Laboratory practices JMS: Java Message Service (2 sessions)

Concurrency and Distributed Systems

Introduction

This practice aims to develop a simple application using the **Java Message Service** (JMS) messaging service. In particular, we will work with a chat application with the same graphical interface as in the previous practice, although it uses JMS, and a ChatRobot will be developed for it.

Given the introductory nature of this practice, basic aspects such as connecting to the JMS provider and sending and receiving messages will be covered. Other aspects that would be fundamental for the development of a real application, such as security and fault tolerance, have been intentionally left out for the sake of simplicity.

The estimated duration of this practice is two weeks. Each week you should attend a laboratory session where you will be able to discuss your progress with the lab professor and resolve any doubts you may have. Please note that you will need to allocate some time from your personal work to complete the practice.

This practice is prepared to be performed on DSIC-Linux systems from Polilabs.

Activity 1

In this activity we will install, configure and run a JMS provider. The JMS provider to be installed is Apache ActiveMQ Artemis (simply *Artemis* in the rest of this document).

Artemis is a set of Java libraries and applications distributed as a compressed archive. For this practice, we provide students with an installation script (*install-artemis-csd.sh*) that facilitates the installation process. In the appendix we explain in detail the contents of the script, as well as a brief description of the Artemis commands that we will need for our practice.

Artemis is available for download from the Apache ActiveMQ web site[1] . To avoid disk space quota problems, the current Artemis distribution is available on the asigDSIC folder, common to all users. We will access this folder through Polilabs. In the event that a student wishes to install Artemis on his or her personal computer, the Annex to this document explains how to proceed.

Instructions

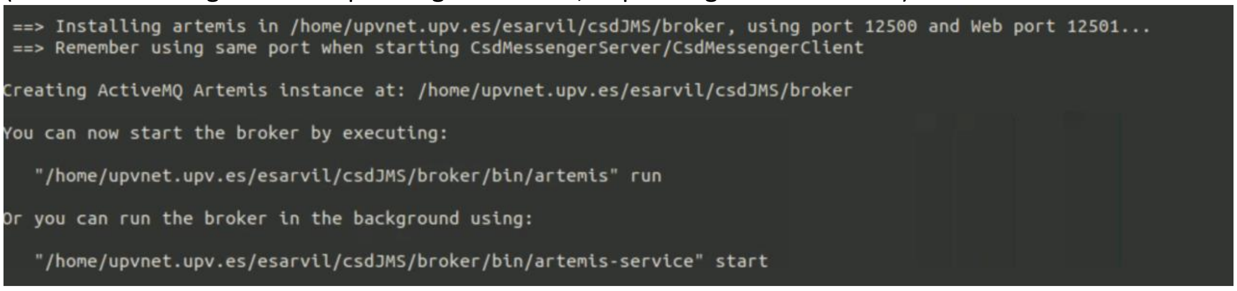
1. Log in to DSIC-Linux from Polilabs. Open a terminal and create a directory called `csdJMS` in your home. This directory is where the Artemis broker will be installed.

```
$ cd
$ mkdir csdJMS
```

2. Now go to the directory `asigDSIC/ETSINF/csd/jms/artemis/bin` and run the `install-artemis-csd.sh` installation script. As parameters of this script, you should specify a **port number** and the directory where the Artemis broker should be installed (which in this case will be the `broker` directory, inside the `csdJMS` directory created in the previous step. ~ indicates the home folder). Regarding the port number, you can use port 12500, as shown below:

```
$ cd
$ cd asigDSIC/ETSINF/csd/jms/artemis/bin
$ bash install-artemis-csd.sh 12500 ~/csdJMS/broker
```

If the installation is successful, the terminal will display a message similar to the following (`esarvil` being the corresponding user name, depending on the student).



```
==> Installing artemis in /home/upvnet.upv.es/esarvil/csdJMS/broker, using port 12500 and Web port 12501...
==> Remember using same port when starting CsdMessengerServer/CsdMessengerClient

Creating ActiveMQ Artemis instance at: /home/upvnet.upv.es/esarvil/csdJMS/broker

You can now start the broker by executing:

   "/home/upvnet.upv.es/esarvil/csdJMS/broker/bin/artemis" run

Or you can run the broker in the background using:

   "/home/upvnet.upv.es/esarvil/csdJMS/broker/bin/artemis-service" start
```

This message indicates that the broker will use port 12500 and the Artemis Web console will use port 12501. It also tells us how to launch the broker. We will do this in the next step.

3. To launch the Artemis broker, go to your home, then to the directory `csdJMS/broker/bin` and execute the command `artemis run`:

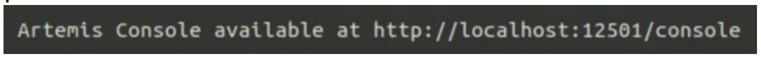
[1] https://activemq.apache.org/components/artemis/download/

```
$ cd
$ cd csdJMS/broker/bin
$ ./artemis run
```

If everything works correctly, you will see a message from the broker on the screen:

```
     _        _               _
    / \  ____| |_  ___ __  __(_) _____
   / _ \|  _ \ __|/ _ \  \/  | |/  __/
  / ___ \ | \/ |_/  __/ |\/| | |\___ \
 /_/   \_\|   \__\____|_|  |_|_|/___ /
 Apache ActiveMQ Artemis 2.17.0
```

And among the many messages that appear, at the end you will be told that the console is available on port 12501.



```
Artemis Console available at http://localhost:12501/console
```

Remember that, once Artemis is installed, from the directory `~/csdJMS/broker/bin`, you can start it with the command:

```
$ ./artemis run
```

and you can stop it typing Ctrl-C in the same terminal.

