Guided Exercise: Configuring Journals and Other Settings

In this lab, you will configure a custom dead-letter queue and change journal parameters.

Resources	
Files:	**ND**
App URL:	**http://localhost:9990**
	http://localhost:8080/messaging-client

Results

You should be able to configure a custom dead-letter queue for an application queue and change the number and size of message journal files.

before you start

This guided exercise uses two simple test applications:

1. **messaging-client.war is a web application that connects to java:/jms/ CustomCF JMS ConnectionFactory and post messages to java:/jms/queue/ JB248TestQueue JMS Queue. Accepts text from the user to use as message properties and the message body.**
2. **messaging-mdb.jar is a Message Driven Bean (MDB) that connects to the same JMS ConnectionFactory and consumes messages from the same JMS Queue as the above application. Displays the message body and properties in the EAP server log.**

Both applications were deployed by the previous guided exercise, on a separate server instance.

Before beginning the guided exercise, run the following command to verify that EAP is installed in /opt/jboss-eap-7.0, that no EAP instances are running, that JMS resources are configured, and that test applications are implemented:

```
[student@workstation ~]$ lab messaging-persistence setup
```

1. **Start a standalone EAP 7 server instance.**

   Start an instance of EAP using the /home/student/JB248/labs/ standalone folder as the base directory, but using the standalone-full.xml configuration file.

   Use the following commands:

   ```
   [student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
   [student@workstation ~]$ ./standalone.sh --server-config=standalone-full.xml \ -Djboss.server.base.dir=/
   home/student/JB248/labs/standalone/
   ```

Leave this terminal running the standalone EAP server instance.

2. Create a new JMS Queue to be used as a DLQ.

2.1. Open another terminal window and start the CLI:

```
[student@workstation ~]$ /opt/jboss-eap-7.0/bin/jboss-cli.sh --connect
```

2.2. Create a jms-queue named TestDLQ, mapped to the JNDI name java:/jms/ JB248TestDLQ:

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default
[standalone@localhost:9990 server=default] ./jms-queue=TestDLQ:add(\ entries=[java:/jms/
JB248TestDLQ])
```

23. Inspect TestDLQ to get your internal ActiveMQ address:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestDLQ:\ read-
attribute(name=queue-address)
```

The expected result is:

```
{
        "outcome" => "success", "result"
        => "jms.queue.TestDLQ" }
```

3. Configure the new queue as the DLQ for the TestQueue.

3.1. Inspect TestQueue to get its internal ActiveMQ address:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:\ read-
attribute(name=queue-address)
```

The expected result is:

```
{
        "outcome" => "success", "result"
        => "jms.queue.TestQueue"
}
```

3.2. Configure TestDLQ as the DLQ only for the TestQueue, and for moving messages after a single failed delivery attempt.

Use the internal addresses queried in the previous steps to identify the queues when creating the address-setting:

```
[standalone@localhost:9990 server=default] ./address-setting=\
jms.queue.TestQueue:add(dead-letter-address=jms.queue.TestDLQ,\ max-delivery-
attempts=1)
```

3.3. Inspect the newly created address-setting element to see that although it defines related dead-letter and redelivery attributes, it does NOT specify any queue and expiration delay:

```
[standalone@localhost:9990 server=default] ./address-setting=\ jms.queue.TestQueue:read-
resource
```

The expected result is similar to:

```
{
        "outcome" => "success", "result"
        => { "address-full-
                policy" => "PAGE", "auto-create-jms-queues"
                => false, "auto-delete-jms-queues" => false,
                "dead-letter-address" => "jms.queue.TestDLQ",
                "expiry-address" => undefined, "expiry-delay" => -1L", "last-
                value-queue" => false, "max-delivery-
                attempts" => 1", "max-
                redelivery-delay" => 0L", "max-size-
                bytes" => -1L, "message-counter-history-
                day-limit" => 0, "page-max-cache-size" =>
                5, "page-size-bytes" => 10485760L,
                "redelivery-delay" => 0L", "redelivery-multiplier" =>
                1.0, "redistribution-delay" => -1L,
                "send-to-dla-on-no-route" => false, "slow-
                consumer-check-period" => 5L, "slow-
                consumer-policy" => "NOTIFY", "slow-
                consumer-threshold" => -1L




        }
}
```

Note the values for dead-letter-address, max-delivery-attempts, max redelivery-delay, redelivery-delay, and redelivery-multiplier. They are all related to dead letter handling, but this book does NOT go into detail about them.

Also note the values of the expiry-delay and expiry-address attributes. This shows how the default values for a new address-setting can be unexpected. In this case, there is no expiration handling, despite having failed message handling.

Leave this terminal running the CLI. We will return to this topic later in this exercise.

4. Send a message known to be processed without problems.

4.1. Open a web browser and enter the producer's test application using the following URL: http://localhost:8080/messaging-client

4.2. Complete the web form as follows:

- Message count: 1

- Message label: first
- Message to send: test message - to be consumed

Click Send Message to generate (or send) the message.

4.3. The web form should update and show the following message in green: Messages sent successfully.

5. Send a message known to trigger processing errors.

5.1. Use the same browser tab as the previous step and the web form as follows:

- Message count: 1
- Message label: second
- Message to send: test message - will FAIL

It is important to write "FAIL" in upper case, or the processing will NOT generate errors.

Click Send Message to generate (or send) the message.

5.2. The web form should update and show the following message in green: Messages sent successfully. This message should fail but not during post, but during consumption.

Leave the producer test application browser tab open, as we will return to this topic a few times during this exercise.

6. Disable the consumer test application so that new messages are kept in the queue.

6.1. Undeploy messaging-mdb.jar. Go back to the terminal running the CLI and run the following command:

