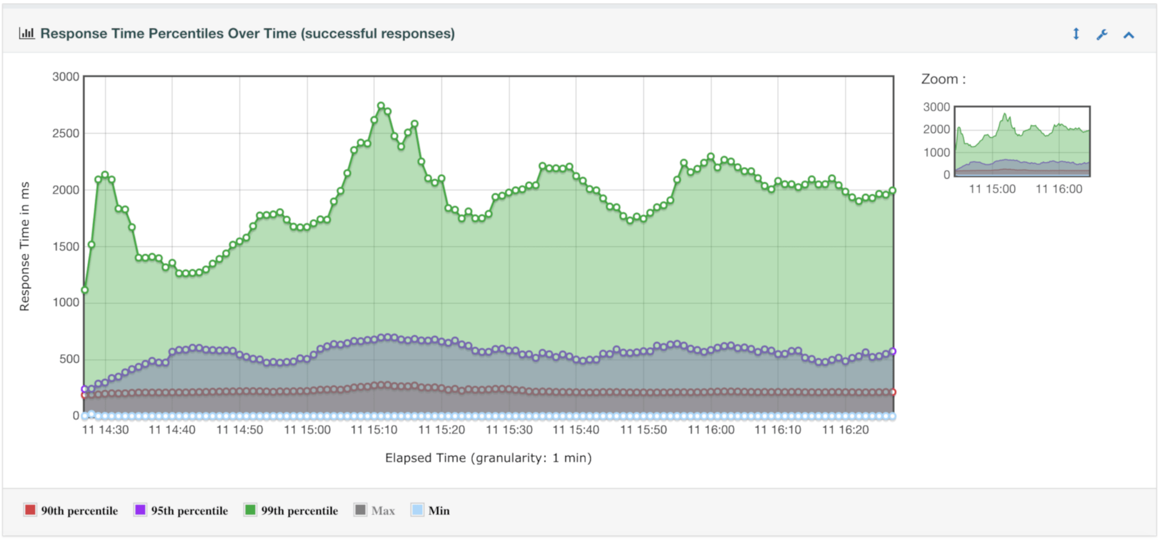
Master Apache JMeter

From load testing to DevOps



Antonio Gomes Rodrigues
Bruno Demion (Milamber)
Philippe Mouawad

APACHE JMeter™

Master Apache JMeter From load testing to DevOps.

Prefaced by Alexander Podelko

Antonio Gomes Rodrigues, Philippe Mouawad, and Milamber

This book is available at
https://leanpub.com/master-jmeter-from-load-test-to-devops

This version was published on 2025-02-28

ISBN 978-2-9555036-2-1

Tweet This Book!

Please help Antonio Gomes Rodrigues, Philippe Mouawad, and Milamber by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought Master Apache JMeter: From load testing to #Devops written by 3 committers of the project @ra0077, @milamberspace, @philmdot on https://leanpub.com/master-jmeter-from-load-test-to-devops

The suggested hashtag for this book is #jmeter.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#jmeter

In no particular order.

Thanks to Vladimir Sitnikov for his feedback.

Thanks to Mark Tomlinson for his feedback.

Thanks to Felix Schumacher for his review.

Contents

Copyright **1**

Preface by Alexander Podelko **2**
    Alexander Podelko bio 5

Presentation of the authors **6**
    Antonio Gomes Rodrigues 6
    Bruno Demion (Milamber) 6
    Philippe Mouawad (Philippe M.) 7

About the reviewers **9**
    Felix Schumacher 9

Load Testing Message Oriented Middleware (MOM) via JMS **10**
    What we will learn 10
    A bit of theory 10
        Asynchronous messages 10
        Decoupling 11
            Technology decoupling 11
            Geographical Decoupling (*Location transparency*) 12
            Time decoupling 13
        Back pressure 13
        Communication models 14
        Types of delivery semantics 16
            At Most Once 17
            At Least Once 17
            Exactly Once 17
            Summary 17

What is a message composed of? 18
Set up with JMeter 19
Installation of MOM libraries 19
JMS Point-to-Point 19
JMS Resources and JNDI Properties 20
Communication style 23
Priority 25
Use non-persistent delivery mode? 25
JMS Selector 28
Example with Apache ActiveMQ Classic 30
Publish/Subscribe 32
JMS Publisher 33
JMS Subscriber 34
Methodology 38
Put into practice with JMeter 41
Example 1: Test the configuration of a MOM server with point-to-point 41
Example 2: Test the performance of our consumer with JMS Publish/-Subscribe 46
Example 3: Testing the configuration of a MOM server with Publish/-Subscribe 51
Example 4: Test any MOM 61
Conclusion 62

Copyright

Preface by Alexander Podelko

An important event, directly related to this book, happened recently and probably went unnoticed. It appears that Apache JMeter became the most popular load testing tool. In 2014, I was preparing a presentation about load testing tools and criteria for their selection. One criterion was the existence of ecosystem (documents, expertise, people, services, etc). It may be not the defining factor, but it is an important factor to consider. To evaluate such ecosystems, in absence of more sophisticated data, I used the number of documents Google finds and the number of jobs Monster finds mentioning each product.

LoadRunner (then an HP product) clearly had first place in both categories with JMeter following not too far. SilkPerformer (then Borland) trailed third far behind. But now, in 2018, JMeter appears to be well ahead of LoadRunner (now MicroFocus) in both the number of documents and the number of jobs mentioning it, apparently becoming the most popular load testing tool. Of course, it doesn't mean that JMeter became the best tool for every task, but its popularity, in addition to being an open source tool, definitely gets it high in the list of options to consider. It is also very important for an open source project to attract people who will work to improve it, thus ensuring the future of the product.

Another interesting trend is that JMeter scripts become a de facto standard, and many SaaS tools are built on the top of JMeter or at least support JMeter scripts. These tools complement JMeter in many important ways and bring its functionality and services to a new level, allowing to compete with commercial products in more sophisticated environments.

Several books about JMeter were published before, but this one is the first one from JMeter contributors who know its internals and are renowned experts in the area. JMeter is not a trivial product. It has a lot functionality, but it may be implemented by many different ways and it is not always easy to figure out the best way to do it. While, as it was already mentioned, there is an enormous number of posts on the Internet discussing some aspects of JMeter, the problem is that most of them are for beginners. And while there is more advanced stuff too, you better exactly know what you are looking for, it may be hard to find it looking for generic terms. This is where the book is invaluable setting a framework of knowledge and notions, so you at least would understand what you should be looking for.

The book, with all that valuable information in one place, is a must for everybody who is seriously working with JMeter. It may be a little too condensed for absolute beginners (although, as already mentioned, there is a lot of introductory materials on the Internet that can help here), but it is the best you can find if you already have some performance testing experience and need to get further. In particular, it would be invaluable to people who want to expand their JMeter knowledge into advanced topics or switch to JMeter from other load testing tools.

In particular, JMeter has many integrations with other popular tools and a large number of plugins. It quite could be that somebody already solved the problem you are confronting, so you may save a lot of efforts re-using the solution. It is great that the book is not limited by the core JMeter functionality and discusses available components and integrations when appropriate. While it is impossible to cover everything that is available in detail, the advantage of the book is that it puts everything into a system, allowing the reader to understand relations between different parts and technologies.

Load testing is an important part of the performance engineering process. However, the industry is changing and load testing should adjust to these changes. A stereotypical, last-moment performance check is not enough anymore. Performance testing should be interwoven into development process, starting early and continuing through the whole lifecycle. The importance of the transformation is stressed in the subtitle of the book, **From load testing to DevOps**, moving from standalone load testing, a mere step at the end of software development cycle, to performance testing fully integrated into DevOps. And while we are not fully there yet, the **Integration of JMeter in the DevOps tool chain** chapter directly dives into what can be done right now.

DevOps, putting together the development and operations sides, is supposed to drastically improve feedback from production to development and free flow of performance information both ways. So a holistic approach to performance should be one of its main advantages. Unfortunately, it doesn't look like such a holistic approach happens often. Rather it looks like "DevOps" teams just drop more sophisticated parts of performance engineering (and performance testing usually gets into that category) and rely more on reactive approach to performance issues, concentrating more on quick fixing of issues than on their prevention. Still, load testing is a very important way of risk mitigation and it can't be fully replaced by other performance engineering activities.

First, there are always risks of crashing a system or experiencing performance issues under heavy load, the only way to mitigate them is to actually test the system. Even stellar performance in production and a highly scalable architecture don't guarantee that it won't crash under a slightly higher load.

It is important to notice load testing doesn't completely guarantee that the system won't crash: for example, if the real-life workload is different from what was tested. So you need to monitor the production system to verify that your test workload is close enough. But load tests considerably reduce the risk if they are carried out correctly (and, of course, can be completely useless and misleading if they are not).

Another important benefit of load testing is to verify how changes affect multi-user performance. The impact on multi-user performance is generally not proportional to what is observed with single-user performance and can often be counter-intuitive. Sometimes, improving single-user performance can lead to a degradation of multi-user performance. The more complex the system, the more exotic multi-user performance problems can occur.

Another value of load testing is to provide a reliable and reproducible way to apply a multi-user load necessary for performance optimization and troubleshooting. You apply exactly the same workload and see if the change makes a difference. In most cases, you cannot do this in production when the workload changes - so you never know if the result is a change in code or a change in workload (except, perhaps, a rather rare case of very homogeneous and manageable workloads when you can apply a very precisely measured portion of the actual workload). And, of course, a reproducible workload greatly simplifies debugging and checking multi-user problems.

In addition, given current trends in system self-regulation (such as auto-scaling or load-dependent service level changes), load tests are required to verify this functionality. You must apply a heavy load to see how auto-scaling will work. Load testing thus becomes a means of testing the functionality of the system, blurring the traditional division between functional and non-functional testing.

You will find further examples of different types of performance tests and their links to different aspects of DevOps in the book. Although the book does not focus on the theoretical aspects of performance testing, it provides sufficient theoretical information to understand the concepts discussed and their practical applications in JMeter. Practical examples of integrating performance testing with DevOps are all the more important as this is a rather new field of expertise. and is probably the main challenge of performance testing at the moment. Thus, the book leads the reader through the basics of working with today's most popular load testing tool JMeter, through more advanced aspects of the tool and performance testing in general until its complete integration into DevOps.

The authors decided to publish the book on LeanPub, in order to keep it up to date and enrich it. There have already been some updates and others are in progress. Although I would like to have a printed copy, I must admit that it is probably a better choice for readers, as many improvements are made to JMeter with each version, not to mention the growing ecosystem of plug-ins, extensions, tools and products. Any paper version would be outdated soon enough - but with LeanPub, authors have the opportunity to keep the book up to date, which is a great service to the community.

Alexander Podelko bio

Over the last twenty years Alex Podelko supported major performance initiatives for Oracle, Hyperion, Aetna, and Intel in different roles including performance tester, performance analyst, performance architect, and performance engineer. Currently he is Consulting Member of Technical Staff at Oracle, responsible for performance testing and tuning of Hyperion (a.k.a. Enterprise Performance Management and Business Intelligence) products. Before specializing in performance, Alex led software development for Rodnik Software. Having more than thirty years of overall experience in the software industry, he holds a PhD in Computer Science from Gubkin University and an MBA from Bellevue University.

Alex periodically talks and writes about performance-related topics, advocating tearing down silo walls between different groups of performance professionals. He currently serves as a board director for the Computer Measurement Group (CMG), a worldwide organization of performance and capacity management professionals.

Presentation of the authors

Antonio Gomes Rodrigues

Antonio Gomes Rodrigues is an expert in the field of application performance. His missions led him to work:

- On the performance of high traffic websites
- On the performance of an application for brokers
- On the performance of rich clients, cloud applications, WEB applications, etc.
- With various profilers: *JProfiler*, *Yourkit*, *PerfView*, etc.
- With various APM: *Dynatrace*, *AppDynamics*, *NewRelic*, etc.
- With various load testing tools: *JMeter*, *LoadRunner*, *Neoload*, etc.
- In various missions: load tests, implementation of performance strategies, training, performance audits, troubleshooting, etc.

He is currently a committer and a PMC member of the JMeter project[1] within the Apache Software Foundation[2].

Bruno Demion (Milamber)

Bruno Demion, better known in the JMeter community under the pseudonym **Milamber** is a French computer scientist living in Morocco since 2002, currently living in Temara (near Rabat).

He works in a technology consulting company, as a partner, architect and senior technical expert on web and cloud technologies.

[1]http://jmeter.apache.org/
[2]http://www.apache.org/foundation/how-it-works.html#what

Thanks to his work and passion, IT, Milamber has strong skills in the field of performance, troubleshooting, IT security as well as technical architectures for web and cloud solutions.

Since December 2003, he has been working with JMeter to perform load tests in various performance missions and also gives training on this topic. He contributes as much as possible to the JMeter project on his free time, especially on the French translation of the graphical interface, bug fixes and some changes (proxy https, new results tree, icon bar, etc.).

He is currently a committer and a PMC member of the JMeter project[3] within the Apache Software Foundation[4]. It is also official ASF member[5]. Its Apache ID is milamber[6].

Milamber also has a personal blog[7] with many articles and tutorials about JMeter, some of which inspired this book.

Philippe Mouawad (Philippe M.)

Philippe Mouawad is a technical expert and architect in JEE and Web environments within the company Ubik-Ingenierie. He has been using JMeter since 2009 as part of performance improvements missions, load testing of intranet or e-commerce websites and trainings on JMeter.

He has been contributing to JMeter since 2009 first through patches and then as a ‘committer’ and member of Project Management Committee at Apache. Among his main contributions are the **CSS selector Extractor**, the **Boundary Extractor**, the **Backend Listener** (allowing to interface among others *Graphite*, *InfluxDB* or *ElasticSearch*), part of the Web reporting feature and the optimization of the performances of the core and its stabilization and various ergonomic improvements, to his credit more than 400 bugs/improvements.

He also contributes to the JMeter-Plugins[8] project, among his contributions are **Redis DataSet**, **Graphs Generator Listener** and various patches to different plugins.

[3]http://jmeter.apache.org/
[4]http://www.apache.org/foundation/how-it-works.html#what
[5]http://www.apache.org/foundation/how-it-works.html#roles
[6]http://people.apache.org/~milamber/
[7]http://blog.milamberspace.net/
[8]https://jmeter-plugins.org/

He also manages the JMeter Maven Plugin[9] project, he has been managing it since version 2.3.0 ensuring its compatibility with last JMeter releases and improving its dependencies management and reporting mechanism.

He is currently a committer and a PMC member of the JMeter project[10] within the Apache Software Foundation[11]. His Apache ID is pmouawad[12].

He is also a lead developer of the Ubik Load Pack[13] solution, a set of Enterprise Plugins which provides support for protocols that are not natively supported by JMeter. Finally, he contributes to the Ubik-Ingenierie blog[14].

[9]https://github.com/jmeter-maven-plugin/jmeter-maven-plugin
[10]https://jmeter.apache.org/
[11]http://www.apache.org/foundation/how-it-works.html#what
[12]http://people.apache.org/~pmouawad/
[13]https://ubikloadpack.com
[14]https://www.ubik-ingenierie.com/blog/

About the reviewers

Felix Schumacher

Felix is a committer in JMeter project since October 2014 and PMC member since February 2015. He is also a committer on Apache Tomcat. He has a diploma in mathematics, but found working in IT more appealing. Since he became a developer on JMeter, he has been active in all fields, from bug fixing, to multiple enhancements, tests, documentation and quality improvements.

Load Testing Message Oriented Middleware (MOM) via JMS

What we will learn

In this chapter we will learn:

- How a MOM works
- Different patterns of MOM
- Load test a point-to-point MOM
- Load test a Publish/Subscribe MOM
- Use *JSR223 Sampler* to extend our test

A bit of theory

It is important to understand the technology you are testing to have realistic tests and relevant analyses.

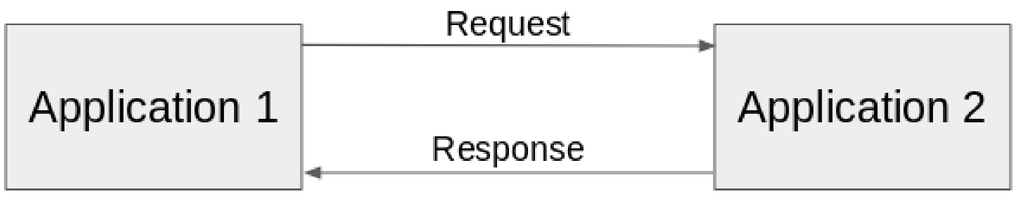
As the architectures become more and more distributed (microservices, Cloud, etc.), it is common to have a Message-Oriented Middleware service (or *MOM*) allowing exchange of messages/events.

Such architectures have numerous advantages, here are some of them.

Asynchronous messages

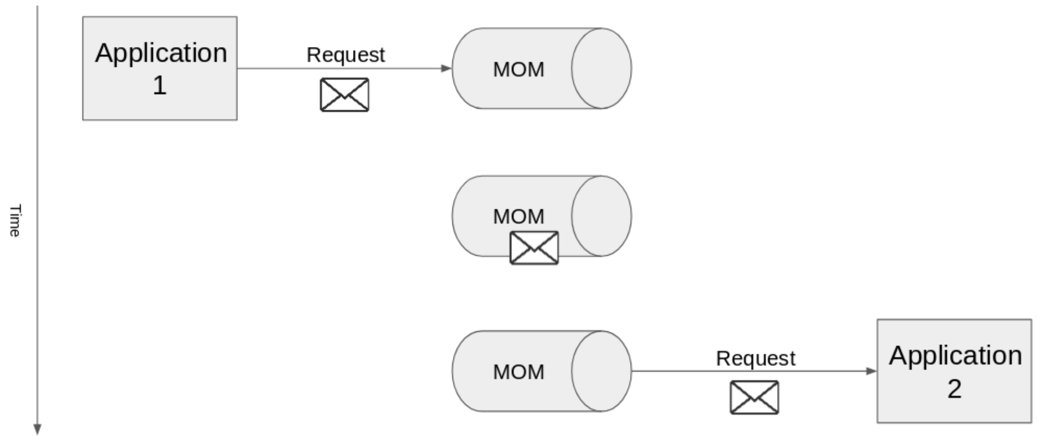
The first is the ability to have asynchronous messages.

In a synchronous system, the sender (producer) of the message must wait for the response before continuing.



Synchronous architecture

In an asynchronous system, the producer of the message can continue his processing after sending the message without waiting for a response from receiver.



Asynchronous architecture

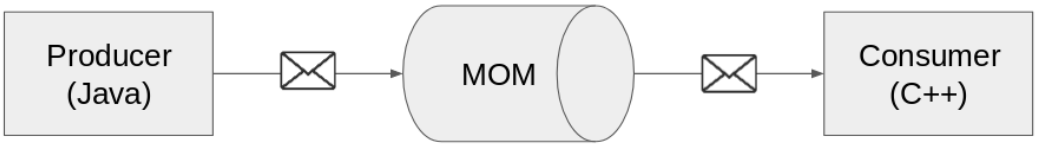
In this mode, the MOM will act as an intermediate and store the message until it is delivered to the application 2.

Decoupling

This proxy role allows a weak coupling between the different entities of the application.

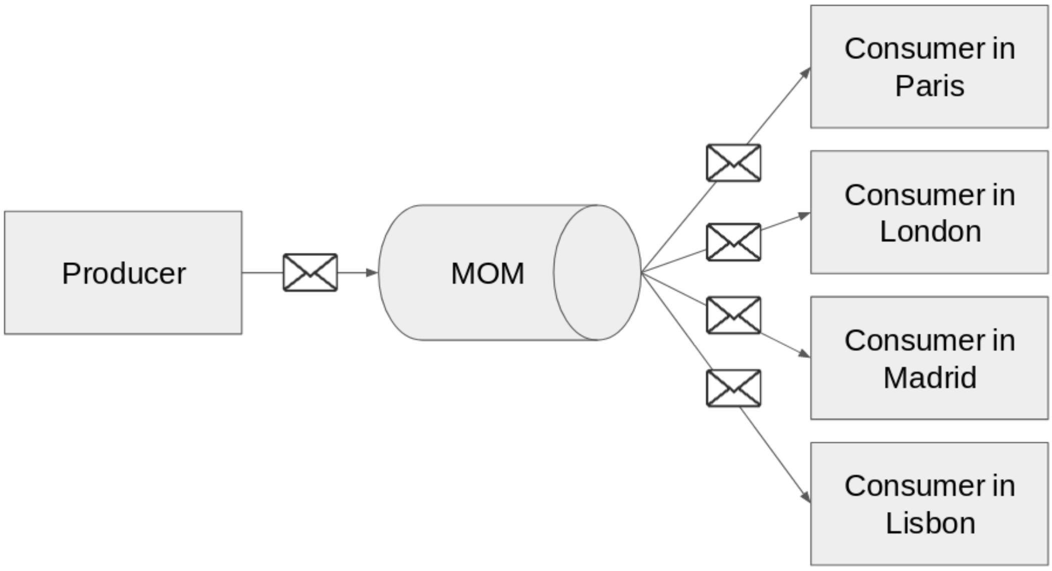
Technology decoupling

For example, we can have a producer in Java and a consumer in C++.