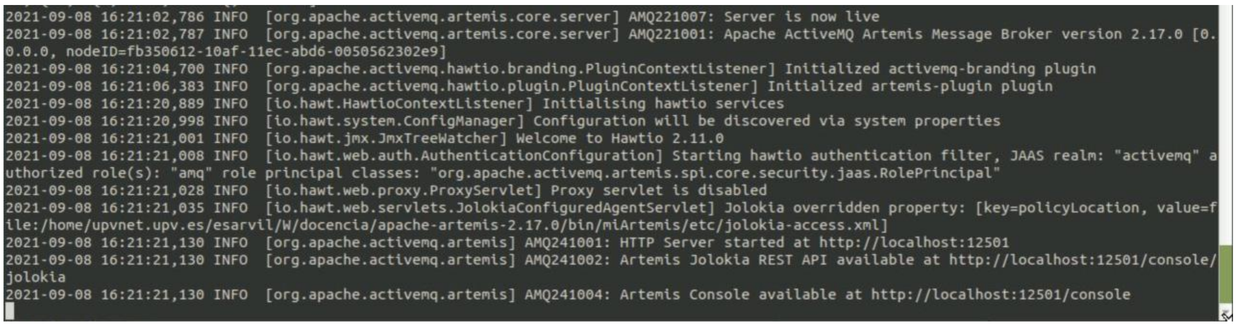
Important: stop Artemis at the end of each session. **You will have to reinstall it and launch it at the start of each practice session**, because we have installed it in the home of the virtual machine, which is self-destructing.

Note: if you wish to have Artemis installed permanently, you can also install it on your W disk. This way, you will not need to reinstall it every time you want to use it, although access to the web console will be much slower (due to the configuration and characteristics of the W disk).

Verification

In the Artemis installation performed using the provided script, a port number has been specified where Artemis will be listening (in our case, port 12500). The port for the Artemis Web console has been set to the indicated port + 1 (in our case, port 12501). In addition, the administrator user has been defined (user "admin", password "admin") and a queue called "csd" has been created.

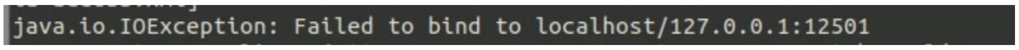
When the broker is launched, if everything works correctly, we will see on the screen that the Artemis server is running, as well as a set of messages like this:



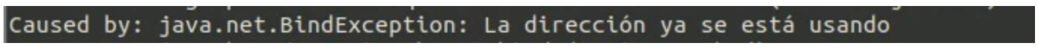
```
2021-09-08 16:21:02,786 INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
2021-09-08 16:21:02,787 INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.17.0 [0.
0.0.0, nodeID=fb350612-10af-11ec-abd6-0050562302e9]
2021-09-08 16:21:04,700 INFO  [org.apache.activemq.hawtio.branding.PluginContextListener] Initialized activemq-branding plugin
2021-09-08 16:21:06,383 INFO  [org.apache.activemq.hawtio.plugin.PluginContextListener] Initialized artemis-plugin plugin
2021-09-08 16:21:20,889 INFO  [io.hawt.HawtioContextListener] Initialising hawtio services
2021-09-08 16:21:20,998 INFO  [io.hawt.system.ConfigManager] Configuration will be discovered via system properties
2021-09-08 16:21:21,001 INFO  [io.hawt.jmx.JmxTreeWatcher] Welcome to Hawtio 2.11.0
2021-09-08 16:21:21,008 INFO  [io.hawt.web.auth.AuthenticationConfiguration] Starting hawtio authentication filter, JAAS realm: "activemq" a
uthorized role(s): "amq" role principal classes: "org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal"
2021-09-08 16:21:21,028 INFO  [io.hawt.web.proxy.ProxyServlet] Proxy servlet is disabled
2021-09-08 16:21:21,035 INFO  [io.hawt.web.servlets.JolokiaConfiguredAgentServlet] Jolokia overridden property: [key=policyLocation, value=f
ile:/home/upvnet.upv.es/esarvil/W/docencia/apache-artemis-2.17.0/bin/miArtemis/etc/jolokia-access.xml]
2021-09-08 16:21:21,130 INFO  [org.apache.activemq.artemis] AMQ241001: HTTP Server started at http://localhost:12501
2021-09-08 16:21:21,130 INFO  [org.apache.activemq.artemis] AMQ241002: Artemis Jolokia REST API available at http://localhost:12501/console/
jolokia
2021-09-08 16:21:21,130 INFO  [org.apache.activemq.artemis] AMQ241004: Artemis Console available at http://localhost:12501/console
```

These messages indicate, among other things, that the Artemis server is active and available at http://localhost:12501 (i.e. at PORT NUMBER + 1, where PORT NUMBER is the port indicated in the installation) and that the Artemis console is available at http://localhost:12501/console.