```
[standalone@localhost:9990 server=default] /deployment=messaging-mdb.jar:\ undeploy
```

6.2. Verify that the consumer test application has stopped:

```
[standalone@localhost:9990 server=default] /deployment=messaging-mdb.jar:\ read-
attribute(name=status)
```

The expected result is:

```
{
        "outcome" => "success",
        "result" => "STOPPED"
}
```

7. Send another message known to be processed without problems.

This message will NOT be processed, it will only be queued, since the consumer application was disabled during the previous step.

7.1. Go back to the browser tab running the producer's test application and complete the web form as follows:

• Message count: 1

• Message label: third

• Message to send: test message - to be queued

Click Send Message to generate (or send) the message.

7.2. The web form should update and display the following message in green: Messages sent correctly.

Note that the consumer has no way of knowing how long the message will take to be processed.

8. Verify that the first message has been consumed, that the second message has been moved to the queue created in Step 2.2 and the third messages are still in the TestQueue.

8.1. Go back to the terminal running the EAP server and verify that there are any log entries related to the processing of the first message. There should be entries similar to:

```
09:14:19,706 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
  (Thread-0 (ActiveMQ-client-global-threads-991436930)) Message Properties: Copy #1 [first]

09:14:19,717 INFO [class com.redhat.training.messaging.mdb.MessageReceiver]
  (Thread-0 (ActiveMQ-client-global-threads-991436930)) Message Body: test message - to be
  consumed
```

Remember that these log entries are generated by the MDB when consuming a message.

Notice that the registry entry about the "Message Properties" property has the value "first" as provided in Step 4.2.

8.2. Also check the EAP server logs for entries related to the second message processing failure.

There should be a few multi-line log entries, each integrating a Java stack trace, similar to:

```
09:14:19,706 ERROR [org.jboss.as.ejb3.invocation] (Thread-1 (ActiveMQ  client-global-
threads-991436930)) WFLYEJB0034: EJB Invocation failed on component
  JB248TestQueueMDB for method public void
  com.redhat.training.messaging.mdb.MessageReceiver.onMessage(javax.jms.Message):
  javax.ejb.EJBTransactionRolledbackException: Message processing failed for no
  reason
 ...
```

```
09:14:19,715 ERROR [org.apache.activemq.artemis.ra] (Thread-1 (ActiveMQ  client-global-
threads-991436930)) AMQ154004: Failed to deliver message:
  javax.ejb.EJBTransactionRolledbackException: Message processing failed for no
  reason
 ...
```

Both entries were generated by a single message processing failure. The first is from the MDB server and the second from the ActiveMQ server.
Recall from Step 3.3 that the address-setting is configured to make a single redelivery attempt (max-delivery-attempts=1), so there should be only one pair of errors matching the above list.

There should also be a warning message indicating that the message has been moved from the application queue to the configured DLQ:

```
09:14:19,745 WARN [org.apache.activemq.artemis.core.server] (Thread-20 (ActiveMQ-server

org.apache.activemq.artemis.core.server.impl.ActiveMQServerImpl
$2@3da34eba-1936375109)) AMQ222149: Message
  Reference[181]:RELIABLE:ServerMessage[messageID=181,durable=true,
userID=8ac5181d-15e7-11e6-b7ce-f31b4b7807d0,priority=4, bodySize=512, timestamp=Mon May
  09 09:11:29 EDT 2016,expiration=0, durable=true,
  address=jms.queue.TestQueue,properties=TypedProperties[Label=second,
__AMQ_CID=a011e476-15e5-11e6-b7ce-f31b4b7807d0,Copy=1]]@322081305 has reached
  maximum delivery attempts, sending it to Dead Letter Address jms.queue.TestDLQ from jms.queue.TestQueue
```

8.3. Go back to the terminal running the CLI and list all the messages in TestDLQ:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestDLQ:list-messages
```

The expected result is similar to:

```
{
    "outcome" => "success", "result"
    => [{ "JMSMessageID"
        => "ID:8ac5181d-15e7-11e6-b7ce-f31b4b7807d0", "address" => "jms.queue.TestDLQ",
        "JMSExpiration" => 0, "JMSTimestamp" =>
        1462799489118L, "Label"
        => "second", "messageID" => 200,
        "JMSDeliveryMode" =>
        "PERSISTENT",
        "_AMQ_ORIG_MESSAGE_ID" => 181,
        "JMSPriority" => 4, "__AMQ_CID" =>
        "a011e476-15e5-11e6-
        b7ce-f31b4b7807d0", "_AMQ_ORIG_QUEUE" => "jms.queue.TestQueue",
        "Copy" => 1, "_AMQ_ORIG_ADDRESS" =>

        "jms.queue.TestQueue"
    }]
}
```

Observe the message property "Label" with the value "second" as indicated in Step 5.1.

