getDataType() {
17        return dataType;
18    }
19 }

```

SimplifiedMessageAttribute hosted with ❤️ by GitHub

[view raw](#)

After this step, you can build an XML document (the implementation of this is not in this post because it's cumbersome) and include the calculated fields (**md5OfMessageBody** and **md5OfMessageAttributes**). For other fields, **requestId** and **messageId**, we used hard coded values. The final result should be something like this:

```

<SendMessageResponse>
  <SendMessageResult>
    <MD5OfMessageBody>fafb00f5732ab283681e124bf8747ed1</MD5OfMessageBody>
    <MD5OfMessageAttributes>3ae8f24a165a8cedc005670c81a27295</MD5OfMessageAttributes>
    <MessageId>5fea7756-0ea4-451a-a703-a558b933e274</MessageId>
  </SendMessageResult>
  <ResponseMetadata>
    <RequestId>27daac76-34dd-47df-bd01-1f6e873584a0</RequestId>
  </ResponseMetadata>
</SendMessageResponse>

```

Response Example for SQS

After this, you can build your project and obtain a jar.

How can we integrate this into Wiremock?

That's not so easy. What I've found is that you have to move your jar in the same path of the Wiremock jar and start the Wiremock with:

```
1 java -cp "*" com.github.tomakehurst.wiremock.standalone.WireMockServerRunner - port 8089
```

run wiremock hosted with ❤️ by GitHub

[view raw](#)

Wiremock has no plugin mechanisms, so what you are doing with this command is:

with `-cp "*"` you are telling Java “take all the jars in this folder”

... your main class is here:

`com.github.tomakehurst.wiremock.standalone.WireMockServerRunner`

... and use this extension: `your.package.name.SQSMock`

From this point onwards, you can use it directly, build a docker container or whatever you want.

IT is not an easy task and, from my point of view, every layer added to simplify things is a new topic to learn and understand.

During this journey we have learnt about SQS, wiremock, mvn, unit test and obviously Java. I want to thank:

- **Giuseppe Gaeta** (Tech Lead)
- **Matteo Moci** (Software Dev Engineer II)
- **Camillo Quatrini** (QA Evangelist)
- **Tiziana Battiston** (Travel Agencies Manager, proofreader)

Ref:

- SqS documentation: <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-message-attributes.html>
- Wiremock Extension: <http://wiremock.org/docs/extending-wiremock/>

* There are some restrictions: such service doesn't work for not-string attributes.

Thanks for reading





Miro Barsocchi: Software tester and also Electronic engineer, radio speaker, actor, surfer, barman, but only two of these are seriously. You can find me on [Twitter](#) or [Github](#) or elsewhere.

Sqs

Wiremock

Testing

AWS

Software Engineering

Medium

[About](#) [Help](#) [Legal](#)

Get the Medium app

 A button that says 'Download on the App Store', and if clicked it will lead you to the iOS App store

 A button that says 'Get it on, Google Play', and if clicked it will lead you to the Google Play store