Note: If the indicated port is already occupied (for example, because several users are using the same virtual machine and one of them has already created and launched his Artemis broker), you will get error messages of the following type:



```
java.io.IOException: Failed to bind to localhost/127.0.0.1:12501
```



```
Caused by: java.net.BindException: La dirección ya se está usando
```

To solve this error, delete the contents of the broker directory you have previously created and repeat steps 2 and 3 using another port number (e.g. 12510, 12520, etc.) If in doubt, consult the professor.

Activity 2

This activity will test the operation of the Chat application implemented using Java Message Service, available in DistributedChatJMS.

Previous concepts

DistributedChatJMS uses the same graphical interface as the previous practice. We now have a "server" application (ChatServerJMS), which takes care of the connected users and available channels (and could be used to control user authentication, although this is not taken into account now); and a client application (ChatClientJMS), which uses a graphical interface to allow a user to connect to the Chat application, join chat channels and send messages to users in that channel (both messages to the channel, and private messages to a user).

In the proposed application, it would not really be necessary to have a ChatServerJMS server, if the clients were the ones requesting the Artemis broker to create their own queues and the channel topics were already initially created. However, we have included the ChatServerJMS server to take care of managing the connected users and available channels. This way we avoid different users connecting with duplicate nicknames and we can keep track of how many users connect to each channel.

When the ChatServerJMS server is launched for the first time, it will create the "Linux", "Spain" and "Friends" channels. And it will ask Artemis to create a multicast queue for each channel created, to which it will subscribe, in order to receive LEAVE messages and keep the list of subscribers to the channel up to date. Each of these queues is called `channel-NAME` (NAME being the name of the corresponding channel).

Note: In Artemis there is no concept of "topic". The concept that approximates the most is a MULTICAST address. These multicasts addresses can be used as JMS topics when using Artemis as a JMS provider.

The ChatServerJMS server will also ask Artemis to create the `ChatServer` queue, of type anycast (equivalent to the JMS "queue" destination), through which it will receive messages from users.

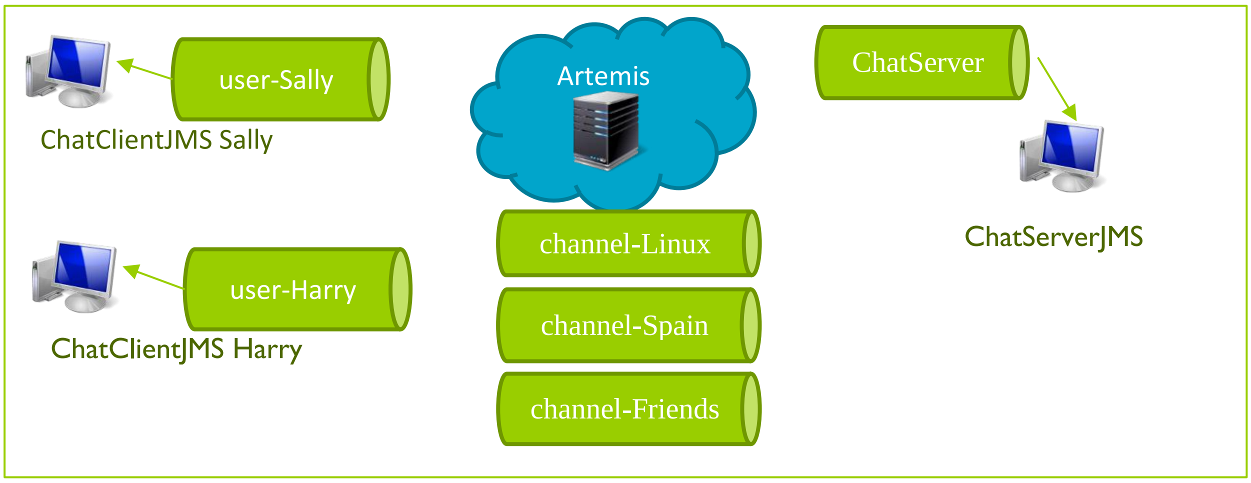
On the other hand, when a user connects for the first time, our ChatServerJMS server instructs "artemis" to create a queue for the user (of type anycast), with the name `user-USERNAME` (where USERNAME is the specific name of the user). This queue will exist permanently, until Artemis is reinstalled.

The main advantage of having a dedicated ChatServerJMS server is the independence of the provider in the management of users, so that the ChatServerJMS server controls how many users are connected at any given time to the channels.

At any given time only one ChatServerJMS application should be active, but several ChatClientJMS can be active, one for each user connected to the system.

The "user-USERNAME" queues are managed by ChatServerJMS on demand, so ChatServerJMS requests the Artemis broker to create them the first time a user logs on to the system.