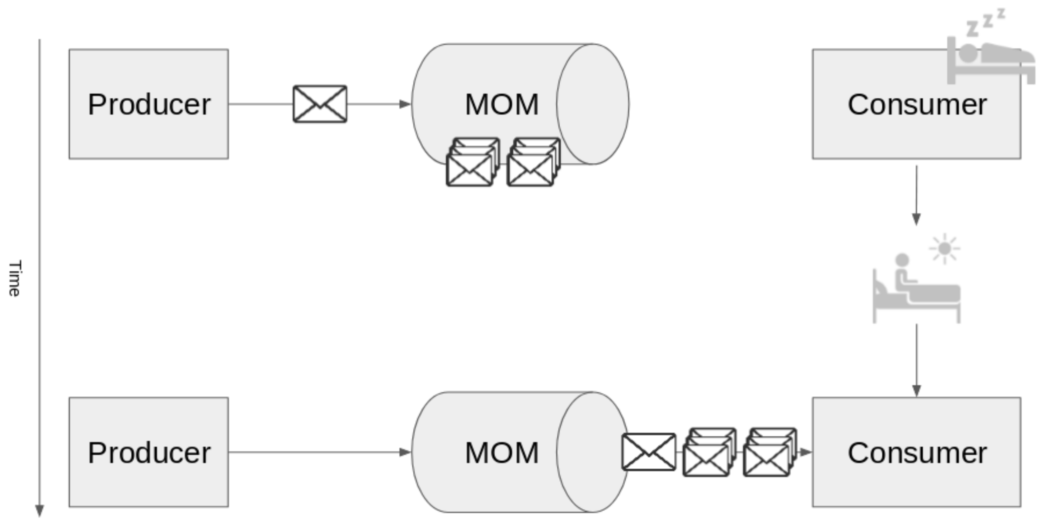
Technology decoupling

Geographical Decoupling (*Location transparency*)



Producer doesn’t know about consumers location

Time decoupling



Producer can send data while consumer is unavailable

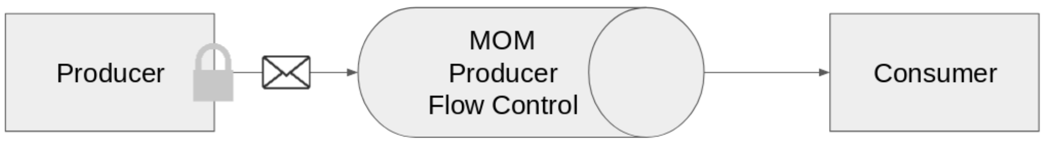
Back pressure

Producer Flow Control feature allows us to implement the *back pressure* pattern.

This pattern slows down the producer when the consumer(s) do not follow the rate of message delivery in the MOM.

To do it, the MOM will use a locking mechanism to block the producer.

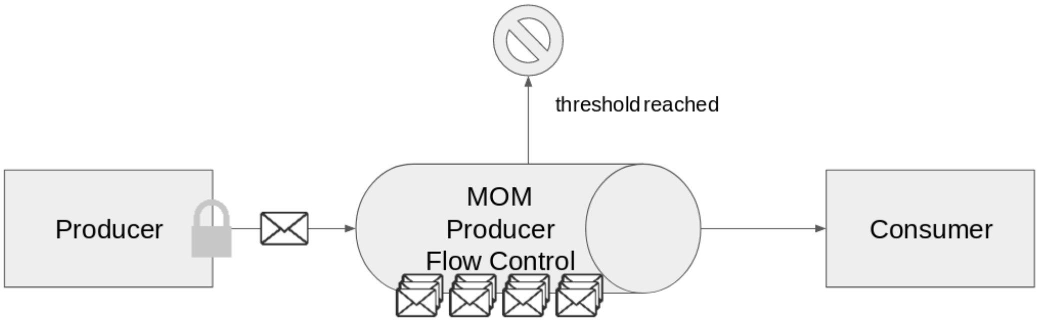
When a message is sent to the MOM by the producer, it waits (this is configurable) for an *ACK* (acknowledgement of receipt) from the MOM before proceeding to the next step (by removing the lock).



Producer Flow Control: Lock Installation

Usually, this lock disappears quickly.

When a threshold on the MOM (number of messages, memory usage, etc.) is reached, the MOM keeps the lock to block the producer.



Producer Flow Control: triggering mechanism

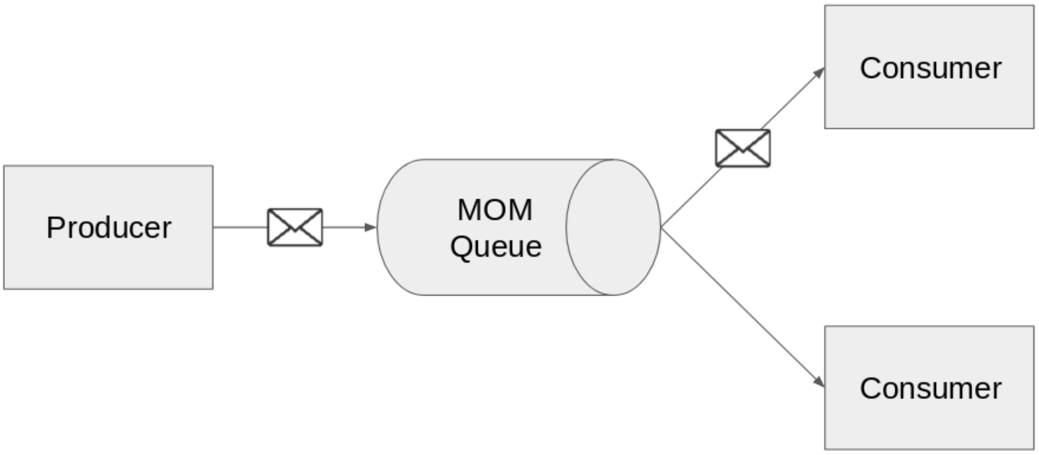
This will give consumers time to consume messages in the MOM which can then send the ACK to the producer to remove the lock.

Communication models

There are three main models of communication:

- Point-to-point communication (*direct exchange*)

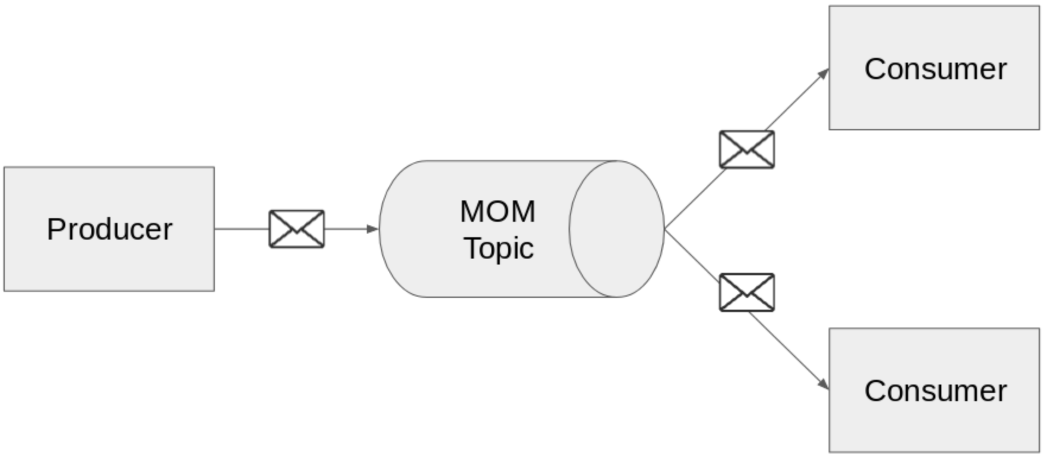
Here, the producer sends his message in a queue that is dedicated to him.
The consumer (who is unique) of this message consumes it in the FIFO order (First In, Frist Out).



Point-to-point communication

- Publication/Subscription communication (*Publish/subscribe* or *Topic exchange*)

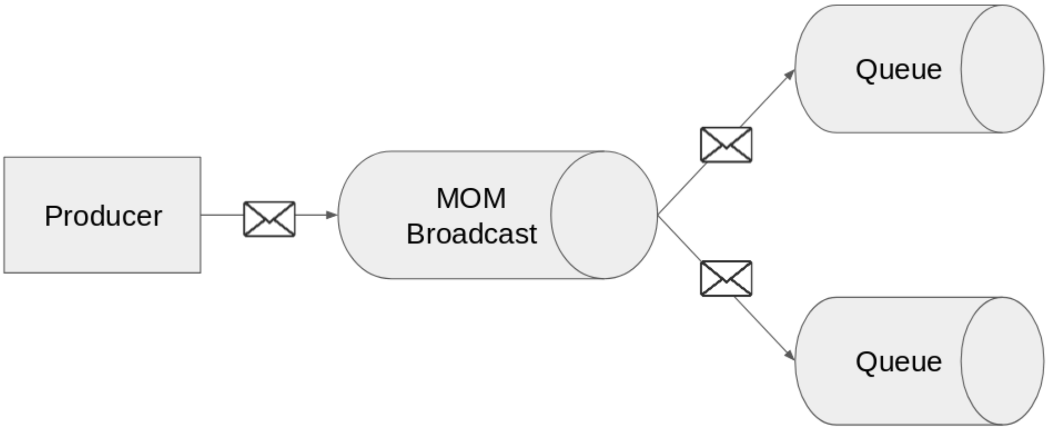
As its name suggests, the producer sends his messages about a subject in a Topic, they are then retrieved by the subscribers to this subject (the message disappears from the topic once all consumers have retrieved it).
Several consumers can subscribe to the same topic.



Communication Publish/subscribe

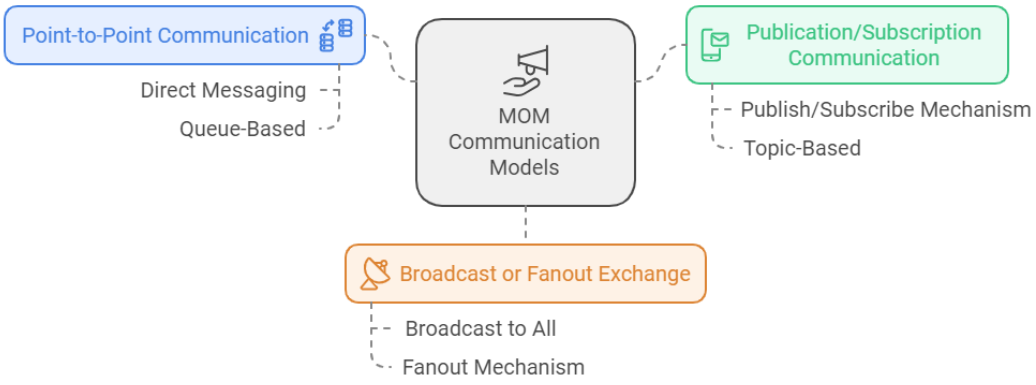
- *Broadcast* or *Fanout exchange*

In this mode, several MOMs follow each other. The first being in charge of distributing the message to all others.



Broadcast Communication

- Summary


Point-to-Point Communication
Direct Messaging
Queue-Based
MOM Communication Models
Publication/Subscription Communication
Publish/Subscribe Mechanism
Topic-Based
Broadcast or Fanout Exchange
Broadcast to All
Fanout Mechanism

Types of delivery semantics

To have a robust and reliable messaging system and answer the question: "*How to guarantee that a message has been correctly delivered to its destination?*" we need to understand the three types of delivery semantics.

At Most Once

In *"At Most Once"*, a message is delivered to the consumer zero or once only.
This means that the message can be lost during transmission, but will never be delivered more than once.

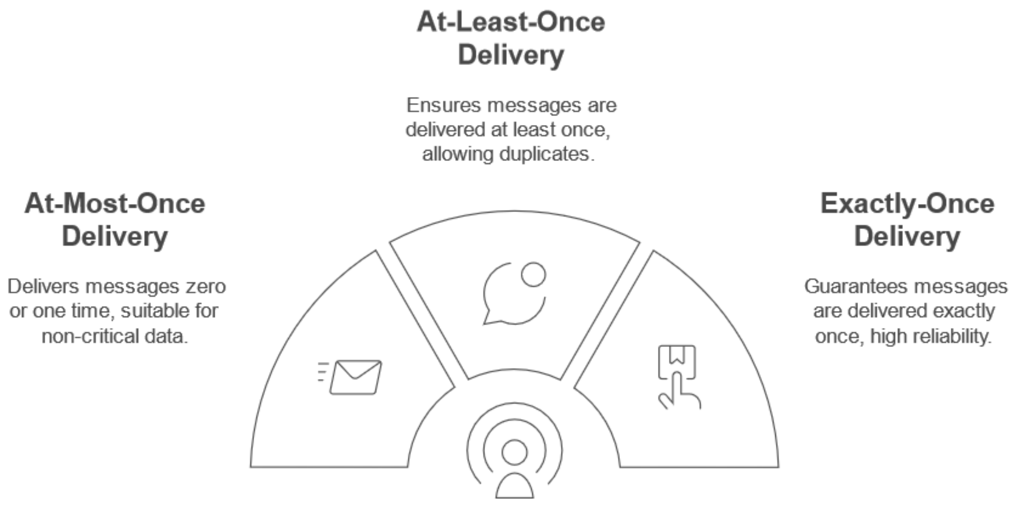
At Least Once

In *"At Least Once"*, a message is delivered to the consumer one or more times.
This means that the message can be delivered several times, resulting in potential duplicates.

Exactly Once

In *"Exactly Once"*, a message is delivered to the consumer exactly once.

Summary


At-Least-Once Delivery
Ensures messages are delivered at least once, allowing duplicates.
At-Most-Once Delivery
Delivers messages zero or one time, suitable for non-critical data.
Exactly-Once Delivery
Guarantees messages are delivered exactly once, high reliability.

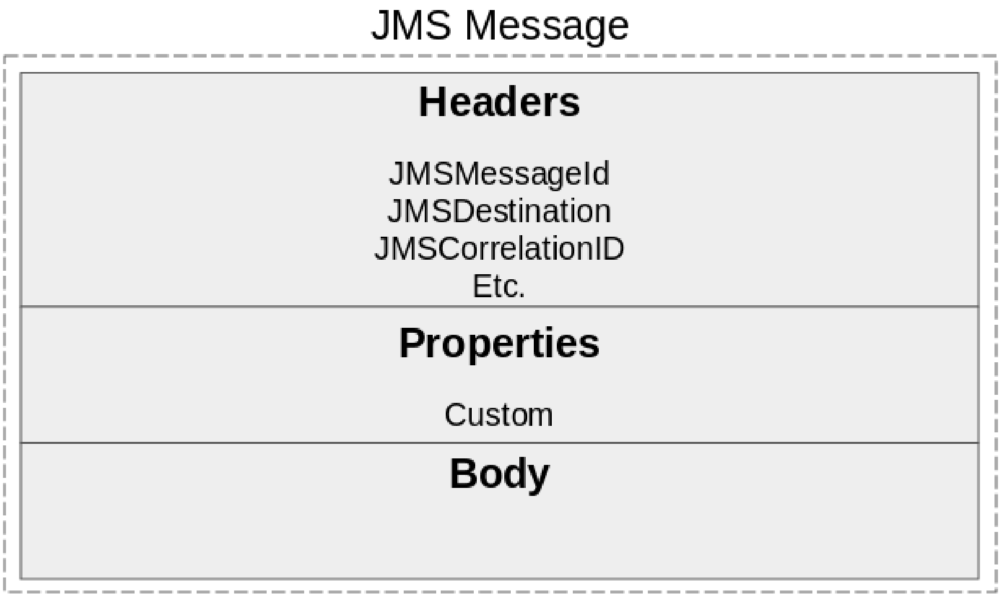
Messaging System Delivery Semantics

What is a message composed of?

In order to make the best use of JMeter's extractors and assertions, let's look at the structure of a message.

A message is divided into three parts:

- *Header* contains metadata about the message. The keys of the headers are part of JMS standard. The main fields in the header are *JMSMessageID* (unique identifier of the message), *JMSDestination* (identification of the queue/topic of the message) and *JMSCorrelationID* (links the current message to other messages).
- *Properties* also contain metadata about the message. But properties can be specific to the MOM provider or the application.
- **Message Body** (text, binary, etc.) is contained in the last part.



Composition of a message

Now that you understand how MOM work, let's study how to load test them using JMeter.

Set up with JMeter

JMeter is a Java program, as such it accesses MOM through Java Messaging Service (JMS) API.

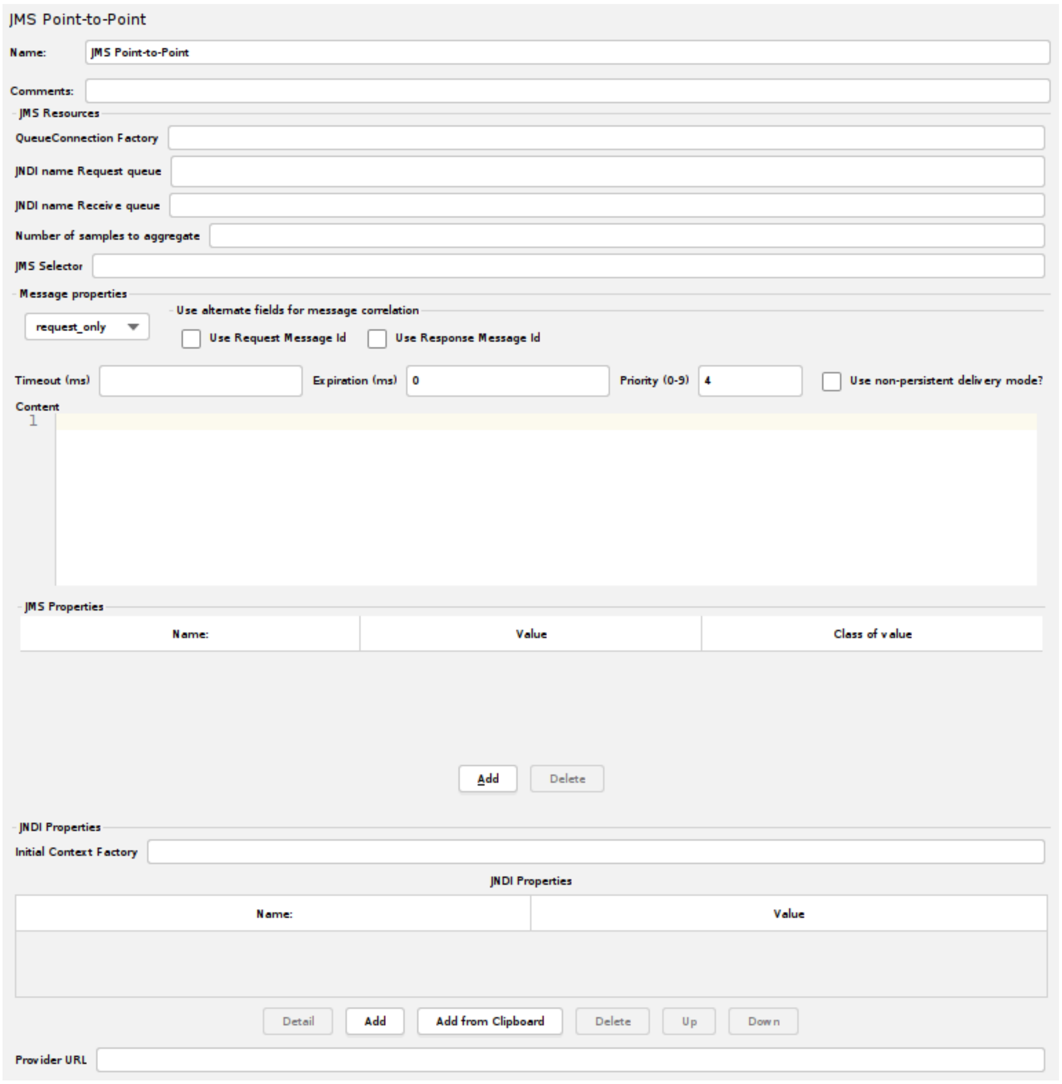
Installation of MOM libraries

The first thing to do is to install the JMS client implementation of the MOM provider (usually a .jar file) in the *JMETER_HOME/lib* directory.