8.4. Go back to the terminal running the CLI and list all the messages in the TestQueue:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected result is similar to:

```
{
        "outcome" => "success", "result"
        => [{ "JMSPriority"
                => 4, "JMSMessageID"
                => "ID:81eda5bd-15e8-11e6-b7ce-f31b4b7807d0", "address" =>
                "jms.queue.TestQueue", "JMSExpiration" => 0,
                "__AMQ_CID" =>
                "a011e476-15e5-11e6-b7ce-f31b4b7807d0", "Copy" => 1, "JMSTimestamp"
                =>
                1462799903781L, "Label" => "third",
                "messageID" => 232,
                "JMSDeliveryMode" =>
                "PERSISTENT"
        }]
}
```

Observe the message property "Label" with the value "third" as indicated in Step 7.1.

9. **Inspect the current journal files, created based on the default EAP configuration.**

   9.1. **In the terminal running the CLI, inspect the ActiveMQ server attributes default related to journal files:**

```
[standalone@localhost:9990 server=default] :read-attribute(name=\ journal-min-files)
[standalone@localhost:9990
server=default] :read-attribute(name=\ journal-file-size)
```

The expected values are 2 and 10485760, respectively.

   9.2. **Open this terminal and list all the files in data/messaging in the base folder of the server instance, that is, /home/student/standalone/data/messaging:**

```
[student@workstation ~]$ ls -lh ~/JB248/labs/standalone/data/activemq/journal/
```

The expected result is similar to:

```
total 21M
-rw-rw-r--. 1 student student 10M May 5 20:34 activemq-data-1.amq -rw-rw-r--. 1 student
student 10M May 6 17:06 activemq-data-2.amq -rw-rw-r--. 1 student student 19 May 5 12:44
server.lock
```

Notice that the number of data files and the size of each file match the values of the attributes inspected in the previous step.

10. Reload the server instance and verify that the second message is still there en TestQueue.

10.1.In the terminal running the CLI, reload the standalone server:

```
[standalone@localhost:9990 server=default] /:reload
```

10.2.Check the log entries on the terminal running the EAP server and wait for messages indicating that the server restarted successfully.

10.3.Go back to the terminal running the CLI and list all the messages in the TestQueue:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected result is the same as in Step 8.4.

11. Change the journal file size and the number of files.

11.1.In the terminal running the CLI, increase the value of the journal-min-files attribute to 5 and the value of the jounal-file-size attribute to 20 MB:

```
[standalone@localhost:9990 server=default] :write-attribute(\ name=journal-min-
files, value=5) [standalone@localhost:9990
server=default] :write-attribute(\ name=journal-file-size, value=20971520)
```

11.2.To change these attributes, the server instance must be reloaded or restarted:

```
[standalone@localhost:9990 server=default] /:reload
```

11.3.Check the log entries on the terminal running the EAP server and wait for messages indicating that the server restarted successfully.

12. Verify that the second message is still in the TestQueue and that they were modified the journal files.

12.1.Go back to the terminal running the CLI and list all the messages in the TestQueue:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected result is the same as in Step 8.4.

12.2.Go back to the idle terminal and verify that there are now five journal files, 20 MB each:

```
[student@workstation ~]$ ls -lh ~/JB248/labs/standalone/data/activemq/journal/
```

The expected result is similar to:

```
total 91M
-rw-rw-r--. 1 student student 20M May 9 09:31 activemq-data-10.amq
```

```
-rw-rw-r--. 1 student student 20M May 9 09:31 activemq-data-11.amq -rw-rw-r--. 1 student
student 20M May 9 09:31 activemq-data-12.amq -rw-rw-r--. 1 student student 20M May 9
09:31 activemq-data-13.amq -rw-rw-r--. 1 student student 10M May 9 09:18 activemq-
data-1.amq -rw-rw-r--. 1 student student 19 May 5 12:44 server.lock
```

The numbers in each log file name are not important, only the number of files and their sizes.

13. Turn off persistence for the entire messaging subsystem.

13.1.Go back to the terminal running the CLI and change the persistence-enabled attribute to false:

```
[standalone@localhost:9990 server=default] :write-attribute(\ name=persistence-
enabled,value=false)
```

13.2.To change these attributes, the server instance must be reloaded or restarted:

```
[standalone@localhost:9990 server=default] /:reload
```

13.3.Check the log entries on the terminal running the EAP server and wait for messages indicating that the server restarted successfully.

14. Verify that the third message was lost.

14.1.Go back to the terminal running the CLI and list all the messages in the TestQueue:

```
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:list-messages
```

The expected result is similar to:

```
{
        "outcome" => "success",
        "result" => []
}
```

This shows that the queue is empty.

use

Curious students can check the journal files folder by repeating the commands from Step 12.2, and they will be surprised to see that the journal files are still there. This is expected to happen, but the files are NO longer used by EAP.

15. Cleanup: Remove the test applications and JMS resources created by this and the previous exercise, and enable messaging persistence.