The following figure summarizes the previously mentioned components, assuming users Sally and Harry:



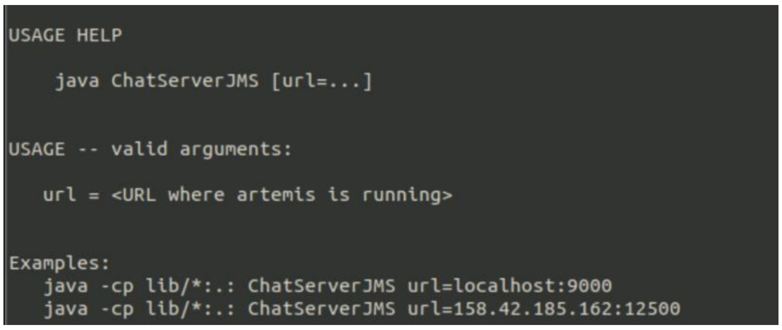
Instructions

1. Download the file ***DistributedChatJMS.tgz*** provided for the practice (available on PoliformaT) to your W drive, unzip it and access the newly created directory. Specifically:
```
   $ tar xvf DistributedChatJMS.tgz
   $ cd DistributedChatJMS
```

2. In the DistributedChatJMS directory you will find the files `ChatClientJMS.java` and `ChatServerJMS.java`, corresponding to the application provided in this practice. Compile these files:
```
   $ javac -cp lib/*:. *.java
```

3. Launch the **ChatServerJMS** application, indicating as input parameter the address where the Artermis broker is listening (in our case, localhost:12500).
```
   $ java -cp lib/*:. ChatServerJMS url=localhost:12500
```

Note: The following are the allowed parameters for the ChatServerJMS application

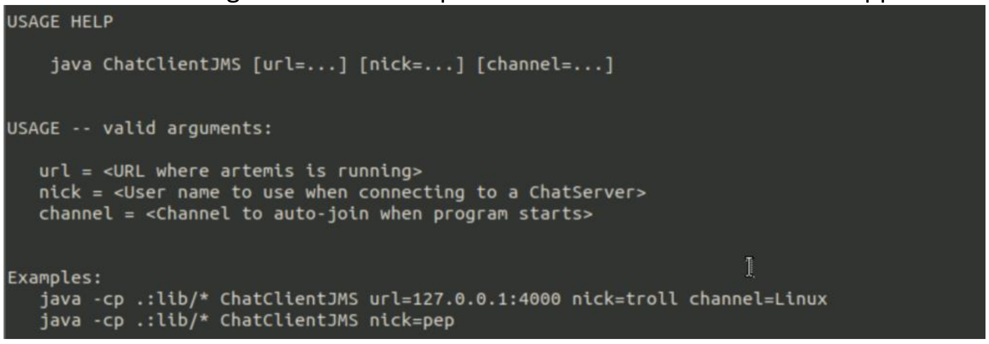


```
USAGE HELP

    java ChatServerJMS [url=...]

USAGE -- valid arguments:

   url = <URL where artemis is running>

Examples:
   java -cp lib/*:.: ChatServerJMS url=localhost:9000
   java -cp lib/*:.: ChatServerJMS url=158.42.185.162:12500
```

4. In another terminal, run the ChatClientJMS application, indicating as input parameter the address where the Artermis broker is listening (in our case, localhost:12500).
   ```
   $ java -cp lib/*:. ChatClientJMS url=localhost:12500
   ```

5. You can run several ChatClientJMS applications. We recommend that you launch them from different terminals.

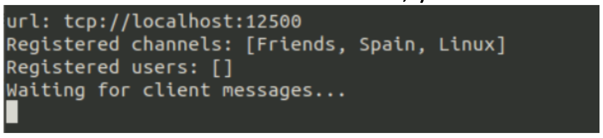
Note: The following are the allowed parameters for the ChatClientJMS application



```
USAGE HELP

    java ChatClientJMS [url=...] [nick=...] [channel=...]

USAGE -- valid arguments:

   url = <URL where artemis is running>
   nick = <User name to use when connecting to a ChatServer>
   channel = <Channel to auto-join when program starts>

Examples:
   java -cp .:lib/* ChatClientJMS url=127.0.0.1:4000 nick=troll channel=Linux
   java -cp .:lib/* ChatClientJMS nick=pep
```

Verification

When ChatServerJMS is launched, you will see messages like these on the screen:



```
url: tcp://localhost:12500
Registered channels: [Friends, Spain, Linux]
Registered users: []
Waiting for client messages...
```

The ChatServerJMS server has created the "Friends", "Spain" and "Linux" channels. In addition, initially we will not have any user registered in the system.

When you launch ChatClientJMS, you will see the same graphical interface as in the previous practice, where you can enter your user name and, once connected, you will see the list of available channels. Note that in the Server field you will see the url of the Artemis broker (`tcp://localhost:12500` in our case).