For example, for Apache ActiveMQ Classic, take the file *activemq-all-X.X.X.jar* (*X.X.X* depending on the version).

JMS Point-to-Point

To simulate point-to-point communication, JMeter offers the **JMS Point-to-Point** element.


JMS Point-to-Point

Name: JMS Point-to-Point

Comments:

JMS Resources

QueueConnection Factory

JNDI name Request queue

JNDI name Receive queue

Number of samples to aggregate

JMS Selector

Message properties

request_only

Use alternate fields for message correlation

Use Request Message Id

Use Response Message Id

Timeout (ms)

Expiration (ms) 0

Priority (0-9) 4

Use non-persistent delivery mode?

Content

JMS Properties

Name:	Value	Class of value

JNDI Properties

Initial Context Factory

JNDI Properties

Name:	Value

Provider URL

JMS Point-to-Point Request

Let's look in more detail at some of the parameters.

JMS Resources and JNDI Properties

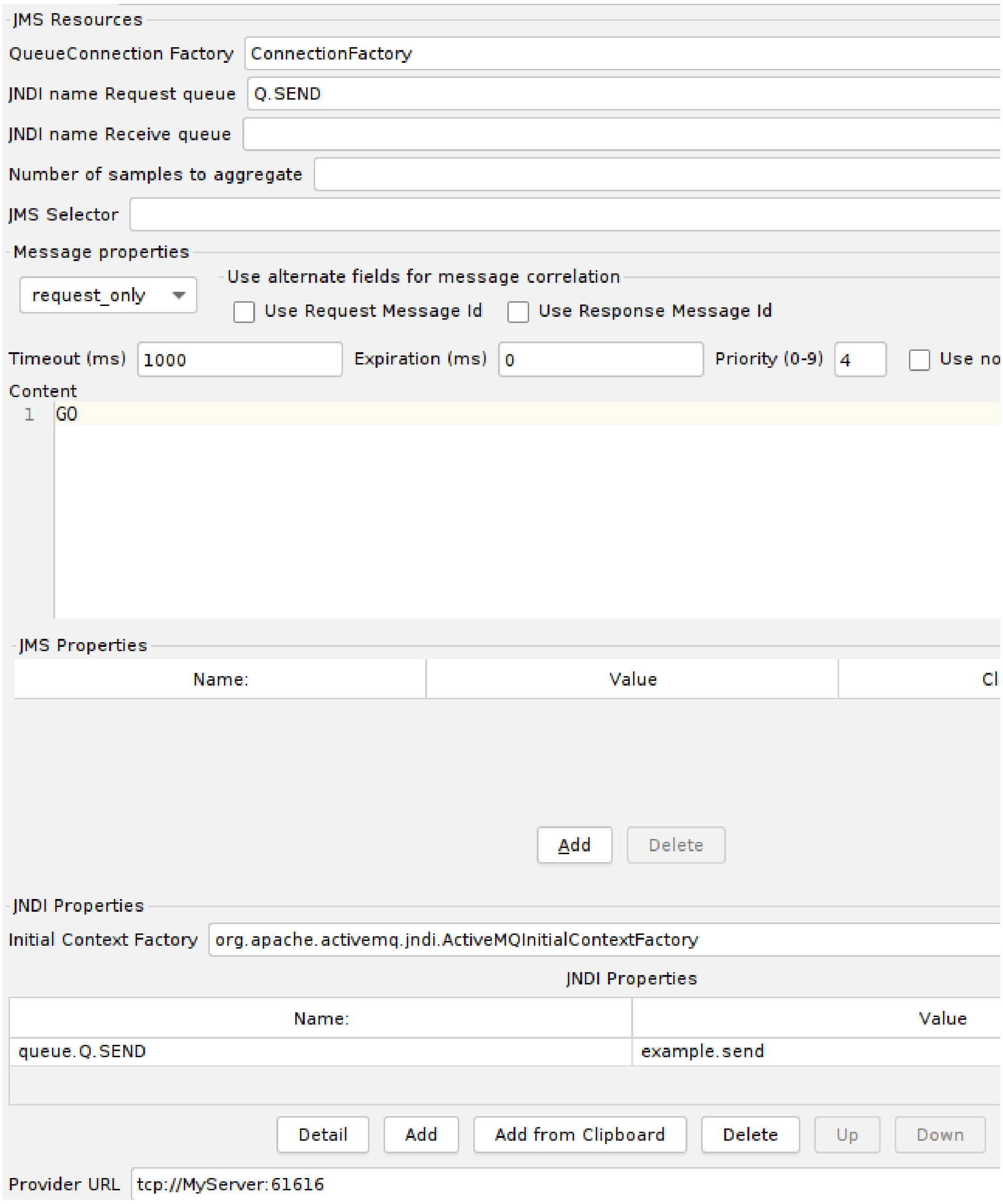
Let's start with the *JMS Resources* and *JNDI Properties* settings.

JNDI is a Java programming interface (API) for Java object naming inside the Java virtual machine, so in summary it is a directory.

In the case of JMS, JNDI is used to retrieve the instances of the destination objects (*Topic* or *Queue*) and *Connection Factory* (*Connection Factory* which allows you to create a connection to the JMS provider).

In order to fill the fields of these two parts, it will be necessary to look at the configuration of the MOM and/or to ask the architect or the developers.
If we want to use automatically created temporary queues, we will have to follow a naming convention for the values â€‹â€‹of *JMS Properties* and *JNDI Properties*.

For example, if the value of the *JNDI Name Request queue* field is *Q.SEND*, then a field named *queue.Q.SEND* will be required in *JNDI Properties*.


JMS Resources

QueueConnection Factory: ConnectionFactory

JNDI name Request queue: Q.SEND

JNDI name Receive queue:

Number of samples to aggregate:

JMS Selector:

Message properties

request_only

Use alternate fields for message correlation

☐ Use Request Message Id ☐ Use Response Message Id

Timeout (ms): 1000 Expiration (ms): 0 Priority (0-9): 4 ☐ Use no

Content

```
GO
```

JMS Properties

Name:	Value	Cl

JNDI Properties

Initial Context Factory: org.apache.activemq.jndi.ActiveMQInitialContextFactory

JNDI Properties

Name:	Value
queue.Q.SEND	example.send

Provider URL: tcp://MyServer:61616

JNDI name of the Request queue

For Apache ActiveMQ Classic[15]:

- *QueueConnection Factory* will be *ConnectionFactory*
- *Initial Context Factory* (the class used to create a connection) will be *org.apache.activemq.jn*
- *Provider URL* (the address and access port for ActiveMQ) will be *tcp://MyServer:61616* for the default configuration.

For Apache ActiveMQ Artemis[16]:

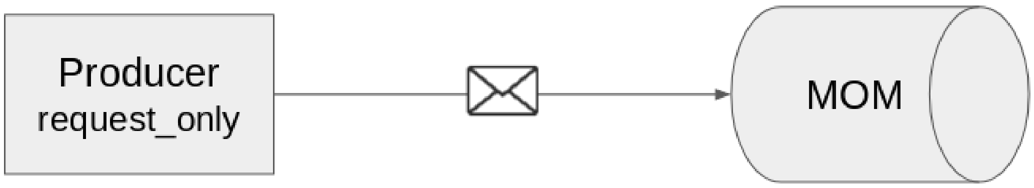
- *QueueConnection Factory* will be *ConnectionFactory*
- *Initial Context Factory* (the class used to create a connection) will be *org.apache.activemq.ar*
- *Provider URL* (the address and access port for Artemis ActiveMQ) will be *tcp://MyServer:61616* for the default configuration.

Communication style

Another important field is *Communication style* in the *Message properties* section.

It can have several values, but let's look at the most useful ones.

request_only, as its name indicates, corresponds to the sending of a request.



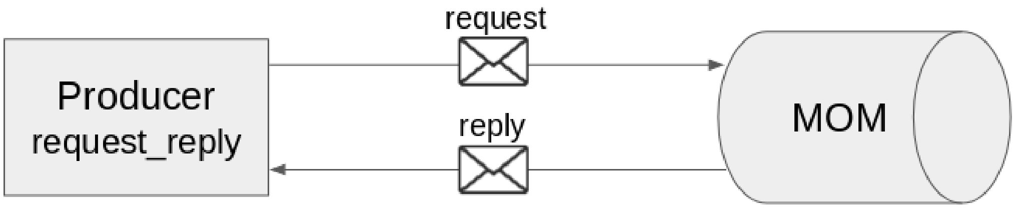
Communication type: Request only

Be aware that in this configuration, messages present in the *Queue* are not consumed, so remember to purge and/or consume them.

The second possible value is *request_reply*. Here we send (*request*) and we wait for the answer (*reply*) synchronously.

[15]http://activemq.apache.org/
[16]https://activemq.apache.org/artemis/index.html

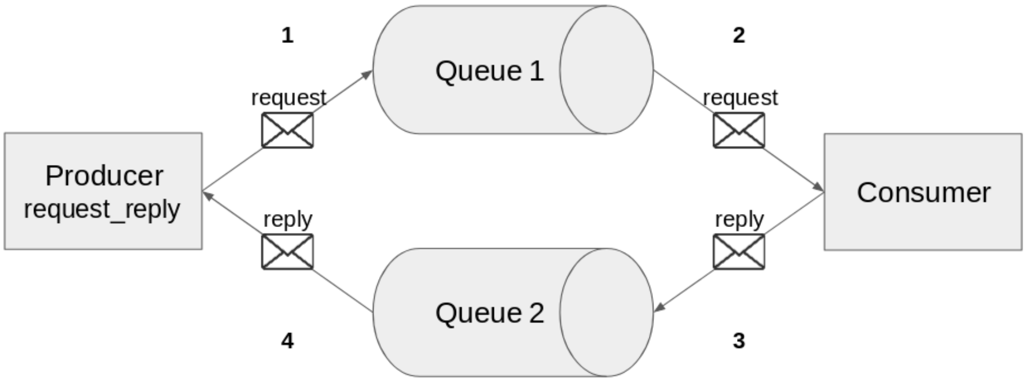
If the value of the *JNDI name Request queue* field and the *JNDI name Receive queue* field are the same, JMeter will take care of the two steps (sending and retrieving the response).



Communication type: Request reply

However, if the two queues have different values, the message sent must be consumed to receive the ACK.

If the message is not consumed, we will have an error after the delay defined in *Timeout (ms).*



Communication steps for: Request reply

1: JMeter is the producer of the message in Queue 1

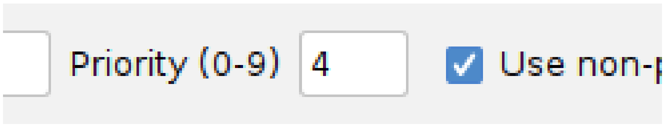
2: The consumer listening on Queue 1 receives it

3: The consumer sends the answer to Queue 2

4: JMeter which listens to the answer in Queue 2 receives it

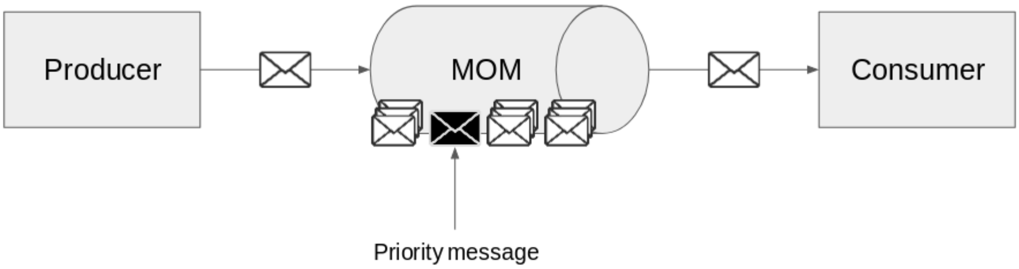
Priority

We can manage the priority of a message using the *Priority* parameter.


Priority (0-9) 4 Use non-

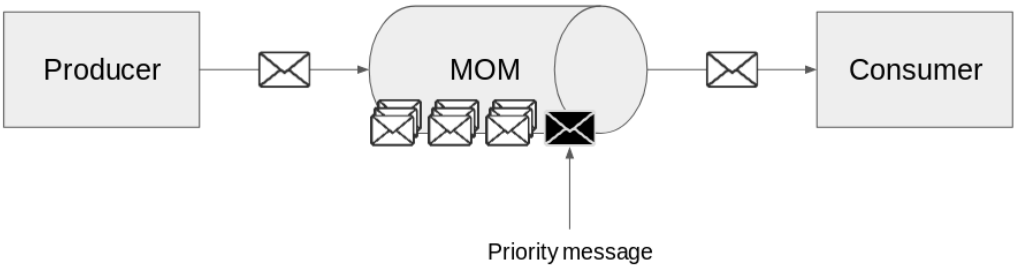
Choosing the priority of a message

Without priority set, the behavior will be as follows.



All messages are sent with the same priority

If we give a higher priority to our message, it will be processed before others.



All messages are sent with different priorities

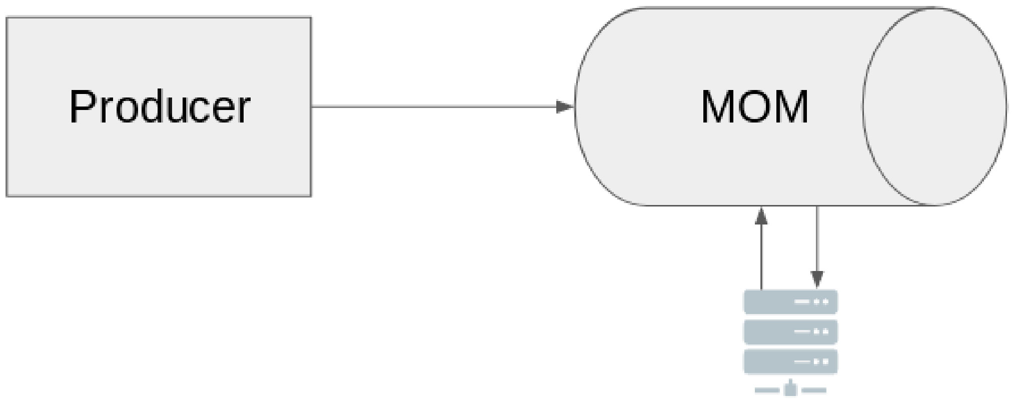
Use non-persistent delivery mode?

The parameter *Use non-persistent delivery mode?* allows us to avoid persisting the message on disk/database/etc.

Persisting a message ensures that the MOM will not lose it if a crash happens.

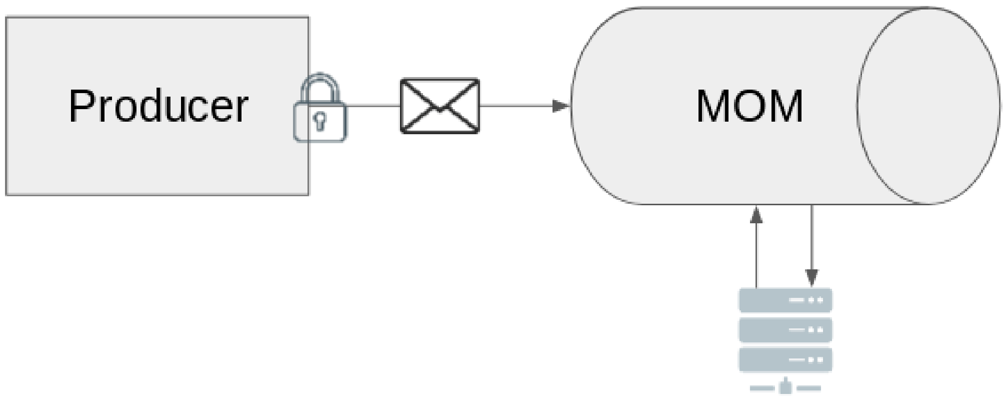
The persistence mechanism is as follows.

The MOM is configured to persist messages on the disk.



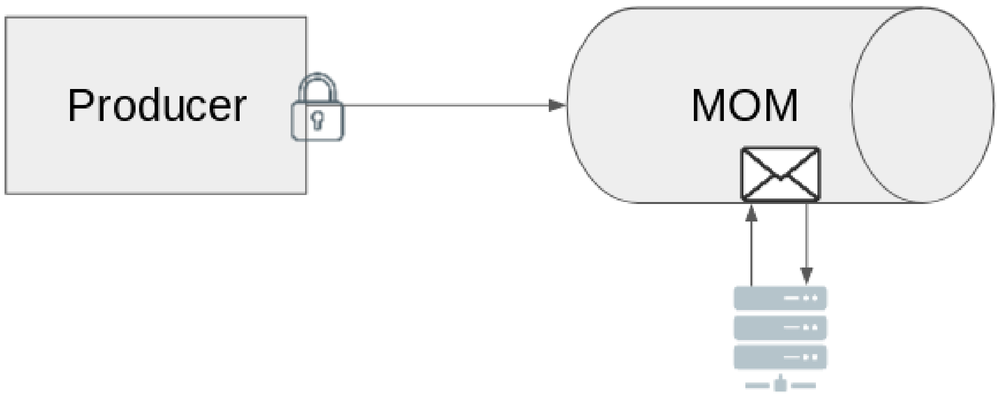
Persistence of messages: beginning

When sending a message, a lock is put on the producer side.



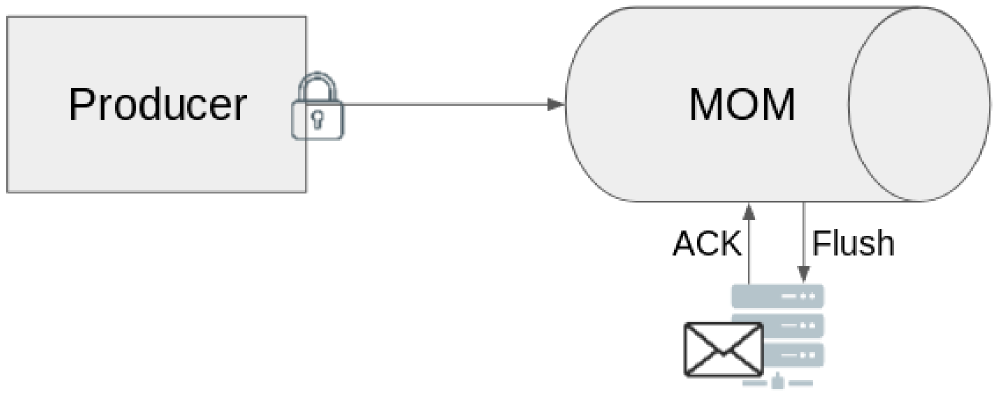
Persistence of messages: setting the lock when sending

The MOM receives the message and prepares to persist.



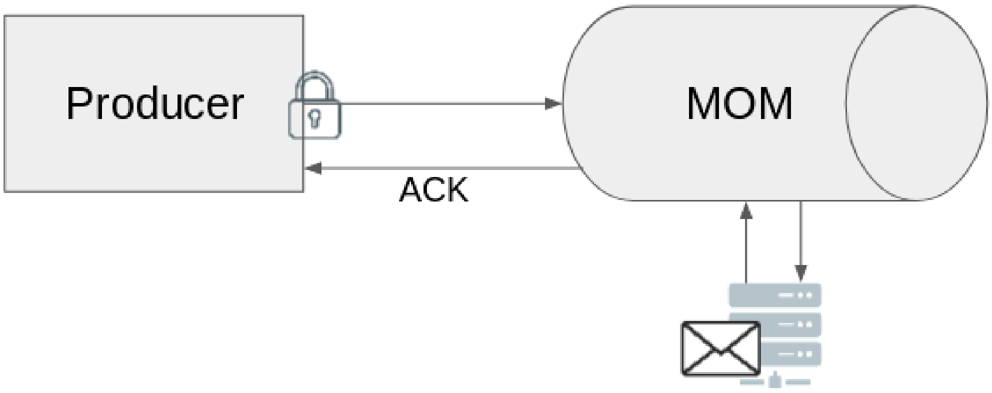
Persistence of messages: reception of the message

The MOM writes the file to disk in a safe way (it waits for an acknowledgment).