15.1.Go back to the terminal running the CLI and remove the producer applications and the consumer:

```
[standalone@localhost:9990 server=default] /deployment=messaging-mdb.jar\
:remove
[standalone@localhost:9990 server=default] /deployment=messaging-client.war\ :remove
```

15.2.Delete the custom address-setting:

```
[standalone@localhost:9990 server=default] ./address-setting=\
jms.queue.TestQueue:remove
```

15.3.Delete the JMS custom Queue and ConnectionFactory resources:

```
[standalone@localhost:9990 server=default] ./pooled-connection-factory=\
custom:remove
[standalone@localhost:9990 server=default] ./jms-queue=TestDLQ:remove
[standalone@localhost:9990 server=default] ./jms-queue=TestQueue:remove
```

15.4.Re-enable persistence for the ActiveMQ server:

```
[standalone@localhost:9990 server=default] :write-attribute(\ name=persistence-
enabled,value=true)
```

15.5.Stop the standalone server instance:

```
[standalone@localhost:9990 server=default] /:shutdown
```

This concludes the guided exercise.

Lab Work: Configuring the Messaging Subsystem

In this lab work, you will configure the JMS resources for a future version of the bookstore application. While the updated application is not available, test applications are provided to test that the JMS configuration is working as expected.

Resources	
Files:	**/opt/domain**
	/home/student/JB248/labs/messaging-lab
	/tmp/messaging-client.war
	/tmp/messaging-mdb.jar
App URL:	**http://172.25.250.10:8080/messaging-client**

Result

You should be able to create a ConnectionFactory and a JMS Queue in a managed domain and deploy test applications that use them.

And Before You Get

Started You can choose either the EAP 7 Management Console or the JBoss EAP CLI to meet your goals, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has configured a managed domain with two host controllers running on the servera and serverb virtual machines, respectively, and the domain controller on the workstation virtual machine. The domain and host configuration files are stored in the /opt/domain folder on all three machines.

This lab work uses two applications:

- **messaging-client.war is the message producer: a web application that accepts text from the user and publishes it as the JMS message body and properties.**
- **messaging-mdb.jar is the consumer of the message: an application with a Bean message-driven (MDB) that consumes JMS messages and displays their properties and body in the EAP server registry.**

Both test applications use the following JMS resources:

- **custom JMS ConnectionFactory, with JNDI address java:/jms/CustomCF. It must point to the ActiveMQ in-vm connector.**
- **TestQueue JMS Queue, with JNDI address java:/jms/queue/JB248TestQueue. There is no need to configure any message value for this queue.**

Applications must be deployed on the server in group Group1, which uses the full-ha clustered profile. Fortunately, a fellow sysadmin gave him the artemis-shell script.

firewalld-rules.sh which configures Linux kernel buffers and firewall rules to match the EAP7 defaults for this profile. The script is available at /home/student/JB248/labs/messaging-lab on the workstation virtual machine.

It is important to ensure that all application server instances in the Group1 server group are consuming messages. This means that message content processing is distributed across EAP hosts and the application is scalable.

Before beginning this chapter, review the lab work, run the following command to verify that EAP is installed to /opt/jboss-eap-7.0, that no EAP instances are running, and that the managed domain is configured, and to download the JEE application package files in the /tmp folder:

```
[student@workstation ~]$ lab messaging-lab setup
```

1. Configure the host OSes for EAP multicast clustering.

   1.1. Download the artemis-firewalld-rules.sh shell script from workstation and run it on the servera virtual machine.

   1.2. Inspect the artemis-firewalld-rules.sh shell script to verify the TCP and UDP ports it opens. Verify that the new firewall rules have been applied using the firewall-cmd command.

   1.3. Verify that the new kernel values have been applied using the command sysctl.

   1.4. Repeat Step 1.1, Step 1.2, and Step 1.3 on the serverb virtual machine.

2. Start the EAP managed domain.

   2.1. Start the domain controller in the workstation virtual machine using /opt/ domain as the base folder and host-master.xml as the host configuration file. All EAP processes must be owned by the jboss user.

   2.2. Start the host controller on the server virtual machine using /opt/domain as the base folder and host-slave.xml as the host configuration file. All EAP processes must be owned by the jboss user.

   You must also provide the computer's IP address 172.25.250.10 as the argument to the -bprivate command line option. Without it, the EAP server instances will not meet to form a cluster.

   23. Start the host controller on serverb using the same parameters reported in the previous step for servera.

   Remember to replace the IP address given to the -bprivate command line option with the 172.25.250.11 IP address of serverb.

   2.4. Verify that both host controllers connect to the domain controller and form a managed domain.

3. Verify that the four server instances are connected to form an ActiveMQ cluster.

The managed domain contains another server group with two more server instances. This group is also configured to use the full-ha profile, so it gets the same clustering parameters as Group1 and its ActiveMQ integrated server instances will also join the same messaging cluster.