Use client applications for chatting. Check that it is possible to send messages to a user, even if the user is not currently logged on, and that messages are delivered when the user subsequently logs on. This is because the messages are stored in the JMS provider queues.

a) Checking the application using a single machine

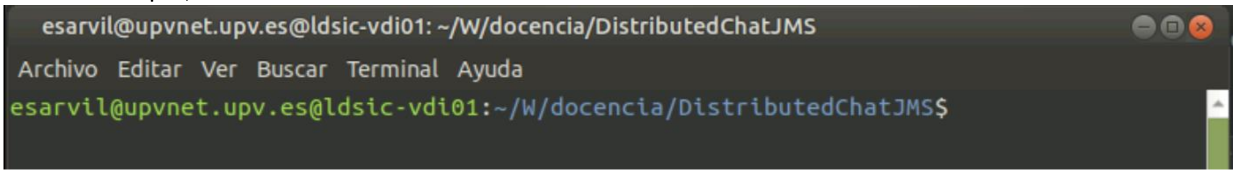
First, we are going to test the Artemis broker and the ChatServerJMS application that we have launched using two ChatClientJMS applications that we will launch on the same machine.

To do this, launch two ChatClientJMS applications, for example one for user Harry and another one for user Sally. Check that they can talk to each other, through some channel (for example, on the Linux channel). Note whether a client, when leaving a channel and re-entering, can read messages that have been sent to that channel in its absence. Also note the messages displayed by ChatServerJMS on your terminal.

Next, from user Harry send several private messages to user Sally. On user Harry's screen, the interface will display the destination "Chatting area Sally". Check that Sally receives the private messages from Harry. If so, close the Sally client application, but keep the chatting area with Sally and continue to send more messages from Harry to Sally.
Important: do not change user or channel, or you will lose the Chat area with Sally and you will not be able to select this user anymore.

Re-open Sally's client application and you should notice that you are now receiving the messages that were sent while you were not connected. Keep these client applications open while you perform the next section.

b) Checking the application using several machines
In this case we are going to run the ChatClientJMS application on another machine and connect it to our broker and to ChatServerJMS. The terminal prompt tells you which machine you are connected to. In this example, the user `esarvil` is connected to the machine `ldsic-vdi01`:

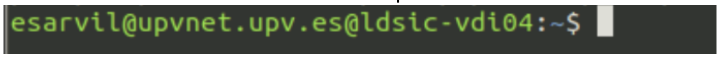

esarvil@upvnet.upv.es@ldsic-vdi01: ~/W/docencia/DistributedChatJMS
Archivo Editar Ver Buscar Terminal Ayuda

```
esarvil@upvnet.upv.es@ldsic-vdi01:~/W/docencia/DistributedChatJMS$
```

Next, connect via ssh to another dsic virtual machine (for example, to `ldsic-vdi04`), and launch several clients to interact with the broker you had launched in the previous section. For example, assuming that our server resides on `ldsic-vdi01`, and that we now want to connect to `ldsic-vdi04`, we would perform the following steps:
1º) We connect to the machine `ldsic-vdi04` via ssh

```
$ ssh -X ldsic-vdi04
```

2º) Next, we enter the password and, if everything works correctly, we will be already connected to the indicated machine. For example:

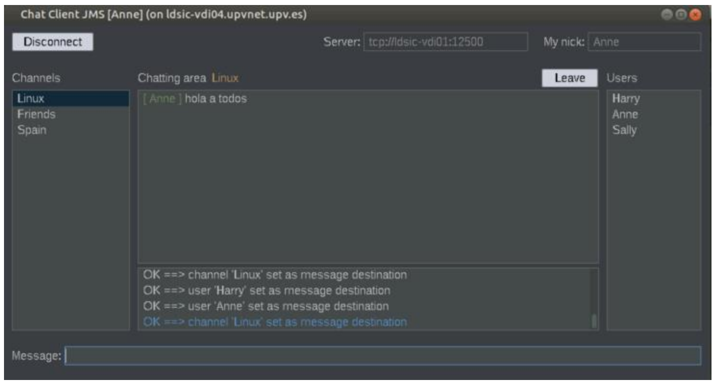


```
esarvil@upvnet.upv.es@ldsic-vdi04:~$
```

3º) We go to the DistributedChatJMS directory of W created previously and launch ChatClientJMS indicating the machine where our server resides (in this case, in ldsic-vdi01).

```
$ cd W/csd/DistributedChatJMS
$ java -cp lib/*:. ChatClientJMS url=ldsic-vdi01:12500
```

Connect to the server with a new user name (e.g. Anne). If you still have the client applications from the previous section open, you will see that users Sally, Harry and Anne can now chat with each other. Note also that the url of the Artemis broker (`tcp://ldsic-vdi01:12500` in this case) appears in the Server field.



```
[ Anne ] hola a todos
```