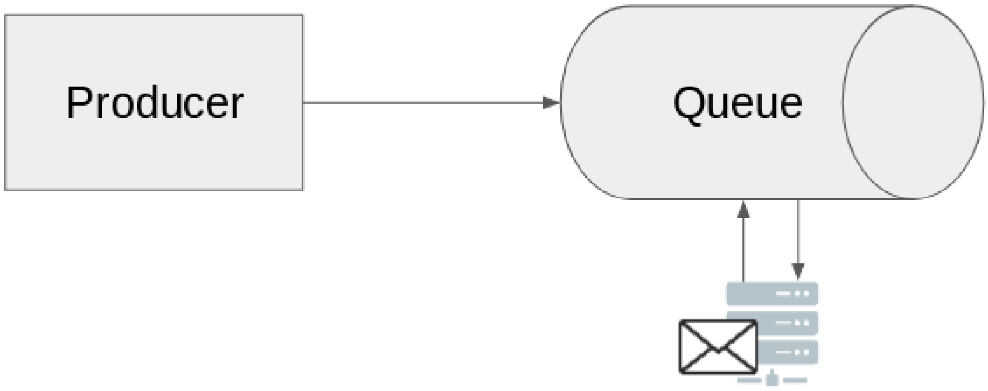
Persistence of messages: write to disk

The MOM sends an acknowledgment to the producer.



Persistence of messages: acknowledgment of receipt

The lock on the producer is removed.

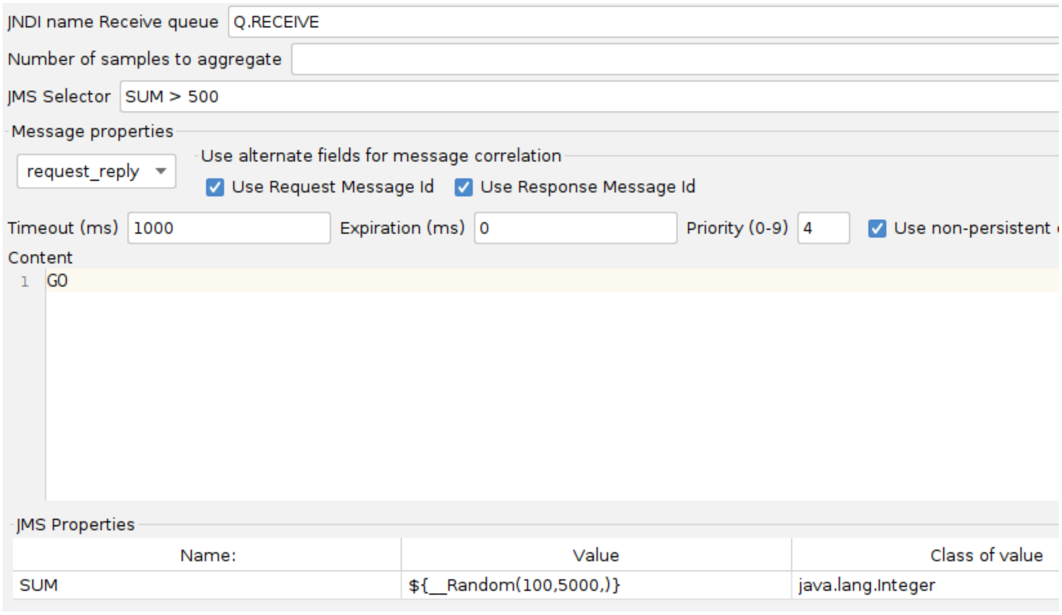


Persistence of messages: end

JMS Selector

Another parameter is the *JMS Selector* that allows a consumer to read only a certain type of message.

The consumer will only process messages that match the selector expression and ignore others.



JNDI name Receive queue	Q.RECEIVE
Number of samples to aggregate	
JMS Selector	SUM > 500

Message properties

request_reply

Use alternate fields for message correlation

✓ Use Request Message Id ✓ Use Response Message Id

Timeout (ms) 1000 Expiration (ms) 0 Priority (0-9) 4 ✓ Use non-persistent

Content

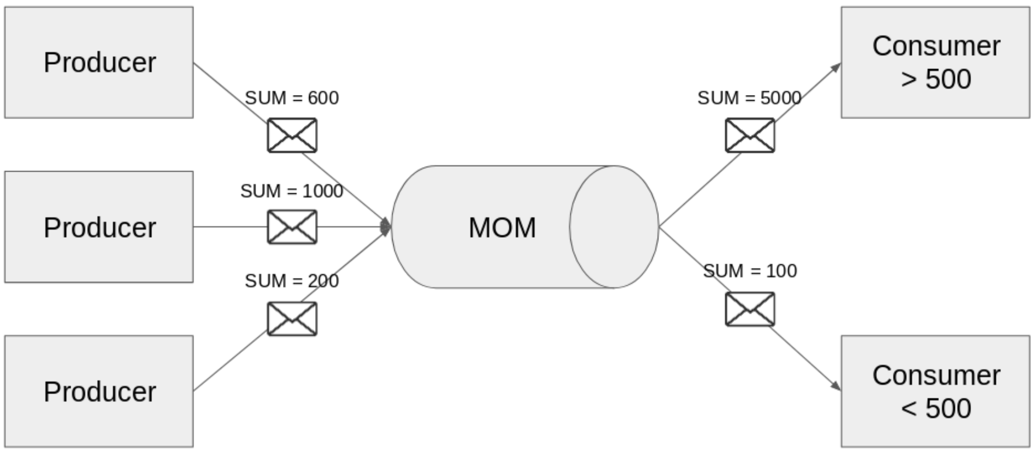
```
GO
```

JMS Properties

Name:	Value	Class of value
SUM	${__Random(100,5000,)}	java.lang.Integer

JMS Selector

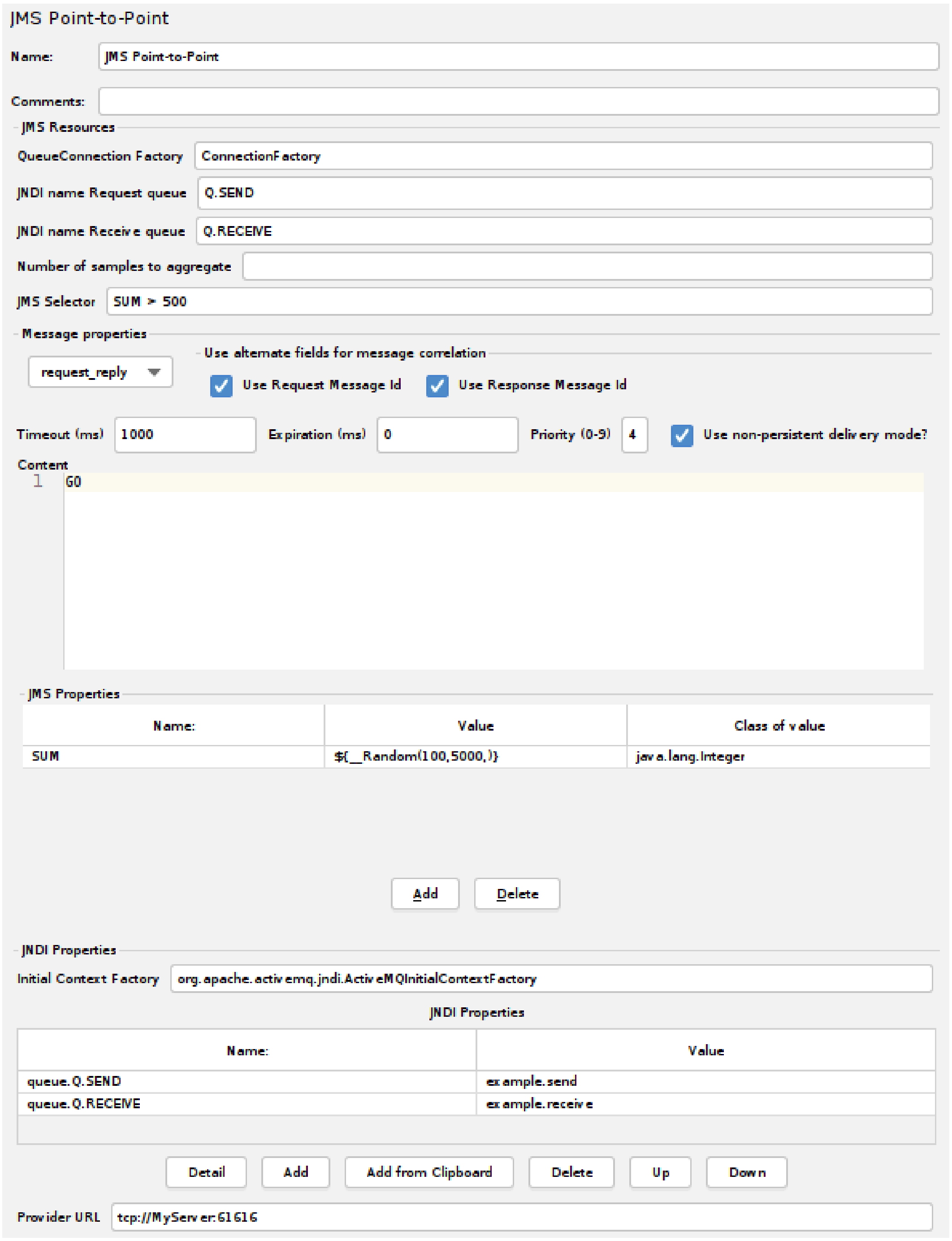
Which will give us the following architecture with a property **SUM**.



JMS Selector: different processing based on SUM property's value

Example with Apache ActiveMQ Classic

With a default installation of Apache ActiveMQ Classic, we end up with:



JMS Point-to-Point

Name:	JMS Point-to-Point
Comments:	

JMS Resources

QueueConnection Factory	ConnectionFactory
JNDI name Request queue	Q.SEND
JNDI name Receive queue	Q.RECEIVE
Number of samples to aggregate	
JMS Selector	SUM > 500

Message properties

request_reply

Use alternate fields for message correlation

✓ Use Request Message Id ✓ Use Response Message Id

Timeout (ms) 1000 Expiration (ms) 0 Priority (0-9) 4 ✓ Use non-persistent delivery mode?

Content

```
GO
```

JMS Properties

Name:	Value	Class of value
SUM	${__Random(100,5000,)}	java.lang.Integer

JNDI Properties

Initial Context Factory	org.apache.activemq.jndi.ActiveMQInitialContextFactory

JNDI Properties

Name:	Value
queue.Q.SEND	example.send
queue.Q.RECEIVE	example.receive

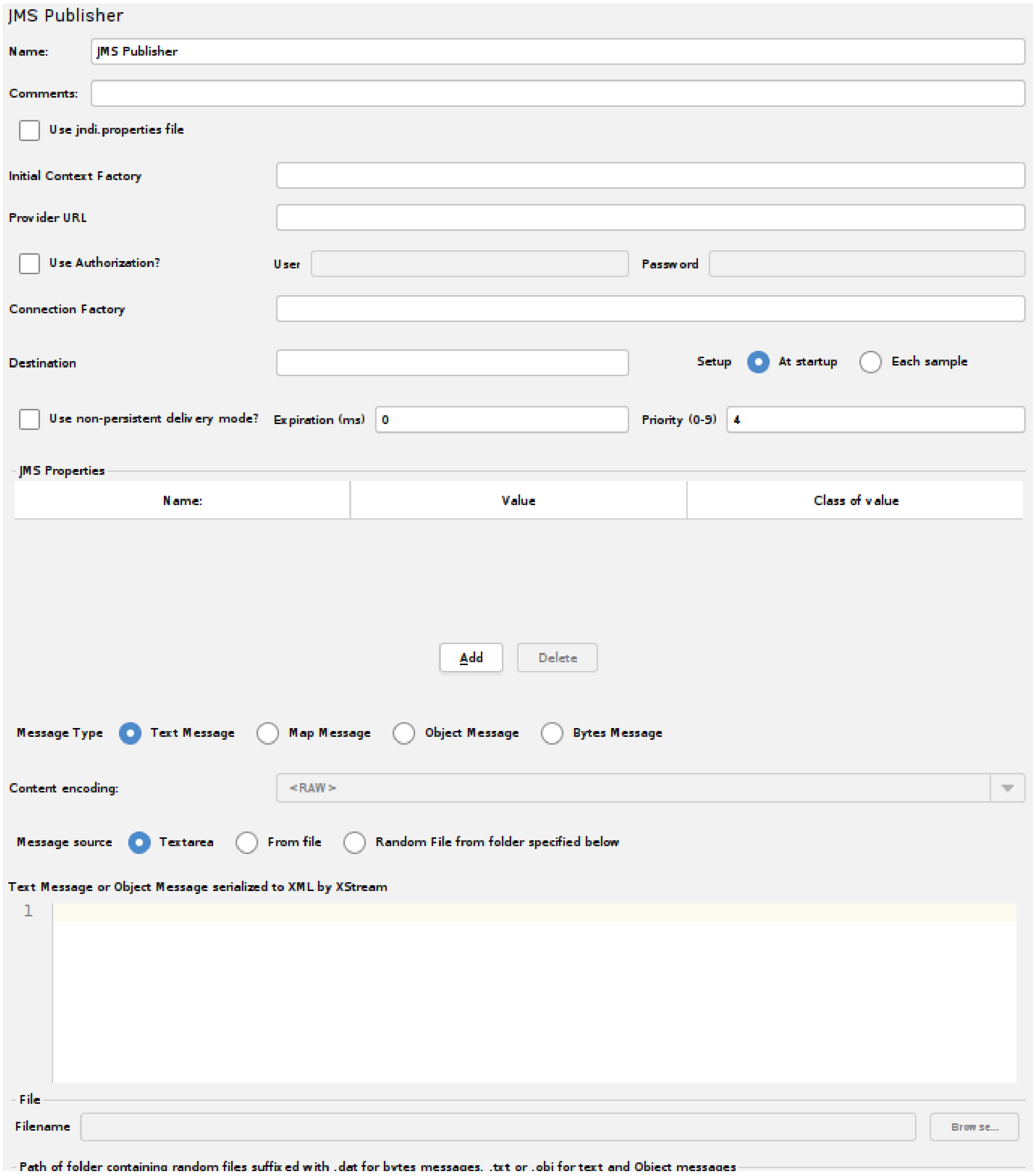
Provider URL	tcp://MyServer:61616

Point-to-Point JMS Request for ActiveMQ Classic

Publish/Subscribe

Finally, *Publish/Subscribe* communication is implemented with **JMS Publisher** and **JMS Subscriber**.

JMS Publisher


JMS Publisher

Name: JMS Publisher

Comments:

[ ] Use jndi.properties file

Initial Context Factory

Provider URL

[ ] Use Authorization? User Password

Connection Factory

Destination

Setup [✓ At startup] Each sample

[ ] Use non-persistent delivery mode? Expiration (ms) 0 Priority (0-9) 4

JMS Properties

Name:	Value	Class of value

Add Delete

Message Type [✓ Text Message] Map Message Object Message Bytes Message

Content encoding: <RAW>

Message source [✓ Textarea] From file Random File from folder specified below

Text Message or Object Message serialized to XML by XStream

File

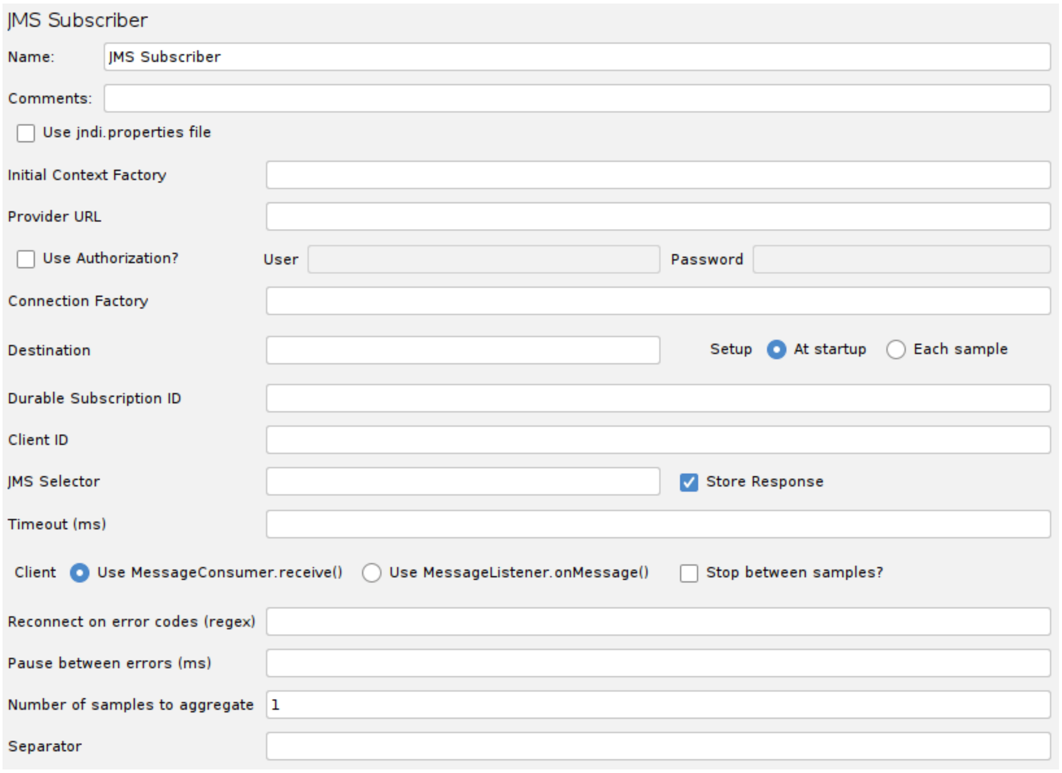
Filename Browse...

Path of folder containing random files suffixed with .dat for bytes messages, .txt or .obj for text and Object messages

JMS Publication Request

As we can see, most parameters are the same as before.

JMS Subscriber



JMS Subscriber

Name:	JMS Subscriber
Comments:	
[ ] Use jndi.properties file	
Initial Context Factory	
Provider URL	
[ ] Use Authorization?	User / Password
Connection Factory	
Destination	Setup [✓ At startup] Each sample
Durable Subscription ID	
Client ID	
JMS Selector	[✓ Store Response]
Timeout (ms)	
Client	[✓ Use MessageConsumer.receive()] Use MessageListener.onMessage() [ ] Stop between samples?
Reconnect on error codes (regex)	
Pause between errors (ms)	
Number of samples to aggregate	1
Separator	

Subscription JMS Request

Two new parameters appear:

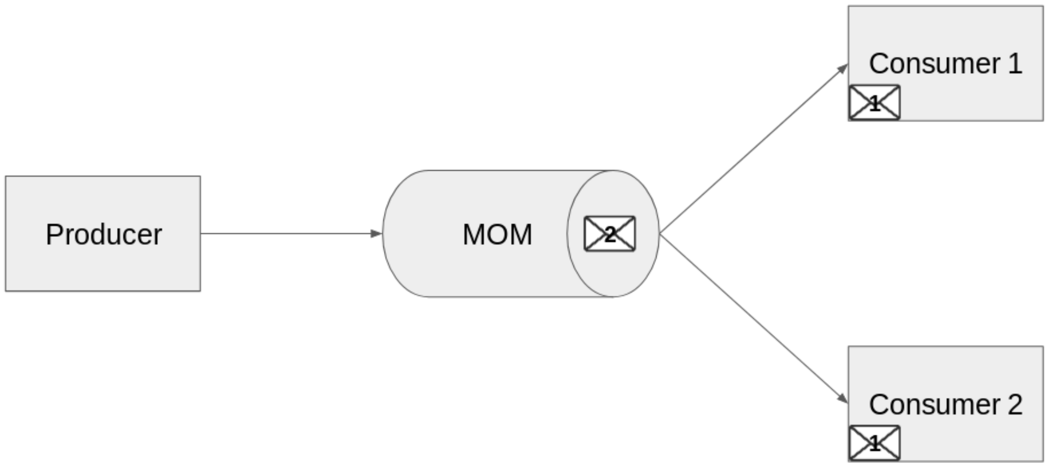
- *Durable Subscription ID*
- *Client ID*

They allow the use of a *Durable Subscriber*.
A *Durable Subscriber* is a subscriber that has established a durable subscription. In such configuration, messages published while the subscriber is not connected will be redistributed whenever it reconnects.