3.1. Enter any of the messaging-activemq subsystems of the server instances that are running.

Configure the logging subsystem to increase the logging level by running the following command:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging\ /root-
logger=ROOT:write-attribute(name=level,value=INFO)
```

3.2. Verify that the topology attribute of the cluster-connection object shows the four server instances in the domain.

4. Create the JMS resources in the full-ha profile.

4.1. Create the custom JMS ConnectionFactory using java:/jms/CustomCF as the JNDI name.

4.2. Create the TestQueue JMS Queue using java:/jms/queue/JB248TestQueue as the JNDI name.

5. Deploy the consumer and producer applications.

5.1. Deploy the message producer application to Group1.

5.2. Deploy the message consumer application to Group1.

5.3. Verify that both implementations were successful.

6. Post a few copies of a test message and verify the TWO instances of the consumed messages server in the Group1 server group.

Also check the built-in messaging servers for messages received from servera.1 and serverb.1 server instances to their local queues.

6.1. Post a test message. Enter the message producer application running on servera.1 at http://172.25.250.10:8080/messaging-client and complete the form as follows:

• Message count: 5

• Message label: testmsg

• Message to send: jms test

6.2. Check server logs for entries generated by the application of the consumer. The servers servera.1 and serverb.1 must have log entries generated by the consumer application.

The Group2 server instances should NOT consume messages, as they do not have any consumer applications implemented.

6.3. Verify that messages were queued to different EAP server instances; that is, the volume of messages was shared by different instances of ActiveMQ servers. Only servera.1 and serverb.1 should have activity on their local queues.

Message servers integrated into Group2 servers should NOT queue any messages, as ActiveMQ defaults do not deliver messages to cluster members that do not have active consumers. This is done to avoid unnecessary network traffic.

7. Perform cleaning and grading.

7.1. Stop the slave hosts.

7.2. Stop the domain controller.

7.3. Run the following workstation command to grade this chapter review lab paper:

```
[student@workstation bin]$ lab messaging-lab grade
```

This concludes the lab work.

Solution

In this lab work, you will configure the JMS resources for a future version of the bookstore application. While the updated application is not available, test applications are provided to test that the JMS configuration is working as expected.

Resources	
Files:	/opt/domain
	/home/student/JB248/labs/messaging-lab
	/tmp/messaging-client.war
	/tmp/messaging-mdb.jar
App URL:	http://172.25.250.10:8080/messaging-client

Result

You must be able to create a JMS ConnectionFactory and Queue in a managed domain and deploy test applications that use them.

And Before You Get

Started You can choose either the EAP 7 Management Console or the JBoss EAP CLI to meet your goals, keeping in mind that the EAP CLI is the preferred option in production environments.

An EAP administrator has configured a managed domain with two host controllers running on the servera and serverb virtual machines, respectively, and the domain controller on the workstation virtual machine. The domain and host configuration files are stored in the /opt/domain folder on all three machines.

This lab work uses two applications:

- messaging-client.war is the message producer: a web application that accepts text from the user and publishes it as the JMS message body and properties.
- messaging-mdb.jar is the consumer of the message: an application with a Bean message-driven (MDB) that consumes JMS messages and displays their properties and body in the EAP server registry.

Both test applications use the following JMS resources:

- custom JMS ConnectionFactory, with JNDI address java:/jms/CustomCF. It must point to the ActiveMQ in-vm connector.
- TestQueue JMS Queue, with JNDI address java:/jms/queue/JB248TestQueue. There is no need to configure any message value for this queue.

Applications must be deployed on the server in group Group1, which uses the full-ha clustered profile. Fortunately, a sysadmin colleague gave him the artemis firewalld-rules.sh shell script that configures Linux kernel buffers and firewall rules to match the EAP7 defaults for this profile. The script is available at /home/student/JB248/labs/messaging-lab on the workstation virtual machine.

It is important to ensure that all application server instances in the Group1 server group are consuming messages. This means that message content processing is distributed across EAP hosts and the application is scalable.

Before beginning this chapter, review the lab work, run the following command to verify that EAP is installed to /opt/jboss-eap-7.0, that no EAP instances are running, and that the managed domain is configured, and to download the JEE application package files in the /tmp folder:

```
[student@workstation ~]$ lab messaging-lab setup
```

1. Configure the host OSes for EAP multicast clustering.

1.1. Download the artemis-firewalld-rules.sh shell script from workstation and run it on the servera virtual machine.

Open a terminal window on the workstation virtual machine to copy the script to the server machine:

```
[student@workstation ~]$ scp \ ~/JB248/
labs/messaging-lab/artemis-firewalld-rules.sh servera:~
```

Open another terminal window on the server machine to run the script using sudo:

```
[student@server ~]$ sudo sh artemis-firewalld-rules.sh
```

1.2. Inspect the artemis-firewalld-rules.sh shell script to verify the TCP and UDP ports it opens. Verify that the new firewall rules have been applied using the firewall-cmd command.

Verify that TCP ports 54200 and 54300, and UDP ports 45688, 55200, and 55300 are open:

```
[student@servera ~]$ firewall-cmd --list-all --zone=public
```

The expected result is similar to:

```
public (default, active) interfaces:
  eth0
  sources:
  services: dhcpv6-client ssh ports:
  54300/tcp 8180/tcp 55300/udp 8080/tcp 55200/udp 54200/tcp 45688/udp masquerade: no forward-
  ports: icmp-blocks:
  rich rules:
```

1.3. Verify that the new kernel values have been applied using the command sysctl.

Check that both net.core.rmem_max and net.core.wmem_max are set to 1 MiB (1073741824 bytes):