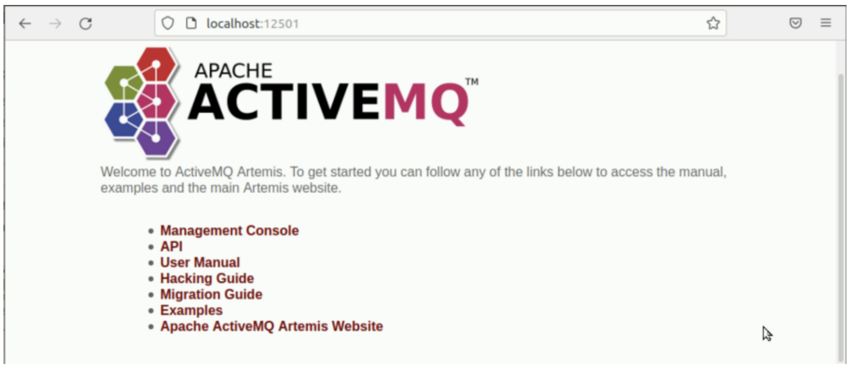
```
OK ==> channel 'Linux' set as message destination
OK ==> user 'Harry' set as message destination
OK ==> user 'Anne' set as message destination
OK ==> channel 'Linux' set as message destination
```

You can also connect to another student's Artemis broker by entering the url address of your colleague's broker when launching your ChatClientJMS application, similar to what is done in this section.

Activity 3

This activity is intended to consult the web console for Artemis management. As mentioned above, once Artemis has been launched on port 12500, we can access the Artemis console from a browser with the address: `localhost:12501`

The following window will appear, where, in addition to the user manual and examples, we can access the Artemis *Management Console*.

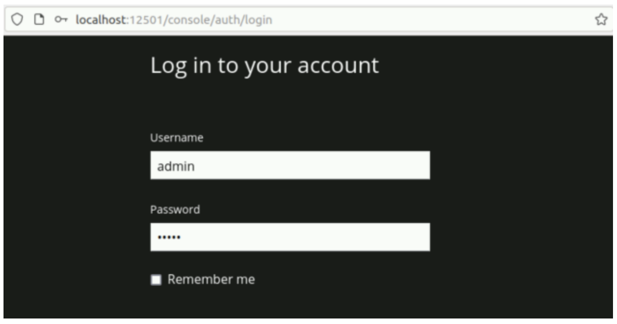


Welcome to ActiveMQ Artemis. To get started you can follow any of the links below to access the manual, examples and the main Artemis website.

- Management Console
- API
- User Manual
- Hacking Guide
- Migration Guide
- Examples
- Apache ActiveMQ Artemis Website

Instructions

1. Access the Artemis **`Management Console`**, available at `http://localhost:12501/console` from any browser. You will be prompted for a username and password. You must enter *admin* (for Username) and *admin* (for Password).



localhost:12501/console/auth/login

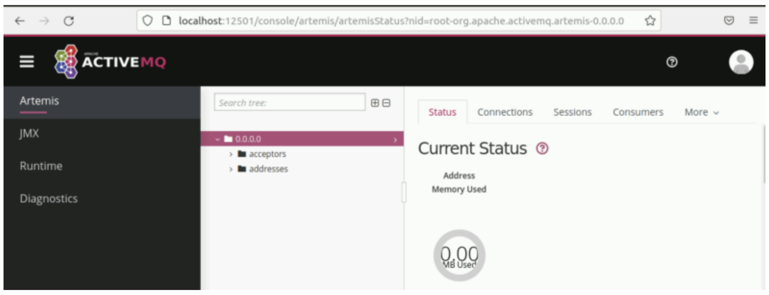
Log in to your account

Username: admin

Password: •••••

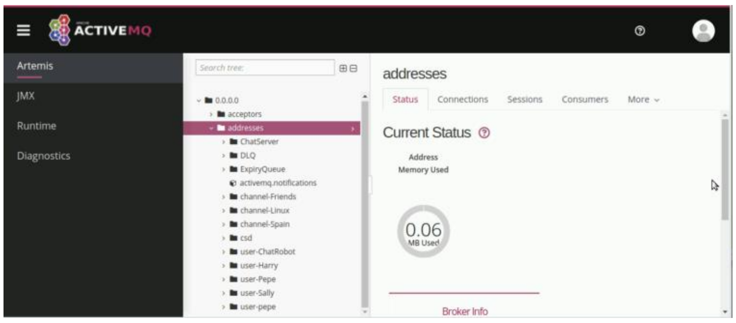
Remember me

2. Once in the Artemis console, you will see the following screen, where the status is shown and you will be able to consult the existing queues and connections.



localhost:12501/console/artemis/artemisStatus?nid=root-org.apache.activemq.artemis-0.0.0.0

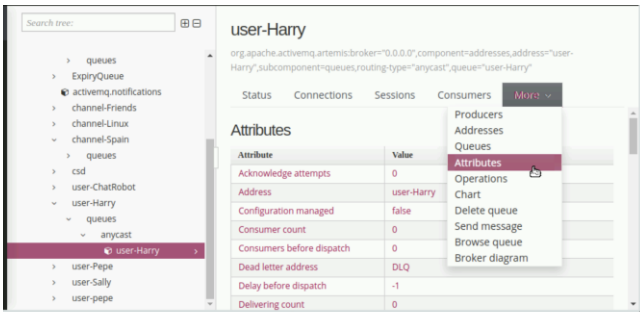
To do so, click on "addresses" to see the queues that have been created.