Let's see what is a Durable Subscriber.

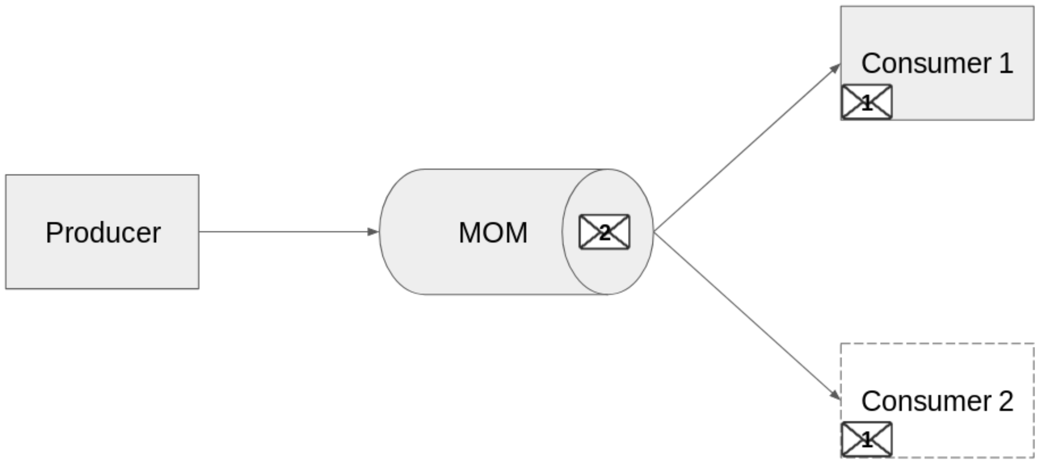
Without *Durable Subscriber*:

Both consumers are active and read the message 1.



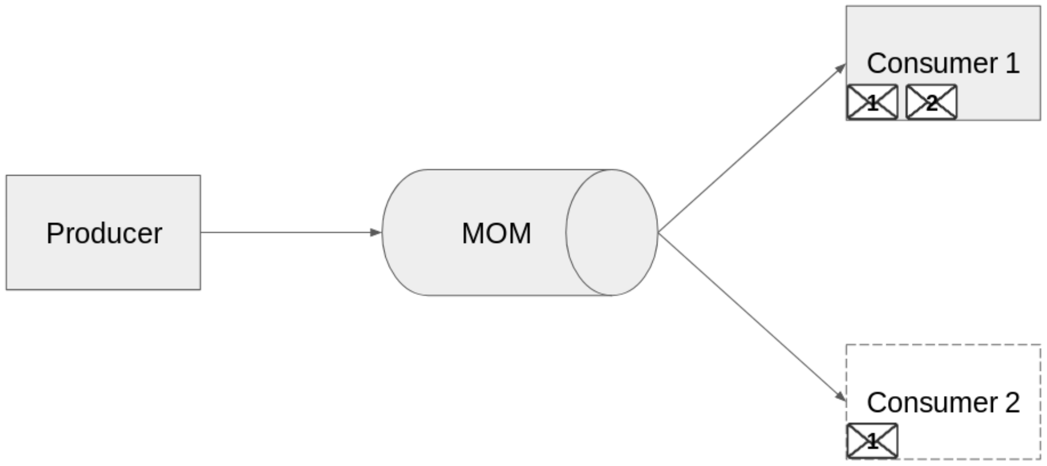
Without Durable Subscriber: 2 active consumers

The consumer 2 becomes inactive before reading the message 2.



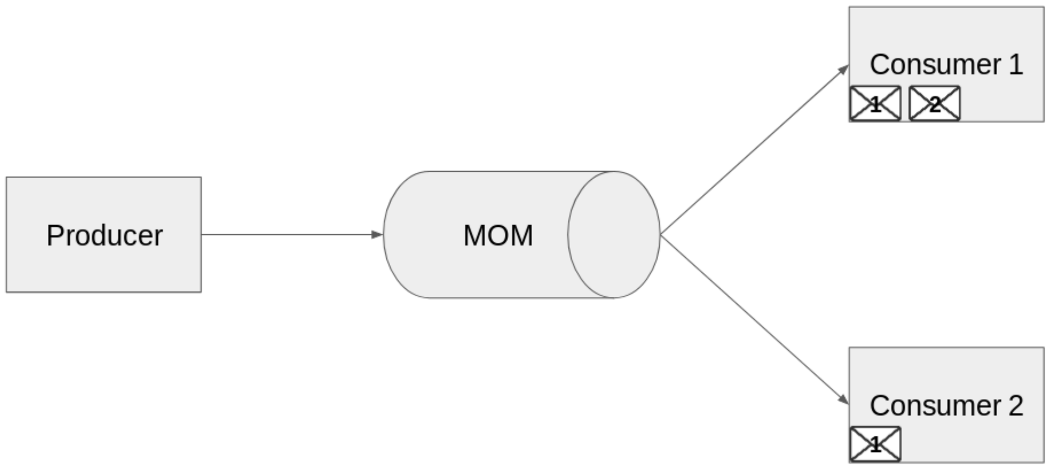
Without Durable Subscriber: consumer 2 inactive

The consumer 1 reads the message 2.



Without Durable Subscriber: message 2 consumed by the consumer 1

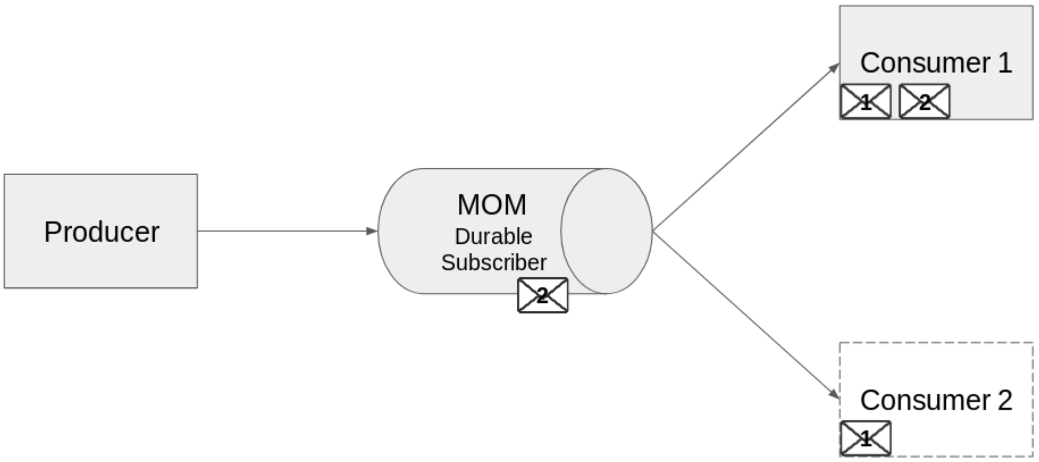
The consumer 2 is active again, unfortunately the message 2 is no longer present in the MOM. It will never be processed by consumer 2, since it is a nondurable receiver, it can only retrieve messages published after its subscription and must remain active to receive them.



Without Durable Subscriber: message 2 is active again

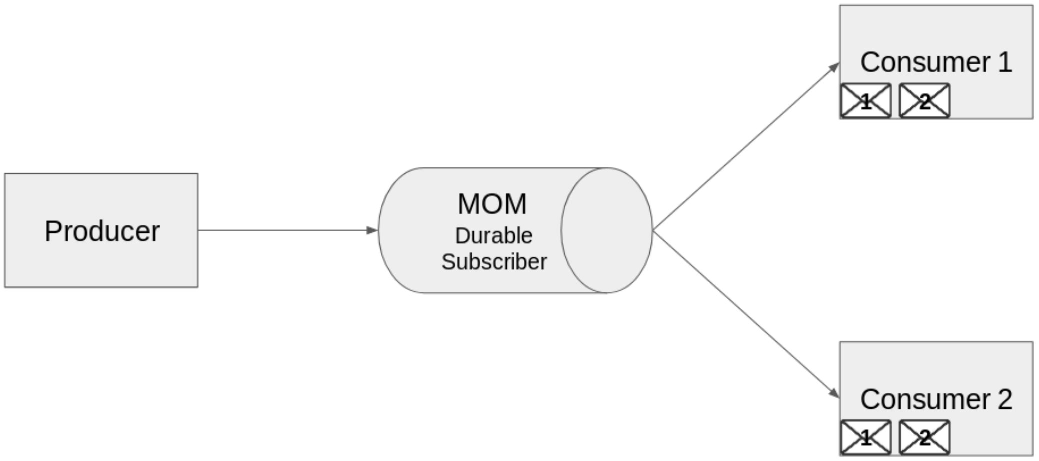
With *Durable Subscriber*, the message 2 would have been processed by the consumer 2. Indeed, in this case, the receiver receives all the messages sent while it was down, this happens as soon as it reconnects.

The MOM will take care of storing message 2 until all *Durable Subscriber* have processed it.



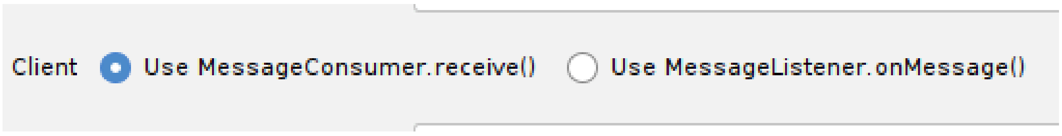
With Durable Subscriber: message 2 is stored

When the consumer 2 is active again, he processes the message 2. Then the message 2 is removed on the MOM.



With Durable Subscriber: message 2 is processed

Note that we can choose how the message will be consumed.


Client [✓ Use MessageConsumer.receive()] Use MessageListener.onMessage()

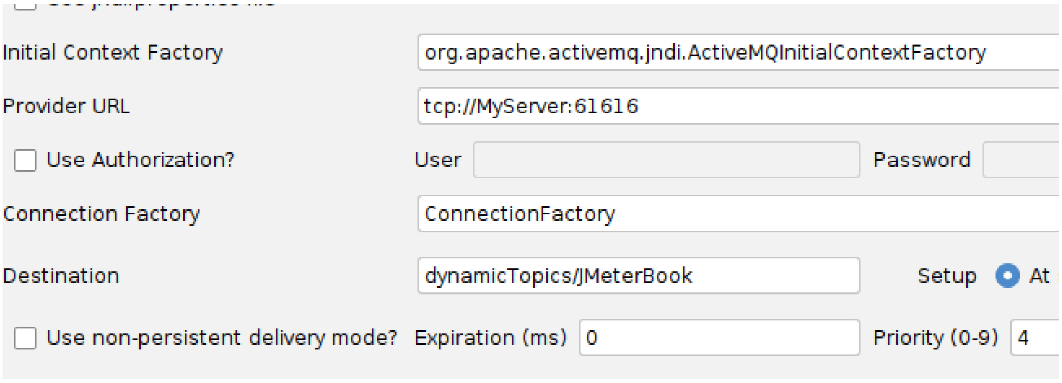
How the message will be consumed for Subscription JMS Request

The choice is between *MessageConsumer.receive()* and *MessageListener.onMessage()*:

- With the first choice (*MessageConsumer.receive()*), the message is consumed synchronously by the application (the method *receive()* blocks until a message is consumed or the waiting time is exceeded (see parameters *Timeout(ms)*).
- With the second choice, the consumption of the message by the application is asynchronous (an object of type *MessageListener* is registered and its method *onMessage()* will be called on each message reception).

Let's go back to our default installation of Apache ActiveMQ Classic.

We will use the Apache ActiveMQ Classic feature that allows us to dynamically create topics. This will consist of naming our destination with prefix *dynamicTopic/*


Initial Context Factory: org.apache.activemq.jndi.ActiveMQInitialContextFactory
Provider URL: tcp://MyServer:61616
☐ Use Authorization? User: Password:
Connection Factory: ConnectionFactory
Destination: dynamicTopics/JMeterBook Setup [✓ At]
☐ Use non-persistent delivery mode? Expiration (ms): 0 Priority (0-9): 4

JMS Query Posting and dynamically creating topics

Methodology

As with all tests, testing a MOM and/or servers should be as realistic as possible.

Before to start the load test we need:

- A Workload Model
- Check if the API calls a third party API (pay-per-use or not)
- A working environment for your test (infrastructure and data)
- How MOM is used (pattern, communication)

For example, if the messages exchanged are persistent, the same thing must be done with JMeter.

Likewise with the other parameters (delay, authentication, etc.).

My advice is to discuss with developers and/or architects to configure the test to be as representative as possible.

Some tips to design Workload Model:

- If you can, use the production monitoring to analyse real-world usage patterns
- List of all the consumers/producers (include background processes like batch jobs)
- Take account of retry mechanisms (which can increase the number of requests with a "retry storm" where multiple users hit a failure point and initiate retries simultaneously)
- Include all types of messages (deprecated messages too) if applicable
- Simulate user security processes (token…), but not necessarily at each iteration

Some tips to design test data:

- Test with different user roles and permission levels
- Test with different request size

See chapter *"Designing a test case"* for more advice.

Some tips to monitor/analyse load test:

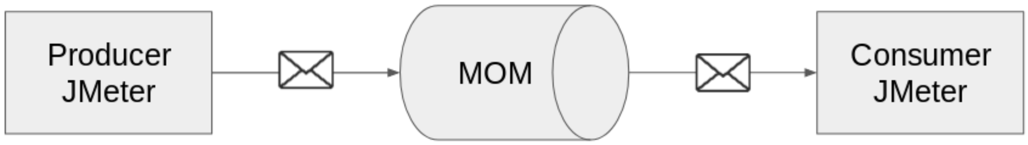
- Monitor the volume of data transferred can be very important. Indeed, messages can be quite verbose, so they can be the cause of performance problems
- Monitor the activation of design for failure patterns like *retry*
- Monitor *DLQ*
- Be careful with SLAs, MOM allow asynchronous message and in JMeter in some case (if we simulate the producer we will measure the response time of *ack* and not necessarily the response time of the process)

See chapter *“Visualising, analysing and reporting the load testing results”* for more advice.

Let’s take a look at what we need to test (non-exhaustive list) :

- The configuration of the MOM

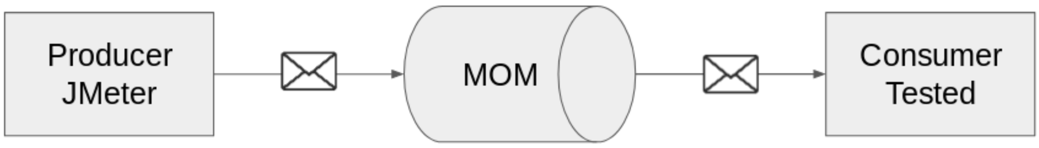
For that, JMeter must play the role of producer and consumer.



Test the configuration of the MOM

- The performances of our consumer

Here we will only use JMeter as producer.



Test the performance of the consumer

- The resilience of our MOM configuration

In this case, JMeter plays the role of producer, consumer and injects errors (*Failure injection*).

- End to end performance (producer - queue - consumer)

Here we will send requests to producer and monitor all the part.
To do this test, go to chapter *"Load Testing web services"*.

Let's move on to practice.

Put into practice with JMeter

To simplify the examples, we will not implement all the good practices of the chapters *"Preparing the test environment (injectors and tested system)"* and *"Design a test case"*.
In particular, the tests will be performed locally (JMeter and MOM), avoid this in real life!

The examples in this book are not necessarily realistic, but they allow you to learn lots of recipes that you can use in your projects.

Example 1: Test the configuration of a MOM server with point-to-point

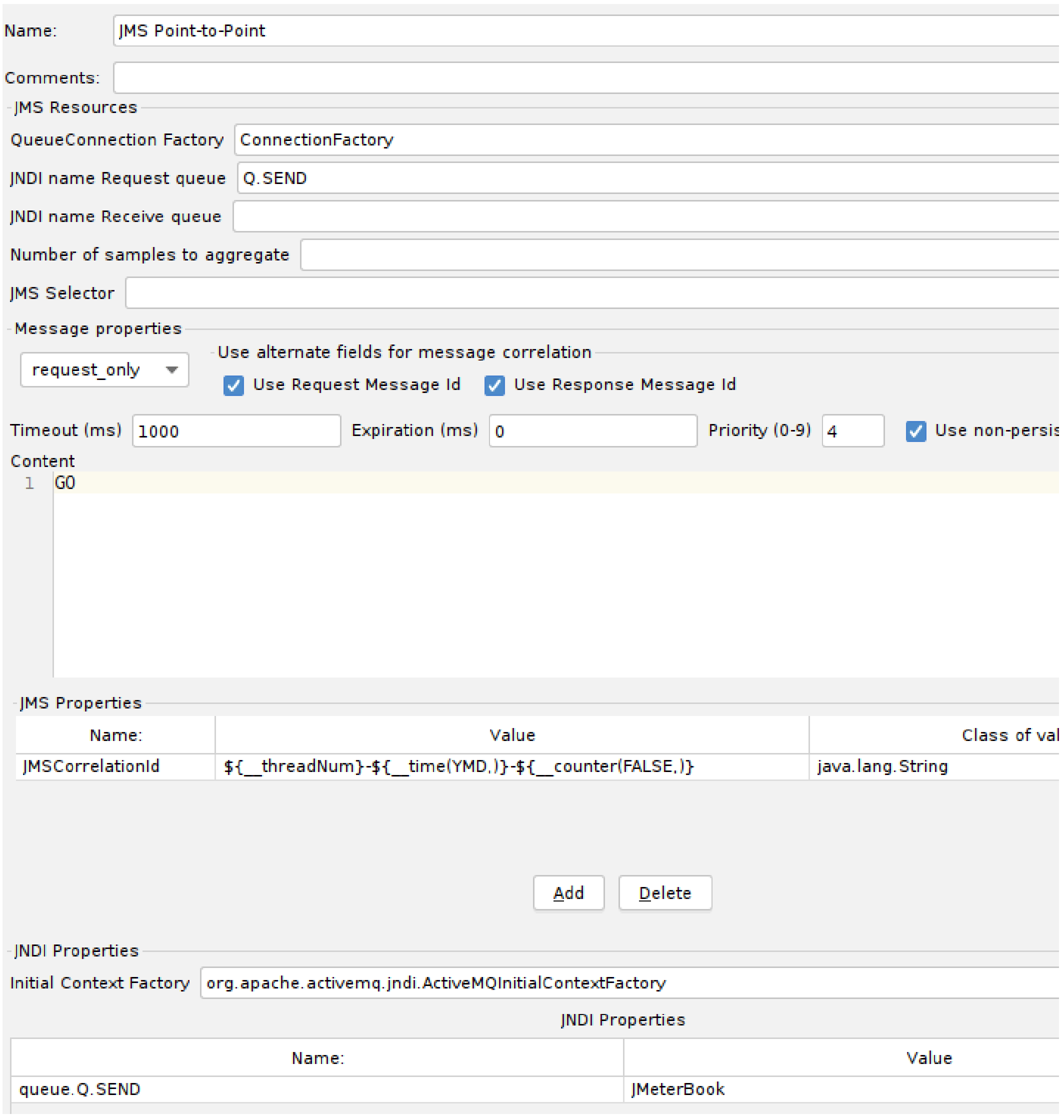
Our first test will be to test a MOM server with point-to-point messaging.

We will use ActiveMQ Classic as MOM.

Before starting with our JMeter test plan, let's create a queue named *JMeterBook*.

At first we will send a message in a queue.

In JMeter we will configure **JMS Point-to-Point** as follows.