```
[student@servera ~]$ sysctl net.core.rmem_max net.core.wmem_max
```

The expected result is:

```
net.core.rmem_max = 1073741824
net.core.wmem_max = 1073741824
```

1.4. Repeat Step 1.1, Step 1.2, and Step 1.3 on the serverb virtual machine.

2. Start the EAP managed domain.

2.1. Start the domain controller in the workstation virtual machine using /opt/ domain as the base folder and host-master.xml as the host configuration file. All EAP processes must be owned by the jboss user.

Open a terminal window on the workstation virtual machine and run the following command:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \ -Djboss.domain.base.dir=/
opt/domain --host-config=host-master.xml
```

2.2. Start the host controller on the server virtual machine using /opt/domain as the base folder and host-slave.xml as the host configuration file. All EAP processes must be owned by the jboss user.

You must also provide the computer's IP address 172.25.250.10 as the argument to the -bprivate command line option. Without it, the EAP server instances will not meet to form a cluster.

Open a terminal window on the server virtual machine and run the following command:

```
[student@servera ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \ -Djboss.domain.base.dir=/
opt/domain --host-config=host-slave.xml \ -Djboss.domain.master.address=172.25.250.254
\ -bprivate=172.25.250.10
```

23. Start the host controller on serverb using the same parameters reported in the previous step for servera.

Remember to replace the IP address given to the -bprivate command line option with the 172.25.250.11 IP address of serverb.

Open a terminal window on the serverb virtual machine and run the following command:

```
[student@serverb ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/domain.sh \ -Djboss.domain.base.dir=/
opt/domain --host-config=host-slave.xml \ -Djboss.domain.master.address=172.25.250.254
\ -bprivate=172.25.250.11
```

2.4. **Verify that both host controllers connect to the domain controller and form a managed domain.**

Check the terminal window in which you started the domain controller to see if both servera and serverb are registered as slaves on the domain controller.

The domain controller registry should include entries similar to the following:

```
...
[Host Controller] 17:15:49,131 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads -
  34) WFLYHC0019: Registered remote slave host "servera", JBoss JBoss EAP 7.0.0.GA (WildFly
  2.1.2.Final-redhat-1)
...
[Host Controller] 17:16:04,294 INFO [org.jboss.as.domain.controller] (Host Controller Service Threads -
  35) WFLYHC0019: Registered remote slave host "serverb", JBoss JBoss EAP 7.0.0.GA (WildFly
  2.1.2.Final-redhat-1)
...
```

3. **Verify that the four server instances are connected to form an ActiveMQ cluster.**

The managed domain contains another server group with two more server instances. This group is also configured to use the full-ha profile, so it gets the same clustering parameters as Group1 and its ActiveMQ integrated server instances will also join the same messaging cluster.

3.1. **Enter any of the messaging-activemq subsystems of the server instances that are running.**

Open a terminal window on the workstation virtual machine and connect the CLI:

```
[student@workstation ~]$ sudo -u jboss /opt/jboss-eap-7.0/bin/jboss-cli.sh \
--connect --controller=172.25.250.254
```

Configure the logging subsystem to increase the logging level by running the following command:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=logging\ /root-
logger=ROOT:write-attribute(name=level,value=INFO)
```

Enter any of the messaging-activemq subsystems of the server instances that are running. In the following command, the server servera.1 was chosen from host servera:

```
[domain@172.25.250.254:9990 /] cd /host=servera/server=servera.1
[domain@172.25.250.254:9990 server=servera.1] cd subsystem=messaging-activemq
```

3.2. **Verify that the topology attribute of the cluster-connection object shows the four server instances in the domain.**

Use the following command: Remember that you cannot cd to a runtime resource, in this case server=default:

```
 [domain@172.25.250.254:9990 subsystem=messaging-activemq] ./server=default\ /cluster-
connection=my-cluster:read-attribute(name=topology)
```

The expected result is similar to:

```
 ...
        "result" => "topology on
  Topology@2f00ce83[owner=ClusterConnectionImpl@1089898802
 [nodeUUID=66010359-1888-11e6-bf74-bba07d6c198b,
  connector=TransportConfiguration(name=http-connector, factory=org-
  apache-activemq-artemis-core-remoting-impl-netty  NettyConnectorFactory) ?
httpUpgradeEnabled=true&httpPpgradeEndpoint=http  acceptor&port=8080&host=172-25-250-10,
address=jms, server=ActiveMQServerImpl::serverUUID=66010359-1888-11e6-
  bf74-bba07d6c198b]]: 771766a0-1888-11e6-a6cd-735504754596 => TopologyMember[ id =
  771766a0-1888-11e6-a6cd-735504754596,
connector=Pair[a=TransportConfiguration(name=http-
  connector, factory=org-apache-activemq-artemis-core-remoting-impl-netty
  NettyConnectorFactory) ?httpUpgradeEnabled=true&httpPpgradeEndpoint=http
acceptor&port=8180&host=172-25-250-11, b=null], backupGroupName=null,
scaleDownGroupName=null] 74943ed3-1888-11e6-80c1-ef9418706af0 => TopologyMember[ id
  = 74943ed3-1888-11e6-80c1-
  ef9418706af0, connector=Pair[a=TransportConfiguration(name=http-
connector, factory=org-apache-activemq-artemis-core-
  remoting-impl-netty  NettyConnectorFactory) ?
  httpUpgradeEnabled=true&httpPpgradeEndpoint=http
acceptor&port=8080&host=172-25-250-11, b=null], backupGroupName=null,
scaleDownGroupName=null] 684f38d4-1888-11e6-a0bc-6f1ea2c1a877 => TopologyMember[ id
  = 684f38d4-1888-11e6-
  a0bc-6f1ea2c1a877, connector=Pair[a=TransportConfiguration(name=http-
connector, factory=org-apache-activemq-artemis-core-
  remoting-impl-netty  NettyConnectorFactory) ?
  httpUpgradeEnabled=true&httpPpgradeEndpoint=http
acceptor&port=8180&host=172-25-250-10, b=null], backupGroupName=null,
scaleDownGroupName=null] 66010359-1888-11e6-bf74-bba07d6c198b => TopologyMember[ id
  = 66010359-1888-11e6-bf74-
  bba07d6c198b, connector=Pair[a=TransportConfiguration(name=http-
connector, factory=org-apache-activemq-artemis-core-
  remoting-impl-netty  NettyConnectorFactory) ?
  httpUpgradeEnabled=true&httpPpgradeEndpoint=http
acceptor&port=8080&host=172-25-250-10, b=null], backupGroupName=null,
scaleDownGroupName=null] nodes=4 members=4",


 ...
```

Notice that the TopologyMember resource has a different id and points to a different http-acceptor pair of port and host.

4. Create the JMS resources in the full-ha profile.

4.1. Create the custom JMS ConnectionFactory using java:/jms/CustomCF as the JNDI name.

Enter the messaging-activemq subsystem in the full-ha profile and create a pooled-connection-factory:

```
[domain@172.25.250.254:9990 /] cd /profile=full-ha
[domain@172.25.250.254:9990 profile=full-ha] cd \
subsystem=messaging-activemq/server=default
[domain@172.25.250.254:9990 server=default] ./pooled-connection-factory=\
custom:add(connectors=[in-vm], entries=[java:/jms/CustomCF])
```

4.2. Create the TestQueue JMS Queue using java:/jms/queue/JB248TestQueue as the JNDI name.

Use the CLI to create a jms-queue resource:

```
[domain@172.25.250.254:9990 server=default] ./jms-queue=TestQueue:add(\ entries=[java:/
jms/queue/JB248TestQueue])
```

5. Deploy the consumer and producer applications.

5.1. Deploy the message producer application to Group1.

Use the CLI to deploy the /tmp/messaging-client.war JEE package:

```
[domain@172.25.250.254:9990 server=default] deploy \ /tmp/
messaging-client.war --server-groups=Group1
```

5.2. Deploy the message consumer application to Group1.

Use the CLI to deploy the /tmp/messaging-mdb.jar JEE package:

```
[domain@172.25.250.254:9990 server=default] deploy \ /tmp/
messaging-mdb.jar --server-groups=Group1
```

5.3. Verify that both implementations were successful.

Check each slave host server log. They should have entries similar to the following:

```
...
[Server:servera.1] 18:41:55,697 INFO [org.jboss.as.server] (ServerService Thread Pool -- 108)
 WFLYSRV0010: Deployed "messaging-client.war" (runtime  name : "messaging-client.war")

...
[Server:servera.1] 18:44:50,425 INFO [org.jboss.as.server] (ServerService Thread Pool -- 113)
 WFLYSRV0010: Deployed "messaging-mdb.jar" (runtime-name : "messaging-mdb.jar")

...
```

6. Post a few copies of a test message and verify the TWO instances of the consumed messages server in the Group1 server group.

Also check the built-in messaging servers for messages received from servera.1 and serverb.1 server instances to their local queues.

6.1. Post a test message. Enter the message producer application running on servera.1 at http://172.25.250.10:8080/messaging-client and complete the form as follows:

- **Message count: 5**
- **Message label: testmsg**
- **Message to send: jms test**

Open a web browser and navigate to http://172.25.250.10:8080/messaging client.

Complete the web form and click Send Message to send the message.

The web form should update and show the following message in green: Messages sent successfully.

6.2. Check server logs for entries generated by the application of the consumer. The servers servera.1 and serverb.1 must have log entries generated by the consumer application.

The Group2 server instances should NOT consume messages, as they do not have any consumer applications implemented.

There should be registry entries similar to the following in the terminal window serve:

```
 [Server:servera.1] 19:54:56,317 INFO [class
   com.redhat.training.messaging.mdb.MessageReceiver] (Thread-11 (ActiveMQ-client  global-
threads-904575248)) Message Properties: Copy #3 [testmsg]
 [Server:servera.1] 19:54:56,318 INFO [class
   com.redhat.training.messaging.mdb.MessageReceiver] (Thread-12 (ActiveMQ-client  global-
threads-904575248)) Message Properties: Copy #5 [testmsg]
 [Server:servera.1] 19:54:56,318 INFO [class
   com.redhat.training.messaging.mdb.MessageReceiver] (Thread-11 (ActiveMQ-client  global-
threads-904575248)) Message Body: jms test [Server:servera.1]
 19:54:56,319 INFO [class
   com.redhat.training.messaging.mdb.MessageReceiver] (Thread-12 (ActiveMQ-client  global-
threads-904575248)) Message Body: jms test [Server:servera.1]
 19:54:56,322 INFO [class
   com.redhat.training.messaging.mdb.MessageReceiver] (Thread-10 (ActiveMQ-client  global-
threads-904575248)) Message Properties: Copy #1 [testmsg]
 [Server:servera.1] 19:54:56,323 INFO [class
   com.redhat.training.messaging.mdb.MessageReceiver] (Thread-10 (ActiveMQ-client  global-
threads-904575248)) Message Body: jms test
```

There should be log entries similar to the following in the serverb terminal window:

```
 [Server:serverb.1] 19:54:58,282 INFO [class
   com.redhat.training.messaging.mdb.MessageReceiver] (Thread-9 (ActiveMQ-client  global-
threads-1005879511)) Message Properties: Copy #4 [testmsg]
 [Server:serverb.1] 19:54:58,285 INFO [class
   com.redhat.training.messaging.mdb.MessageReceiver] (Thread-8 (ActiveMQ-client  global-
threads-1005879511)) Message Properties: Copy #2 [testmsg]
```

```
[Server:serverb.1] 19:54:58,285 INFO [class
  com.redhat.training.messaging.mdb.MessageReceiver] (Thread-9 (ActiveMQ-client  global-
threads-1005879511)) Message Body: jms test [Server:serverb.1]
 19:54:58,285 INFO [class
  com.redhat.training.messaging.mdb.MessageReceiver] (Thread-8 (ActiveMQ-client  global-
threads-1005879511)) Message Body: jms test
```

Remember that it is normal to have messages consumed out of order because EAP runs many MDB instances concurrently. It is also normal to have log entries from different messaging threads mixed up in the server log.

The distribution of messages between the servers you see may be different, but you should see that both server instances in Group1 (servera.1 and serverb.1) are receiving messages.

6.3. Verify that messages were queued to different EAP server instances; that is, the volume of messages was shared by different instances of ActiveMQ servers. Only servera.1 and serverb.1 should have activity on their local queues.

Message servers integrated into Group2 servers should NOT queue any messages, as ActiveMQ defaults do not deliver messages to cluster members that do not have active consumers. This is done to avoid unnecessary network traffic.

Return to the terminal window on the CLI-connected workstation virtual machine and check the message counters on servera.1:

```
[domain@172.25.250.254:9990 server=default] cd /host=servera\ /server=servera.1

[domain@172.25.250.254:9990 server=servera.1] cd \ ./
 subsystem=messaging-activemq
 [domain@172.25.250.254:9990 subsystem=messaging-activemq] ./server=default\ /jms-
queue=TestQueue:read-attribute(name=messages-added)
```

The expected result is similar to:

```
{
        "outcome" => "success",
        "result" => 3L
}
```

Check the message counters on serverb.1:

```
[domain@172.25.250.254:9990 server=default] cd /host=serverb\ /server=serverb.1

[domain@172.25.250.254:9990 server=serverb.1] cd \ ./
subsystem=messaging-activemq
[domain@172.25.250.254:9990 subsystem=messaging-activemq] ./server=default\ /jms-
queue=TestQueue:read-attribute(name=messages-added)
```

The expected result is similar to:

```
{
        "outcome" => "success", "result"
        => 2L
}
```

The attribute values you see must match the registry entries in the MDB. That is, the MDB will consume messages from your local ActiveMQ server.

7. Perform cleaning and grading.

7.1. Stop the slave hosts.

Run the following CLI command on each slave host in the managed domain:

```
[domain@172.25.250.254:9990 messaging-activemq] /host=servera:shutdown
[domain@172.25.250.254:9990 messaging-activemq] /host=serverb:shutdown
```

7.2. Stop the domain controller.

Run the following CLI command to stop the master host:

```
[domain@172.25.250.254:9990 /] /host=master:shutdown
```

7.3. Run the following workstation command to grade this chapter review lab paper:

```
[student@workstation bin]$ lab messaging-lab grade
```

This concludes the lab work.

Summary

In this chapter, you learned the following:

- **EAP 7 integrates the latest ActiveMQ message-driven middleware (MOM) based on the Artemis project. It is a high performance and availability MOM, suitable for highly demanding environments.**
- **The application server administrator configures JMS ConnectionFactory resources. and Destination using JNDI names expected by applications.**
  - ÿ **Generally, a PooledConnectionFactory object is preferred over a PooledConnectionFactory object. ConnectionFactory simple.**
  - ÿ **A JMS Destination can be a Queue (single consumer per message) or a Topic (publisher/subscriber with multiple consumers per message).**
- **Many MOM features, eg message redelivery and flow control, are configured within address-settings that can combine multiple destinations.**
- **The main component of ActiveMQ persistence is the message log files. They are tuned for production to avoid the need for file system metadata operations.**