Check that the following addresses (destinations in Artemis) are available:

- **chatServer**: a destination for the ChatServerJMS server, with an associated "queue anycast" queue, which is equivalent to a JMS queue.
- **channel-Friends, channel-Linux, channel-Spain**: that is, a destination for each chat channel. All of them have an associated "queue multicast" queue, which is equivalent to the JMS Topic.
- **user-Harry, user-Sally** (or corresponding user-Nick): that is, a destination for each user that has connected to the chat at some point. In this case, each one will have an associated "queue anycast" queue, which is equivalent to a JMS queue.
- **csd**: this is the queue that has been created in the Artemis installation process (as indicated in the installation script). It has not been used in our Chat application. But it allows us to show how queues can be created in the Artemis installation process.

If we click on each of the queues associated to these addresses, we can consult their attributes. For example, if we click on the user-Harry queue and select the menu option `More – Attributes` (as shown in the following figure), we will see the specific characteristics of that queue. In this case, the queue has been created as "anycast" and the Max Consumers attribute has been set to 1. This implies that only 1 consumer will be able to read messages from this queue.



user-Harry

org.apache.activemq.artemis:broker="0.0.0.0",component=addresses,address="user-Harry",subcomponent=queues,routing-type="anycast",queue="user-Harry"

Attributes

Attribute	Value
Acknowledge attempts	0
Address	user-Harry
Configuration managed	false
Consumer count	0
Consumers before dispatch	0
Dead letter address	DLQ
Delay before dispatch	-1
Delivering count	0

Note: In our application, we have used ChatServerJMS to control that two users cannot be using the same name at the same time (i.e. they cannot connect as user "Harry" both at the same time). However, with the queue properties provided by Artemis, this would already be controlled, since the user queues have been created as anycast and with Max Consumer value set to 1, then there can only be at most 1 consumer associated to that queue. [2]

To conclude this activity, indicate in the following table the value of Max Consumer for the queues, as well as the type associated with that queue (i.e. anycast / multicast). Comment why you think they have these values.

Queues	Max Consumer	Queue type	Comment
user-Harry	1	anycast	Only the user Harry has to read from the queue
chatServer	1	anycast	
channel-Friends	-1	multicast	Multiple users can connect to the channel at the same time
csd	-1	anycast	

Activity 4

We want to implement a ChatRobot that initially connects to a channel and greets each time a user enters that channel. To do this, the DistributedChatJMS directory provides the **ChatRobot** file, which contains a partial implementation of the ChatRobot functionality for JMS. In this activity, the student is expected to complete the ChatRobot implementation to provide the following functionality:

- The ChatRobot application will connect to the Spain channel by default and will send a greeting message to the `channel-Spain` queue every time a user connects to that channel.

Instructions

In the file `ChatRobot.java`, complete the main method as indicated. As a guide, you can use the ChatClientJMS code provided with this practice. Specifically, you should:

1. Establish the connection to Artemis and obtain the server queue.
    a) Connect to Artemis, using the connection factory called "ConnectionFactory".

[2] The queues are created through the Artemis broker. To configure the type of queues to be created, our application uses the ArtemisCore class (available in the utils_jms folder). In this class, in the createQueue method, Artemis is requested to create the corresponding addresses and queues. In the case of a queue of type anycast, the number of Max Consumers is set to 1.

b) Create a default JMS context.
c) Get the server queue of our application (called “ChatServer”). This name is stored in the SERVER_QUEUE constant of the Destinations class (in utils_jms package).

2. Send a CONNECT message to the server queue (i.e. ChatServer queue) and wait for a response. This response includes the list of users and channels registered on the server.
   a) Create a producer.
   b) Construct a message of type "ConnectMessage", to tell the ChatServerJMS server that we want to connect to the chat. Use the MessageFactory class to construct the message of type ConnectMessage.
   c) Create a temporary queue to receive the response from the server.
   d) Assign this temporary queue to the "replyTo" property of the message, so that the server, when receiving our message, knows which queue to reply to.
   e) Using the producer created previously, send the message to the ChatServer queue.
   f) Create a consumer for our temporary queue.
   g) The consumer waits for a message to be received. If everything goes well, the server will create, if necessary, the queue associated to the user and will send a message of type ConnectOKMessage. Optionally, the student can implement the necessary code to read the received message and obtain from this message the list of users and channels.

3. Now we will make ChatRobot connect to the indicated channel and send a welcome message. To do this, you must:
   a) Construct a message of type "Join", to indicate to the ChatServerJMS server that we want to join the channel. Assign the temporary queue created in step 2 to the "replyTo" property of the message, in order to let the server, when receiving our message, know which queue it should reply to.
   b) Send the created message, using the producer we created in step 2, to the ChatServer queue.
   c) Get the reference to the JMS topic associated to the channel (Note: this step is already provided in the code).
   d) Create a message (of type ChatMessage) to send a welcome message to the channel. Send it using the producer.

4. Finally, we will make ChatRobot iterate indefinitely, waiting for messages in the channel. It will process each received message and, if it is a JOIN message, it will send a greeting message to the channel. As a guide, you can use the ChatServerJMS code (check the runServerLoop method of that class). In our case, you should:
   a) Create a consumer for the topic associated with the channel.
   b) The consumer waits for the reception of a message, with blocking wait.
   c) In case of receiving a message of type USER_JOINS, send a message to the channel, welcoming the particular user that joined the channel.