Name:	JMS Point-to-Point
Comments:	

JMS Resources

QueueConnection Factory	ConnectionFactory
JNDI name Request queue	Q.SEND
JNDI name Receive queue	
Number of samples to aggregate	
JMS Selector	

Message properties

request_only

Use alternate fields for message correlation

✓ Use Request Message Id ✓ Use Response Message Id

Timeout (ms) 1000 Expiration (ms) 0 Priority (0-9) 4 ✓ Use non-persis

Content

```
GO
```

JMS Properties

Name:	Value	Class of val
JMSCorrelationId	${__threadNum}-${__time(YMD,)}-${__counter(FALSE,)}	java.lang.String

JNDI Properties

Initial Context Factory org.apache.activemq.jndi.ActiveMQInitialContextFactory

JNDI Properties

Name:	Value
queue.Q.SEND	JMeterBook

Sending a message in ActiveMQ Classic

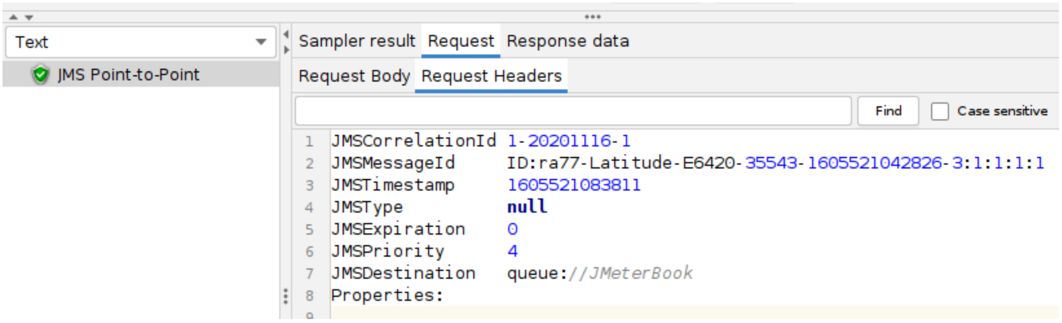
In this example, we generate a unique *JMSCorrelationId* identifier using the following JMeter functions:

- *${__time(YMD,)}*: return the current date
- *${__threadNum}*: return the number of the current thread

- *${__counter(FALSE,)}*: return a unique number for each call

Validate our script by sending a message.

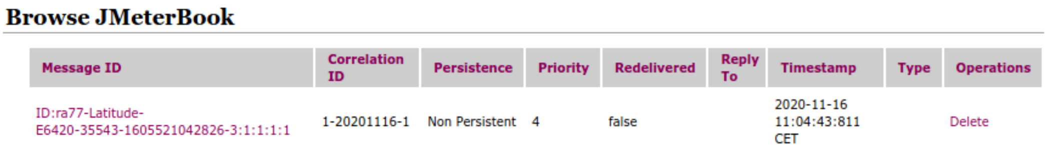
In JMeter we can see our message using a **View Results Tree**.



```
JMSCorrelationId 1-20201116-1
JMSMessageId     ID:ra77-Latitude-E6420-35543-1605521042826-3:1:1:1:1
JMSTimestamp     1605521083811
JMSType          null
JMSExpiration    0
JMSPriority      4
JMSDestination   queue://JMeterBook
Properties:
```

Result of sending in JMeter

In the ActiveMQ Classic Administration Console we can see that the message has arrived, but not yet consumed.



Browse JMeterBook

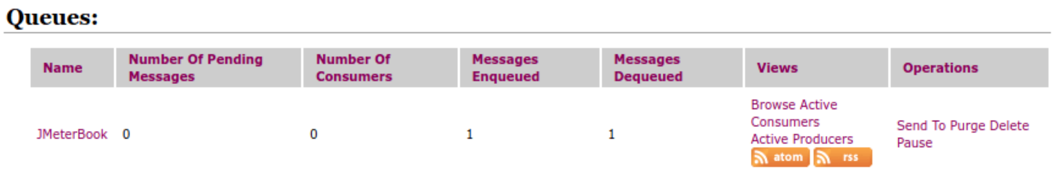
Message ID	Correlation ID	Persistence	Priority	Redelivered	Reply To	Timestamp	Type	Operations
ID:ra77-Latitude-E6420-35543-1605521042826-3:1:1:1:1	1-20201116-1	Non Persistent	4	false		2020-11-16 11:04:43:811 CET		Delete

Result of sending in ActiveMQ Classic console

Now we want to consume the messages sent. For this we will configure **JMS Point-to-Point** to read the message.

Let's change the value of the parameter:

- *Communication style* in *request_reply*
- *JNDI name Receive queue* value equal to *JNDI name Request queue* value



Queues:

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
JMeterBook	0	0	1	1	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause

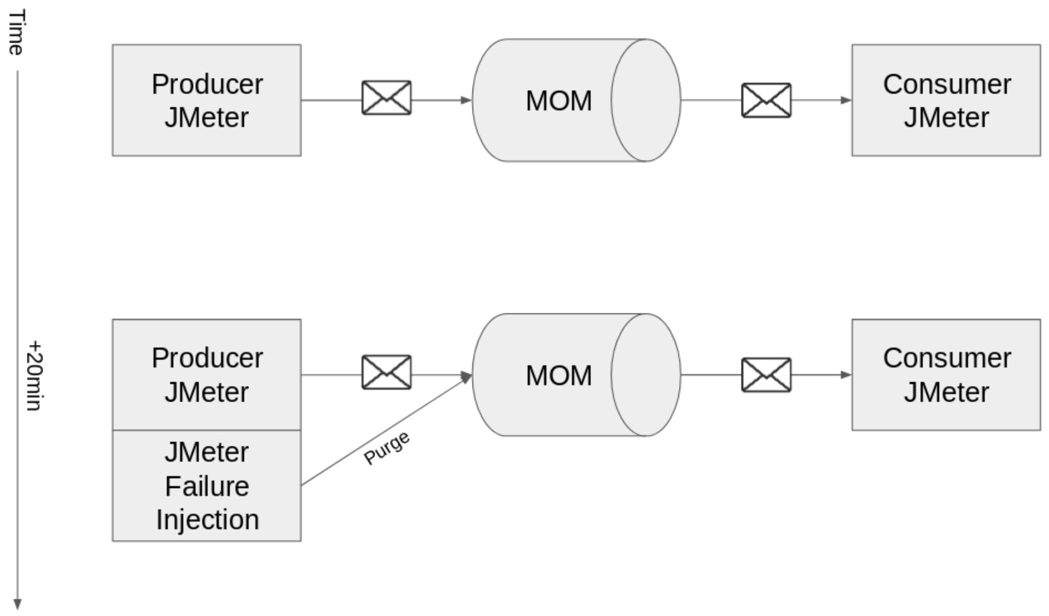
Result of sending/receiving in ActiveMQ Classic console

Now that our script is ready, we only have to test all the parameters and their combination in order to find the most appropriate ones for our project.

At this point, the configuration of the server (memory, processor, etc.) and *ActiveMQ Classic* (size of JMS queues, etc.) is validated.

We are now required to test the following scenario: "*What happens if the queue is purged while serving application?*"

Our test protocol will be the following.



Test protocol

We can purge a queue in many ways:

- Use the Rest API

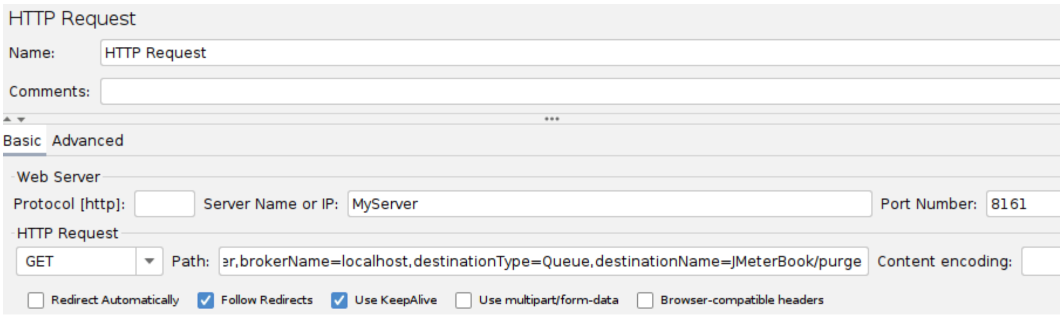
- Use JMX
- Use the CLI

In our example, we will use the first solution: the REST API.

Our call will be:

```
http://MyServer:8161/api/jolokia/exec/org.apache.activemq:type=Broker,b\
rokerName=localhost,destinationType=Queue,destinationName=JMeterBook/pu\
rge
```

And in JMeter.


HTTP Request
Name: HTTP Request
Comments:
Basic Advanced
Web Server
Protocol [http]: Server Name or IP: MyServer Port Number: 8161
HTTP Request
GET Path: ɜr,brokerName=localhost,destinationType=Queue,destinationName=JMeterBook/purge Content encoding:
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Purging a queue with JMeter

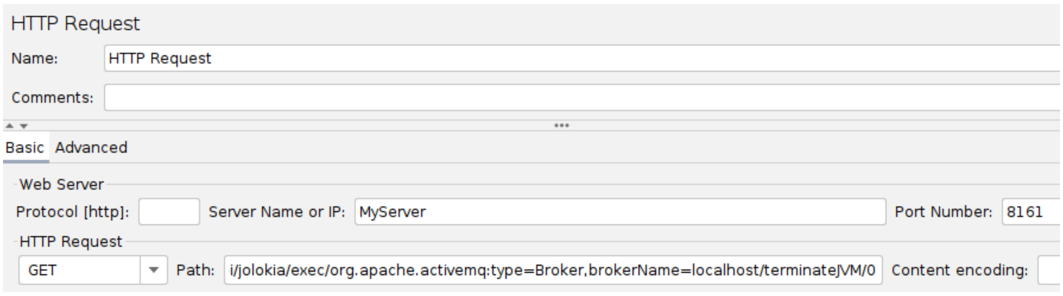
A new real-life scenario we need to check: "*With our current configuration, can we lose messages if the MOM crashes?*"

For this we will continue to use the ActiveMQ Classic Rest API.

The call to kill the JVM that runs ActiveMQ Classic will be.

```
http://MyServer:8161/api/jolokia/exec/org.apache.activemq:type=Broker,b\
rokerName=localhost/terminateJVM/0
```

In JMeter.


HTTP Request
Name: HTTP Request
Comments:
Basic Advanced
Web Server
Protocol [http]: Server Name or IP: MyServer Port Number: 8161
HTTP Request
GET Path: i/jolokia/exec/org.apache.activemq:type=Broker,brokerName=localhost/terminateJVM/0 Content encoding:

Crash the MOM using JMeter

The tests were successful.

Example 2: Test the performance of our consumer with JMS Publish/Subscribe

In this example we will use Apache ActiveMQ Artemis. To do this, install the *artemis-jms-client-all-X.X.X.jar* file from ActiveMQ Artemis into the *lib* directory of JMeter.

To reproduce this example, perform the following steps.

To create the necessary, run:

```
artemis create JMeterBookBrokerExamples --name JMeterBookBroker --user \
user --password password --allow-anonymous
```

We will add our topic JMeterBookTopic. In the configuration file of the MOM *<broker-instance>/etc/broker.xml* add the following lines in the *addresses* section.

```
<address name="JMeterBookTopic">
    <multicast>
        <queue name="JMeterBookTopic" />
    </multicast>
</address>
```

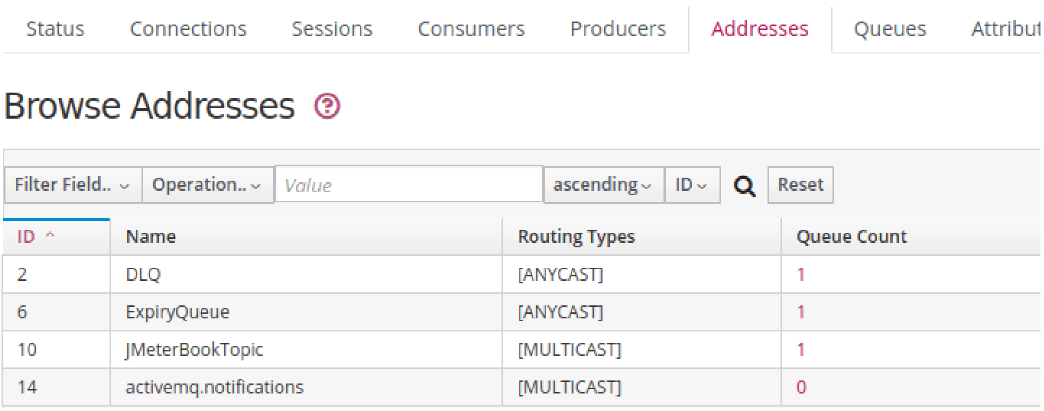
Let's run our MOM with :

```
artemis run
```

or

```
artemis-service start
```

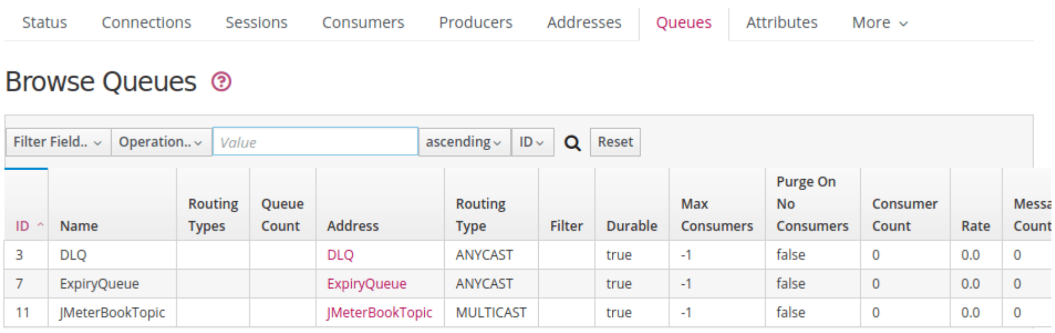
Let's make sure everything works by using the Artemis ActiveMQ console available at *http://MyServer:8161/console/*.



Browse Addresses

ID	Name	Routing Types	Queue Count
2	DLQ	[ANYCAST]	1
6	ExpiryQueue	[ANYCAST]	1
10	JMeterBookTopic	[MULTICAST]	1
14	activemq.notifications	[MULTICAST]	0

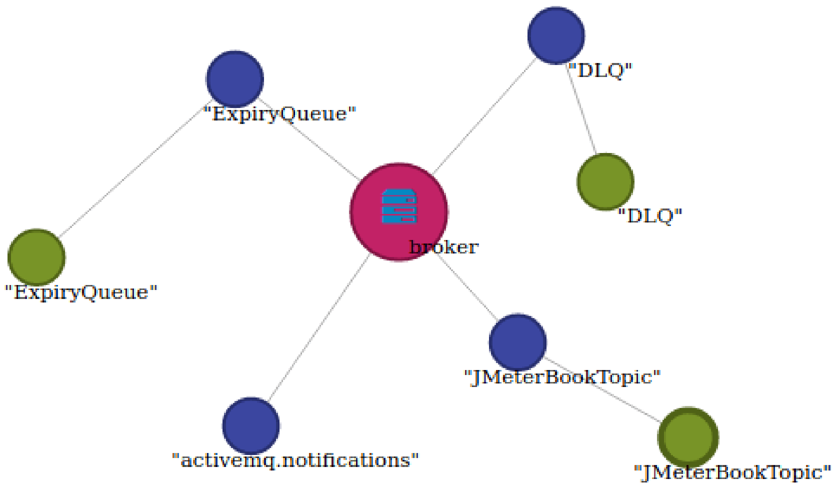
Check that Addresses have been created



Browse Queues

ID	Name	Routing Types	Queue Count	Address	Routing Type	Filter	Durable	Max Consumers	Purge On No Consumers	Consumer Count	Rate	Messa Count
3	DLQ			DLQ	ANYCAST		true	-1	false	0	0.0	0
7	ExpiryQueue			ExpiryQueue	ANYCAST		true	-1	false	0	0.0	0
11	JMeterBookTopic			JMeterBookTopic	MULTICAST		true	-1	false	0	0.0	0

Check that our topic has been created



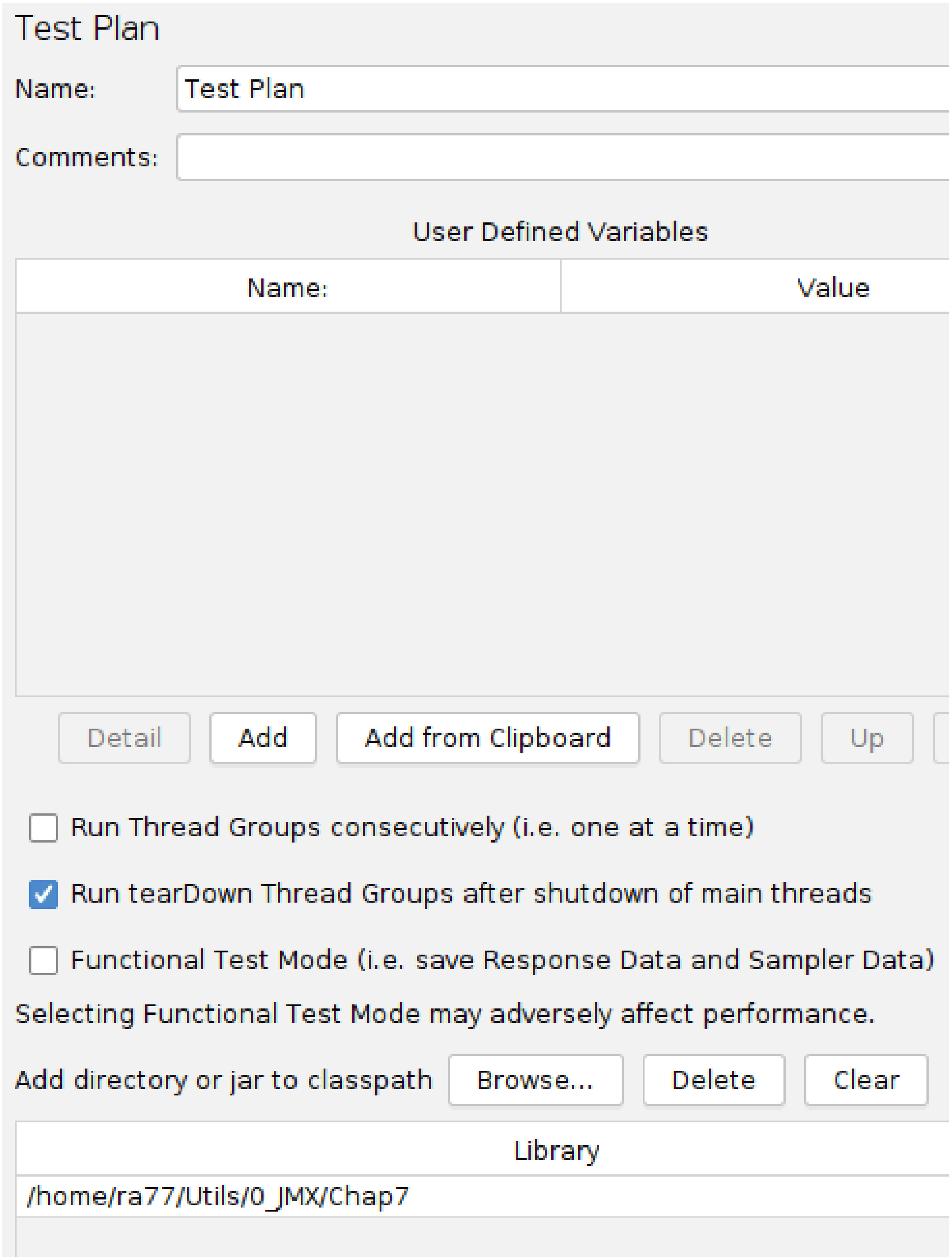
Our ActiveMQ Artemis configuration

Let's create a *jndi.properties* file.

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQIn\
itialContextFactory
connectionFactory.ConnectionFactory=tcp://MyServer:61616
topic.topic/JMeterBookTopic=JMeterBookTopic
```

With the default configuration, ActiveMQ Artemis blocks the producer if the disk is filled by more than 90%. Modify it if necessary.

The first thing to do is to add our *jndi.properties* file in the *classpath* of JMeter. The chosen solution is to use the *Add directory or jar to classpath* function in **Test Plan**.


Test Plan

Name: Test Plan

Comments:

User Defined Variables

Name:	Value

☐ Run Thread Groups consecutively (i.e. one at a time)

✓ Run tearDown Thread Groups after shutdown of main threads

☐ Functional Test Mode (i.e. save Response Data and Sampler Data)

Selecting Functional Test Mode may adversely affect performance.

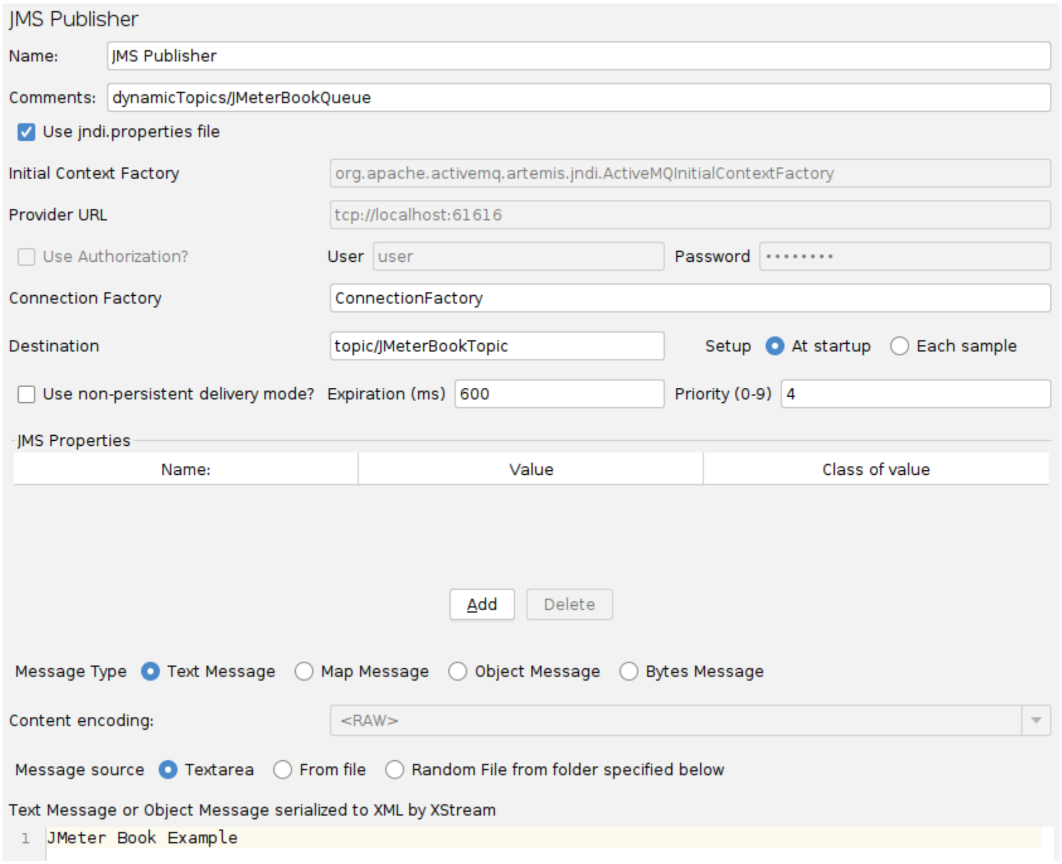
Add directory or jar to classpath

Library
/home/ra77/Utils/0_JMX/Chap7

Add jndi.properties in the JMeter classpath

Let's add a **JMS Publisher** to our script and configure it as below:

- *Use jndi.properties file* checked
- *Connection Factory = ConnectionFactory*
- *Destination = topic/JMeterBookTopic*


JMS Publisher

Name:	JMS Publisher
Comments:	dynamicTopics/JMeterBookQueue
✓ Use jndi.properties file	
Initial Context Factory	org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory
Provider URL	tcp://localhost:61616
☐ Use Authorization?	User: user; Password: ••••••••
Connection Factory	ConnectionFactory
Destination	topic/JMeterBookTopic
Setup	● At startup ○ Each sample
☐ Use non-persistent delivery mode?	Expiration (ms): 600; Priority (0-9): 4

JMS Properties

Name:	Value	Class of value

Message Type ● Text Message ○ Map Message ○ Object Message ○ Bytes Message

Content encoding: <RAW>

Message source ● Textarea ○ From file ○ Random File from folder specified below

Text Message or Object Message serialized to XML by XStream

```
JMeter Book Example
```

JMS Publisher

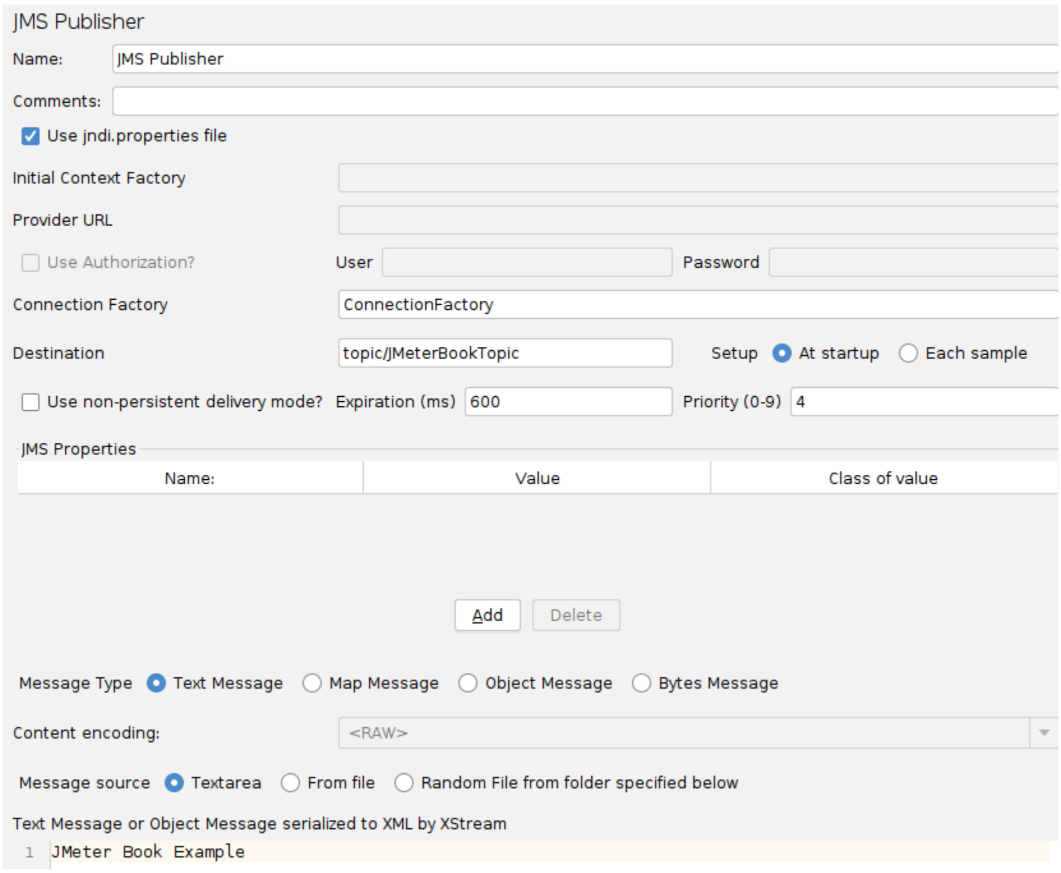
Finally, we execute our script to produce messages and validate the correct behaviour of our consumer.

Example 3: Testing the configuration of a MOM server with Publish/Subscribe

For the part *JMS Publish*, we will take the example above and complete it.

In a new **Thread Group**, add a **JMS Subscriber** with the following options:

- *Use jndi.properties file* checked
- *Connection Factory = ConnectionFactory*
- *Destination = topic/JMeterBookTopic*



JMS Publisher

Name: JMS Publisher

Comments:

[✓] Use jndi.properties file

Initial Context Factory

Provider URL

[ ] Use Authorization? User Password

Connection Factory: ConnectionFactory

Destination: topic/JMeterBookTopic Setup [✓ At startup] Each sample

[ ] Use non-persistent delivery mode? Expiration (ms) 600 Priority (0-9) 4

JMS Properties

Name:	Value	Class of value

Add Delete

Message Type [✓ Text Message] Map Message Object Message Bytes Message

Content encoding: <RAW>

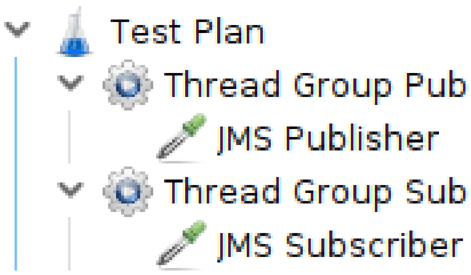
Message source [✓ Textarea] From file Random File from folder specified below

Text Message or Object Message serialized to XML by XStream

```
JMeter Book Example
```

JMS Subscriber

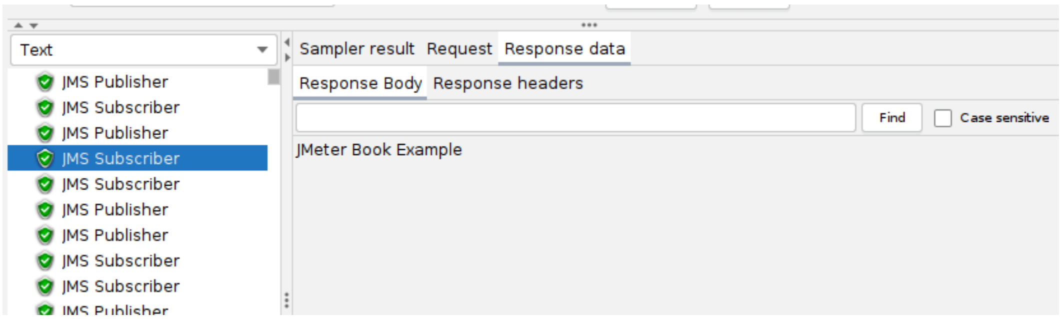
Our script looks like this:



JMS Pub/Sub Script

Let's execute it;

During the validation phase of the script, we can use a **View Results Tree** for this purpose.

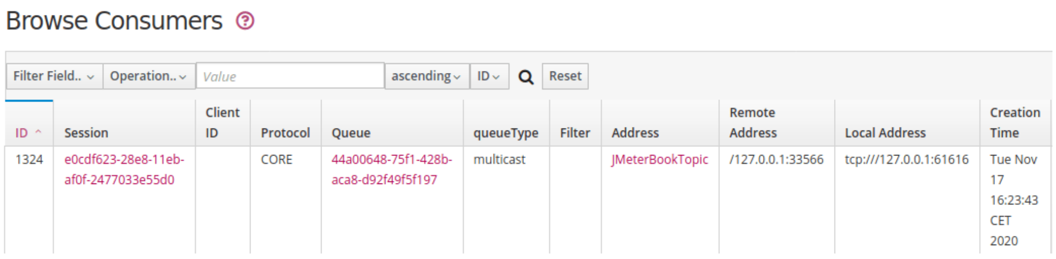


JMeter Book Example

Validation of the receipt of the message with a View Results Tree

During the load test, the ActiveMQ Administration Console Artemis will come to the rescue.

Validation that the consumer is present.

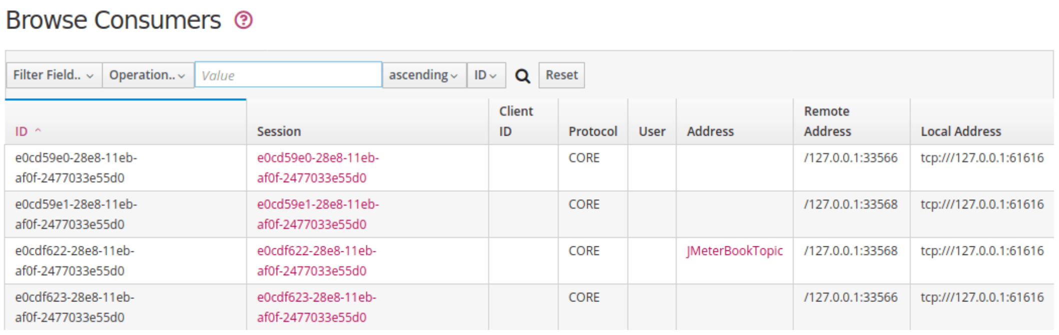

Browse Consumers

Filter Field.. | Operation.. | Value | ascending | ID | Reset

ID	Session	Client ID	Protocol	Queue	queueType	Filter	Address	Remote Address	Local Address	Creation Time
1324	e0cdf623-28e8-11eb-af0f-2477033e55d0		CORE	44a00648-75f1-428b-aca8-d92f49f5f197	multicast		JMeterBookTopic	/127.0.0.1:33566	tcp:///127.0.0.1:61616	Tue Nov 17 16:23:43 CET 2020

Consumer validation

Validation that the producer is present.

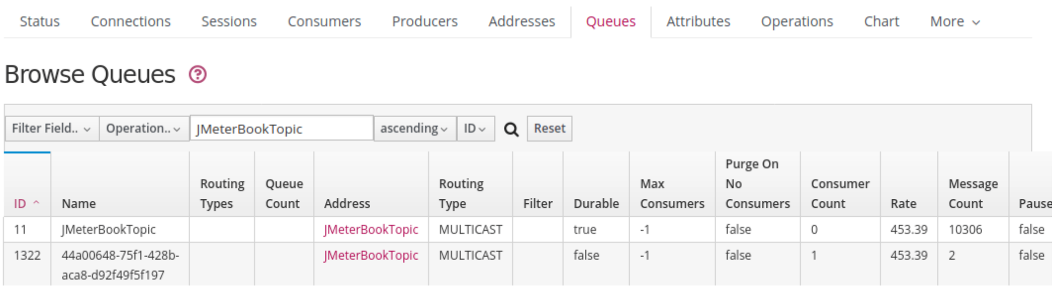

Browse Consumers

Filter Field.. | Operation.. | Value | ascending | ID | Reset

ID	Session	Client ID	Protocol	User	Address	Remote Address	Local Address
e0cd59e0-28e8-11eb-af0f-2477033e55d0	e0cd59e0-28e8-11eb-af0f-2477033e55d0		CORE			/127.0.0.1:33566	tcp:///127.0.0.1:61616
e0cd59e1-28e8-11eb-af0f-2477033e55d0	e0cd59e1-28e8-11eb-af0f-2477033e55d0		CORE			/127.0.0.1:33568	tcp:///127.0.0.1:61616
e0cdf622-28e8-11eb-af0f-2477033e55d0	e0cdf622-28e8-11eb-af0f-2477033e55d0		CORE		JMeterBookTopic	/127.0.0.1:33568	tcp:///127.0.0.1:61616
e0cdf623-28e8-11eb-af0f-2477033e55d0	e0cdf623-28e8-11eb-af0f-2477033e55d0		CORE			/127.0.0.1:33566	tcp:///127.0.0.1:61616

Validation of the producer

Validation that the topic works.



Browse Queues

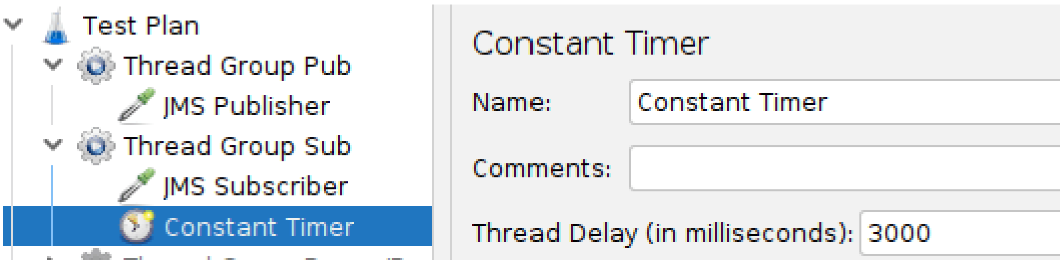
Filter Field.. | Operation.. | JMeterBookTopic | ascending | ID | Reset

ID	Name	Routing Types	Queue Count	Address	Routing Type	Filter	Durable	Max Consumers	Purge On No Consumers	Consumer Count	Rate	Message Count	Pause
11	JMeterBookTopic			JMeterBookTopic	MULTICAST		true	-1	false	0	453.39	10306	false
1322	44a00648-75f1-428b-aca8-d92f49f5f197			JMeterBookTopic	MULTICAST		false	-1	false	1	453.39	2	false

Topic validation

Now, our customer asks: “*What happens if consumers take more time to consume the messages?*”.

To do this, we only need to add a **Constant Timer** to our **Thread Group** containing **JMS Subscriber**.



Constant Timer

Name: Constant Timer

Comments:

Thread Delay (in milliseconds): 3000

Slow consumer with a Constant Timer

Our customer asks: "*What happens if the Topic is paused for five minutes?*".

To simulate this, we will use the ActiveMQ Artemis management capabilities with its API[17] to pause the topic.

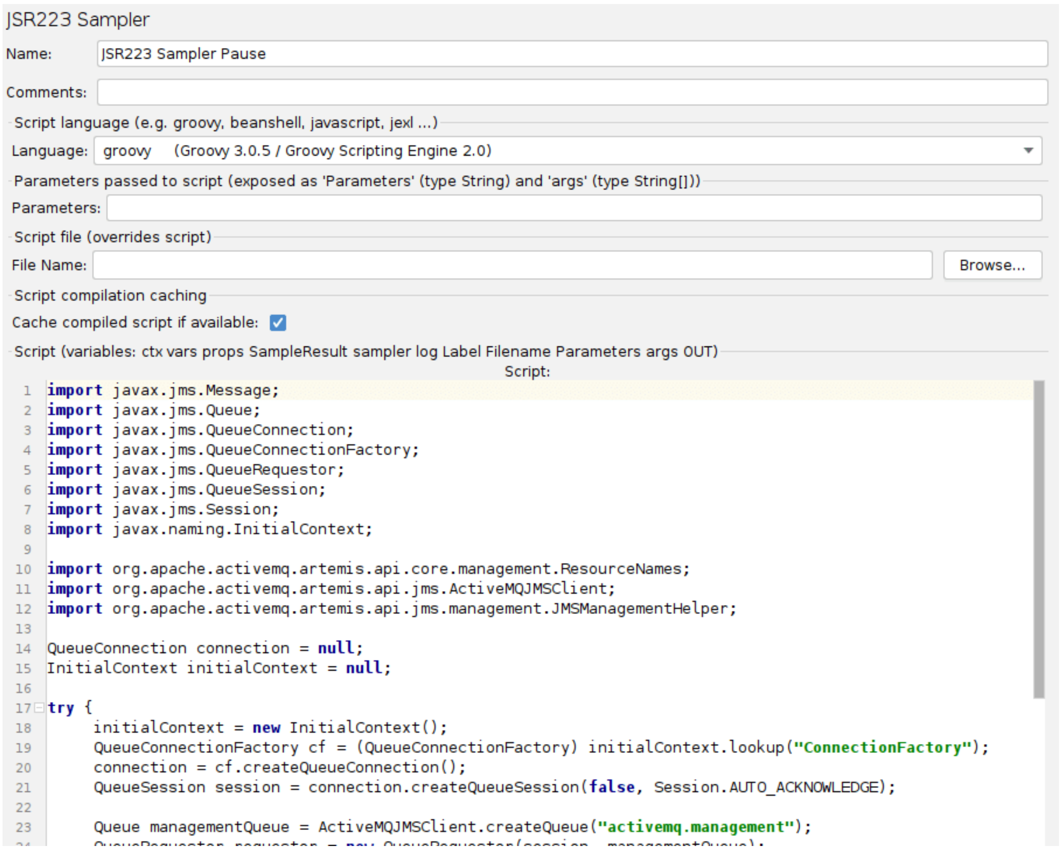
The script to pause the topic is as follows.