Activity 5 – Optional

In this activity, which is completely optional, the aim is to complete the ChatRobot application, providing it with the following functionality:

- When a user connects to the channel to which the ChatRobot is connected, it will send a greeting message to both the user queue and the channel queue.
- When a user disconnects from the channel to which the ChatRobot is connected, it will send a goodbye message to both the user queue and the channel queue.

- Since ChatRobot will appear as another user in the list of chat users, any other connected user may send it a private message. In this case, the ChatRobot should reply with a message such as "I am a bot".

APPENDIX 1: Artemis installation

A detailed explanation on the configuration of the Apache ActiveMQ Artemis server is available at:

https://activemq.apache.org/components/artemis/documentation/1.5.3/using-server.html

The following is a description of the most relevant aspects of this server related to this practice.

Download latest version of Artemis

To install Artemis, access the website:

https://activemq.apache.org/components/artemis/download/past_releases

and download ***apache-artemis-2.17.0-bin.tar.gz*** to a directory on your disk. Unzip this file and go to the directory that has been created. Access the `bin` subdirectory. Inside you will see the `artemis` file that we will use to create an Artemis broker.

Creation of an Artemis broker. To install the broker into a directory DIR, open a terminal, go to the `bin` directory that contains the `artemis` file, and execute the command:

```
$ ./artemis create DIR
```

Note: in this practice we make use of the script "install-artemis-csd.sh" which, in addition to installing Artemis, sets certain parameters for its installation, detailed in APPENDIX 3.

APPENDIX 2: Questions and Answers on Artemis

How to start Artemis?

Artemis can be started with the following command, which will launch the broker in the foreground:

```
$ ./artemis run
```

Or, to start it in the background:

```
$ ./artemis-service start
```

In our practice, it is recommended to start it with the "run" command, because in case the broker has been configured incorrectly, it will show messages for those points of the configuration where there are problems. And we will be able to try to solve them.

How to stop Artemis?

If Artemis has been started in the foreground (with “artemis run”), it can be stopped by typing Ctrl-C at the terminal. If it was started in the background, it can be stopped with the command:

```
$ ./artemis-service stop
```

How to reset Artemis?

To delete all the mailboxes created and thus be able to use Artemis again as if it had just been installed, we must stop the Artemis service (see above), delete the files in the "data" directory of the installed broker and reactivate the Artemis service (with `artemis run`).

APPENDIX 3: Detail of the Artemis installation script provided in practice

For this practice we have provided the `install-artemis-csd.sh` script that facilitates the Artemis installation process. In this Appendix we will describe the most relevant aspects of the commands included in this script. This script requires the user to enter two input parameters: PORT and DIR, which correspond to the port number (PORT) where Artemis will be listening when it is launched, and to the directory (DIR) where Artemis will be installed. The script checks that these two values have been entered and that they are correct (i.e. that PORT is a number greater than 1024 and that DIR is not an existing directory). All this checking is done in lines 1 to 47.

In line 50, the port number for the Artemis Web console is stored in WPORT (which will be the port number indicated by the user plus 1, i.e. PORT + 1). And in lines 59 onwards we proceed to install Artemis using the command `./artemis create DIR` with the options discussed below (we indicate the most relevant for our practice):

`--default-port $PORT`	Port number where artemis will be listening (PORT is the value specified by the user).
`--http-port $WPORT`	Port number for the artemis web console, where WPORT=PORT+1.
`--allow-anonymous`	Enables anonymous security configuration.
`--user admin`	Indicates the user name ("admin" in this case).
`--password admin`	Indicates the user's password ("admin" in this case).
`--queues csd:anycast`	Creates a queue called "csd", of type anycast (point-to-point messaging, equivalent to the "Queue" concept seen in theory, in which a message sent by a producer has only one consumer).

Note: Artemis also allows the creation of "multicast" queues, with publish-subscribe messaging, equivalent to the "Topic" concept seen in theory, where messages are sent to all consumers subscribed to a given address. Multicasts can be used as JMS Topics when using Artemis as a JMS provider.

`--no-autocreate`	Disables automatic address creation. An address represents a messaging endpoint. Within the configuration, a typical address is given a unique name, 0 or more queues and a routing type.
`--relax-jolokia`	Disables strict checking in jolokia-access.xml. Jolokia is one of the mechanisms provided by Artemis to modify server configuration and manage resources (queues and addresses).
`--no-autotune`	Disables auto-tuning in the journal.
`--no-amqp-acceptor`	Disables the AMQP-specific acceptor
`--no-hornetq-acceptor`	Disables the HornetQ specific acceptor.
`--no-mqtt-acceptor`	Disables the MQTT specific acceptor.
`--no-stomp-acceptor`	Disables the STOMP-specific acceptor.

The acceptor element contains a URI that defines the type of Acceptor to create along with its configuration. Each acceptor defines how connections to the Apache ActiveMQ Artemis server can be made. In this case all the specific acceptors have been disabled to leave only the acceptor used by Artemis to listen for connections on the port indicated above.