```
import javax.jms.Message;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueRequestor;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.naming.InitialContext;

import org.apache.activemq.artemis.api.core.management.ResourceNames;
import org.apache.activemq.artemis.api.jms.ActiveMQJMSClient;
import org.apache.activemq.artemis.api.jms.management.JMSManagementHelp\
er;

QueueConnection connection = null;
InitialContext initialContext = null;

try {
        initialContext = new InitialContext();
        QueueConnectionFactory cf = (QueueConnectionFactory) initialContext.lo\
```

[17]https://activemq.apache.org/components/artemis/documentation/latest/management.html

```
okup("ConnectionFactory");
        connection = cf.createQueueConnection();
        QueueSession session = connection.createQueueSession(false, Session.AU\
TO_ACKNOWLEDGE);

        Queue managementQueue = ActiveMQJMSClient.createQueue("activemq.manage\
ment");
        QueueRequestor requestor = new QueueRequestor(session, managementQueue\
);
        connection.start();
        Message m = session.createMessage();
        JMSManagementHelper.putOperationInvocation(m, ResourceNames.QUEUE + "J\
MeterBookTopic", "pause");
        Message reply = requestor.request(m);

        return "OK";

} finally {
        if (initialContext != null) {
                initialContext.close();
        }
        if (connection != null) {
                connection.close();
        }
}
```

Let’s add this script in a **JSR223 Sampler**.


JSR223 Sampler

Name: JSR223 Sampler Pause

Comments:

Script language (e.g. groovy, beanshell, javascript, jexl ...)

Language: groovy (Groovy 3.0.5 / Groovy Scripting Engine 2.0)

Parameters passed to script (exposed as 'Parameters' (type String) and 'args' (type String[]))

Parameters:

Script file (overrides script)

File Name:

Script compilation caching

Cache compiled script if available: ✓

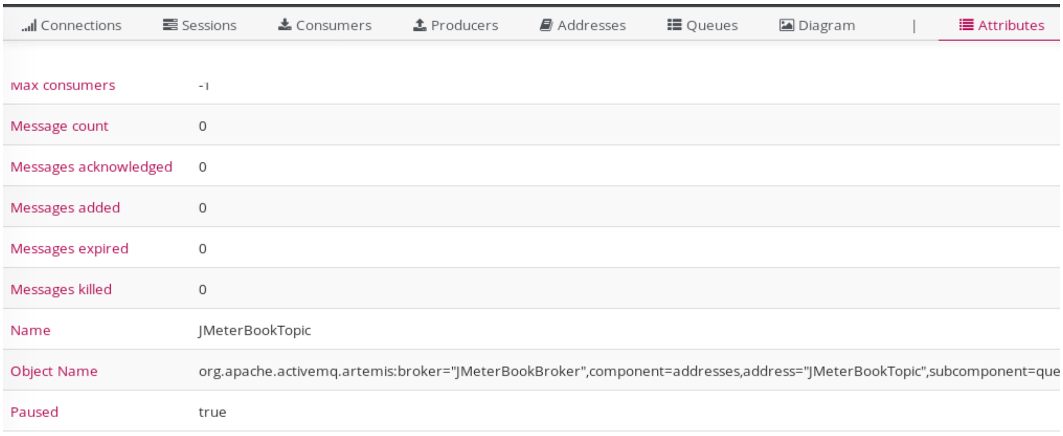
Script (variables: ctx vars props SampleResult sampler log Label Filename Parameters args OUT)

Script:

```
import javax.jms.Message;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueRequestor;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.naming.InitialContext;

import org.apache.activemq.artemis.api.core.management.ResourceNames;
import org.apache.activemq.artemis.api.jms.ActiveMQJMSClient;
import org.apache.activemq.artemis.api.jms.management.JMSManagementHelper;

QueueConnection connection = null;
InitialContext initialContext = null;

try {
    initialContext = new InitialContext();
    QueueConnectionFactory cf = (QueueConnectionFactory) initialContext.lookup("ConnectionFactory");
    connection = cf.createQueueConnection();
    QueueSession session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

    Queue managementQueue = ActiveMQJMSClient.createQueue("activemq.management");
```

The pause step

Validate that it works well using the Artemis ActiveMQ management interface.



Max consumers	-1
Message count	0
Messages acknowledged	0
Messages added	0
Messages expired	0
Messages killed	0
Name	JMeterBookTopic
Object Name	org.apache.activemq.artemis:broker="JMeterBookBroker",component=addresses,address="JMeterBookTopic",subcomponent=que
Paused	true

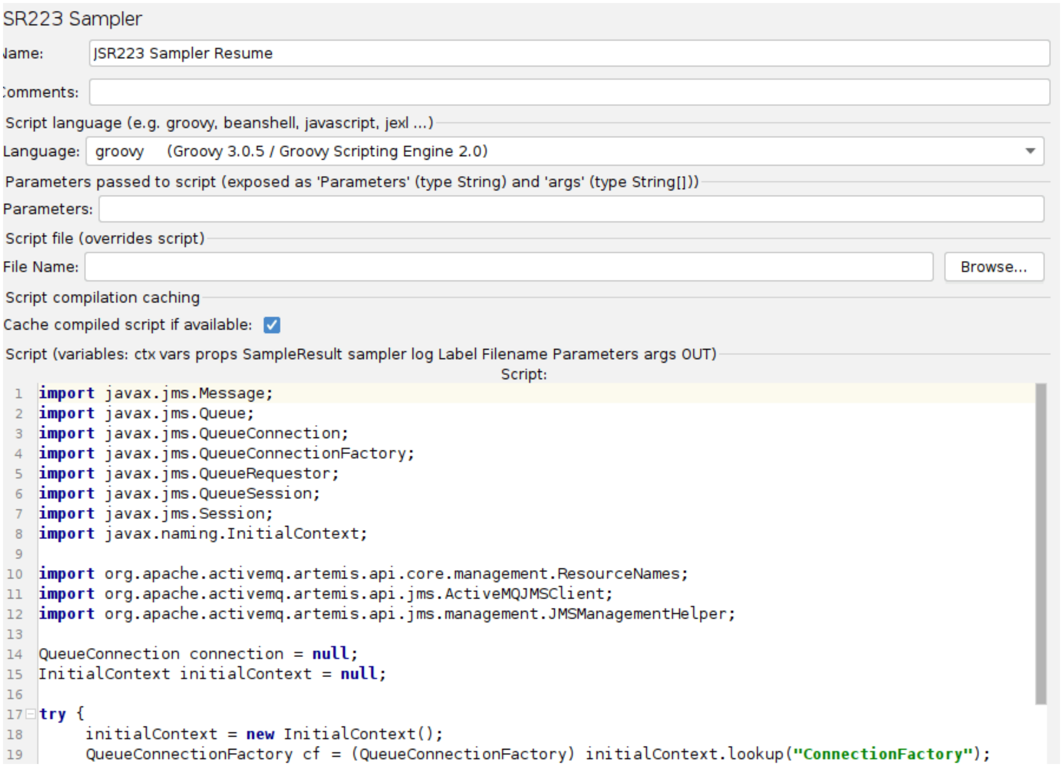
Confirm topic pause

Now let's add the resume step.

The script is almost the same, we just call "resume" instead of "pause"

```
import javax.jms.Message;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueRequestor;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.naming.InitialContext;

import org.apache.activemq.artemis.api.core.management.ResourceNames;
import org.apache.activemq.artemis.api.jms.ActiveMQJMSClient;
import org.apache.activemq.artemis.api.jms.management.JMSManagementHelp\
er;

QueueConnection connection = null;
InitialContext initialContext = null;

try {
        initialContext = new InitialContext();
        QueueConnectionFactory cf = (QueueConnectionFactory) initialContext.lo\
okup("ConnectionFactory");
        connection = cf.createQueueConnection();
        QueueSession session = connection.createQueueSession(false, Session.AU\
TO_ACKNOWLEDGE);

        Queue managementQueue = ActiveMQJMSClient.createQueue("activemq.manage\
ment");
        QueueRequestor requestor = new QueueRequestor(session, managementQueue\
);
        connection.start();
        Message m = session.createMessage();
        JMSManagementHelper.putOperationInvocation(m, ResourceNames.QUEUE + "J\
MeterBookTopic", "resume");
        Message reply = requestor.request(m);

        return "OK";

} finally {
        if (initialContext != null) {
                initialContext.close();
        }
        if (connection != null) {
                connection.close();
        }
}
```

Let's add this script in a **JSR223 Sampler**.



SR223 Sampler

lame: JSR223 Sampler Resume

omments:

Script language (e.g. groovy, beanshell, javascript, jexl ...)

Language: groovy (Groovy 3.0.5 / Groovy Scripting Engine 2.0)

Parameters passed to script (exposed as 'Parameters' (type String) and 'args' (type String[]))

Parameters:

Script file (overrides script)

File Name:

Script compilation caching

Cache compiled script if available: ✓

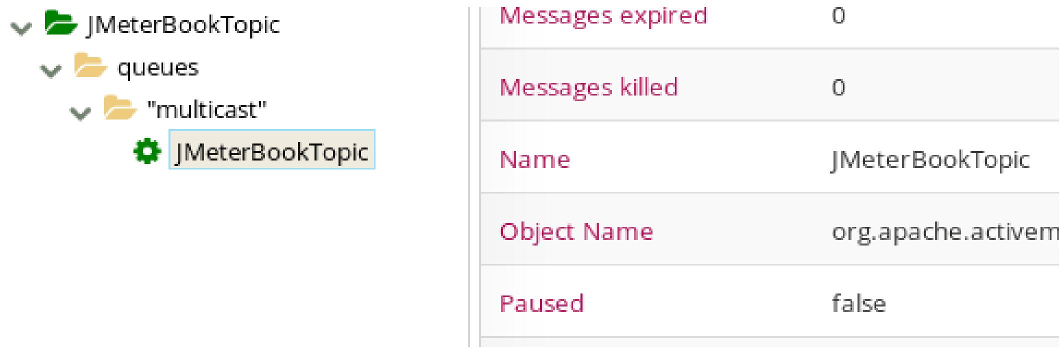
Script (variables: ctx vars props SampleResult sampler log Label Filename Parameters args OUT)

Script:

```
import javax.jms.Message;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueRequestor;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.naming.InitialContext;

import org.apache.activemq.artemis.api.core.management.ResourceNames;
import org.apache.activemq.artemis.api.jms.ActiveMQJMSClient;
import org.apache.activemq.artemis.api.jms.management.JMSManagementHelper;

QueueConnection connection = null;
InitialContext initialContext = null;

try {
     initialContext = new InitialContext();
     QueueConnectionFactory cf = (QueueConnectionFactory) initialContext.lookup("ConnectionFactory");
```

The resume step

Validate that it works well using the Artemis ActiveMQ management interface.



JMeterBookTopic
queues
"multicast"
JMeterBookTopic

Messages expired	0
Messages killed	0
Name	JMeterBookTopic
Object Name	org.apache.activem
Paused	false

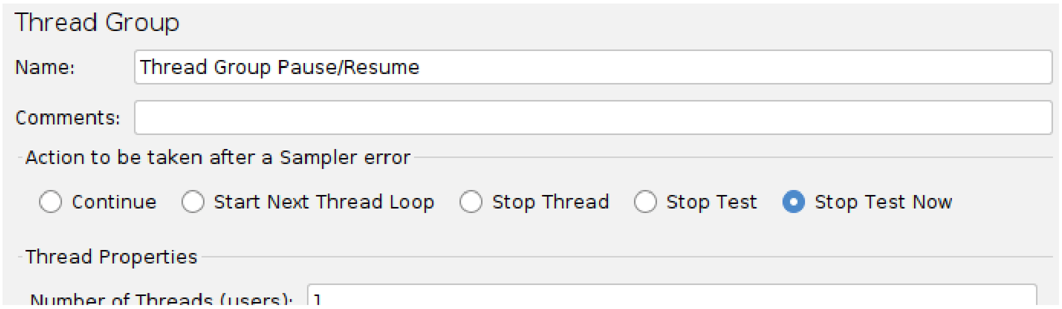
Check status of topic

Let's put these two **JSR223 Samplers** in a **Thread Group**.

Then add a **Flow Control Action** to pause for five minutes (300,000 ms) between the two.

Let's add a **Response Assertion** to check that the script returns the word *OK*.

Do not forget to configure our **Thread Groups** to stop the test if an error occurs during the manipulation (*pause* and *resume*) of our topic.



Thread Group

Name: Thread Group Pause/Resume

Comments:

Action to be taken after a Sampler error

Continue | Start Next Thread Loop | Stop Thread | Stop Test | [✓ Stop Test Now]

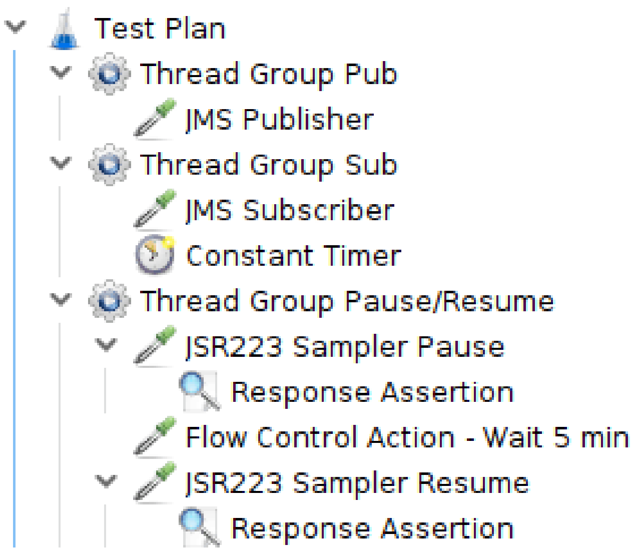
Thread Properties

Number of Threads (users): 1

Configuration of our Thread Groups

Another thing to remember, run this **Thread Groups** at the right time depending on the test case (at the beginning of the test, wait for peak load, etc.).

Our final test plan looks like:



Test Plan
- Thread Group Pub
  - JMS Publisher
- Thread Group Sub
  - JMS Subscriber
  - Constant Timer
- Thread Group Pause/Resume
  - JSR223 Sampler Pause
    - Response Assertion
  - Flow Control Action - Wait 5 min
  - JSR223 Sampler Resume
    - Response Assertion

Our final test plan

Example 4: Test any MOM

It may happen that testing our MOM with JMeter's standard JMS elements is impossible.

For example, in the following cases:

- In-house framework that encapsulates JMS calls
- MOM functionality not supported by JMS
- Etc.

We will be able to work around this problem thanks to **JSR223 Sampler**.

The Java development of the consumer and the producer JMS is finished, let's go to the tests.

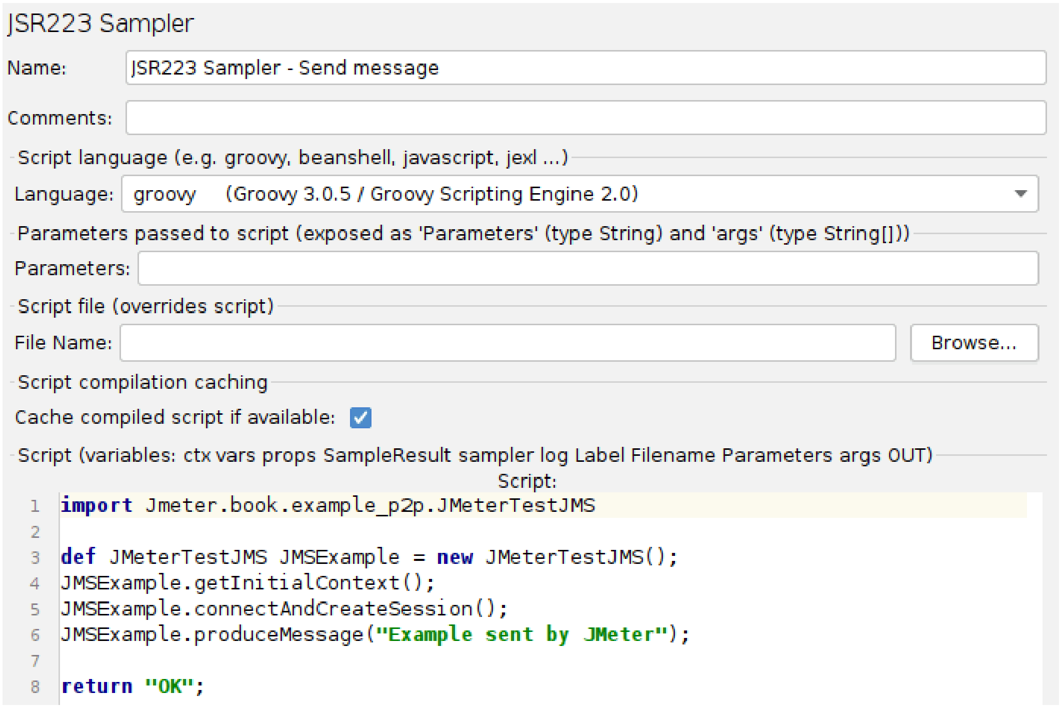
The producer and the consumer are distributed as a jar file.

To test them, let's add this jar file to JMeter's Classpath so that we can use these new JMS consumer and producer.

Do not forget to restart JMeter to take into account the new jar files.

All we need to do is change the JMS elements with **JSR223 Sampler** elements and use a Groovy script.

A simple import of the package into the Groovy script will be enough (here, it will be *import jmeter.book.example_p2p.JMeterTestJMS*).


JSR223 Sampler

Name: JSR223 Sampler - Send message

Comments:

Script language (e.g. groovy, beanshell, javascript, jexl ...)

Language: groovy (Groovy 3.0.5 / Groovy Scripting Engine 2.0)

Parameters passed to script (exposed as 'Parameters' (type String) and 'args' (type String[]))

Parameters:

Script file (overrides script)

File Name:

Script compilation caching

Cache compiled script if available: ✓

Script (variables: ctx vars props SampleResult sampler log Label Filename Parameters args OUT)

Script:

```
import Jmeter.book.example_p2p.JMeterTestJMS

def JMeterTestJMS JMSExample = new JMeterTestJMS();
JMSExample.getInitialContext();
JMSExample.connectAndCreateSession();
JMSExample.produceMessage("Example sent by JMeter");

return "OK";
```

Using the JMS transmitter with Groovy

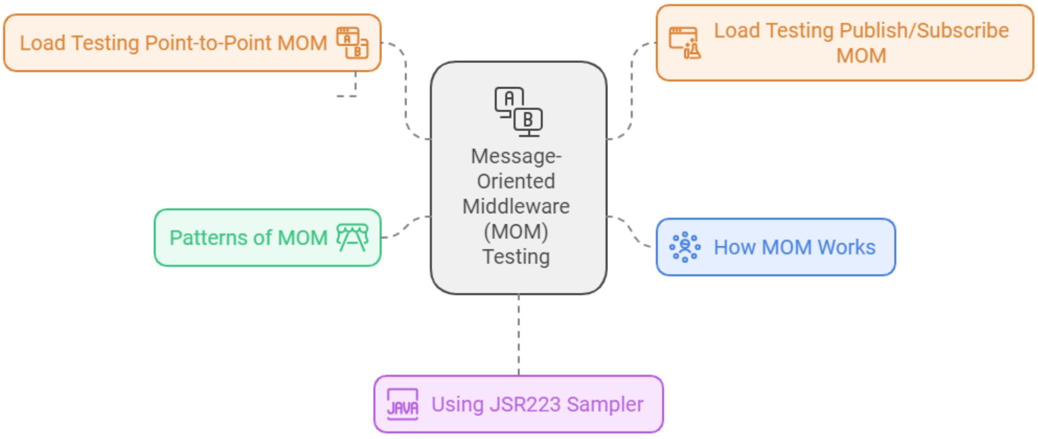
And now, to have a more realist script, it's possible to change the message sent using a **CSV Data Set Config** or from a table in the database.

Conclusion

In this chapter we have seen the main features of a MOM:

- Asynchronicity
- Decoupling
- Back pressure
- Point-to-point communication
- Communication Publication/Subscription
- Broadcast
- Priority of messages
- Persistence of messages
- JMS Selector
- Durable Subscriber
- Etc.

We have tested those features in our JMeter scripts using *Apache ActiveMQ Classic* and *Apache ActiveMQ Artemis*.
Then we saw how to take advantage of the management features of the MOM using JMeter (**JSR223 Sampler** and API Rest).
When the core JMS elements of JMeter are not sufficient, it is still possible to do our tests using the integration between JMeter and Groovy via a **JSR223 Sampler**.
Once again, you see the endless possibilities that JMeter offers.


Load Testing Point-to-Point MOM
Load Testing Publish/Subscribe MOM
Message-Oriented Middleware (MOM) Testing
Patterns of MOM
How MOM Works
Using JSR223 Sampler