Configuring newspapers/Journales and other values

Goals

After completing this section, students should be able to do the following:

• Configure messaging journals and other settings of the messaging subsystem.

Setting up messaging journals: NIO versus AIO

In JMS, the term "persistence" refers to the ability to guarantee delivery of messages to a topic or queue, even if the JMS provider fails. JMS implementations typically use a database to implement persistence. However, ActiveMQ takes the unique approach of using the file system, and the files that store the messages are known as journals.

ActiveMQ message persistence applies only to queues marked as durable and to all topics (due to subscriptions). Other data, such as the active subscriber list for each topic, is also stored on the file system. ActiveMQ maintains a few sets of files, but in this course, we only cover the messaging journal.

ActiveMQ journals are optimized to handle a large volume of messages quickly and reliably. The on-disk data structure of the journal is similar to the transaction logs used by databases like PostgreSQL and metadata in file systems like Linux ext4 and XFS.

Journals have the following characteristics:

• The journal consists of a set of files on your computer's file system.

• Each file is created with a fixed size and is filled with additional space for maximize disk access performance.

• Messages are attached to the journal. The set of journal files is read and written as a ring buffer, so the already consumed message space is reused by new published messages.

• When a journal file is full, ActiveMQ moves on to the next one. if they don't stay remaining empty journal files, ActiveMQ creates a new one.

• ActiveMQ also features a scavenging algorithm that reclaims space used by already consumed messages and eliminates fragmentation within messaging journals. This may leave some empty journal files eligible for deletion.

• The journal also fully supports transactional operations, if required. required, and supports local and XA transactions.

To maximize ActiveMQ performance on disk, avoid creating new journal files and running the compaction algorithm. When files must be created, expanded, or deleted,

the OS must perform additional I/O operations on the file system metadata.
By avoiding those operations, the OS can perform the minimum number of physical I/O operations to read and write application data.

To achieve these results, the administrator can configure the embedded MOM instance to server=default within the messaging-activemq subsystem with the ideal number of messaging journal files and the ideal size for each file. These numbers are obtained through capacity planning or load testing.

The following list shows all the attributes related to the messaging journal of the default EAP 7 configuration in managed domain mode:

```
[domain@127.0.0.1:9990 /] cd /profile=full/subsystem=messaging-activemq/server=default [domain@127.0.0.1:9990
server=default] :read-resource {

    "outcome" => "success",
    "result" => {
        ...
        "journal-buffer-size" => undefined, "journal-
        buffer-timeout" => undefined, "journal-compact-min-
        files" => 10, "journal-compact-percentage" =>
        30, "journal-file-size" => 10485760, "journal-
        max-io" => undefined, "journal-min-files"
        => 2, "journal-pool-files" => -1, "journal-
        sync-non-transactional" => true,
        "journal-sync-transactional" => true,
        "journal-type" => "ASYNCIO",


        ...
```

As the above list shows, the default messaging journal has two 10 MiB (journal-file-size) files (journal-min-files). All journal files are the same size, and each can store messages from multiple queues and durable topics.

If the administrator wants to switch to nine files of 50 MiB each, the following commands should be used, this time in standalone server mode:

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default
[standalone@localhost:9990 server=default] :write-attribute(\ name=journal-min-
files, value=9) [standalone@localhost:9990
server=default] :write-attribute(\ name=journal-file-size, value=52428800)
```

The main attributes of the journals can also be changed using the administration console.
In managed domain mode, go to Configuration > Profiles > full (or full-ha) > Messaging-ActiveMQ > default, to enter the Messaging Provider page. Then select Provider Settings to display the Provider Settings pop-up window, and click Journal for messaging journal attributes.

The following figure shows an example with default values, in managed domain mode. The popup window has been resized and expanded to display all the attributes displayed by the current section of the book.

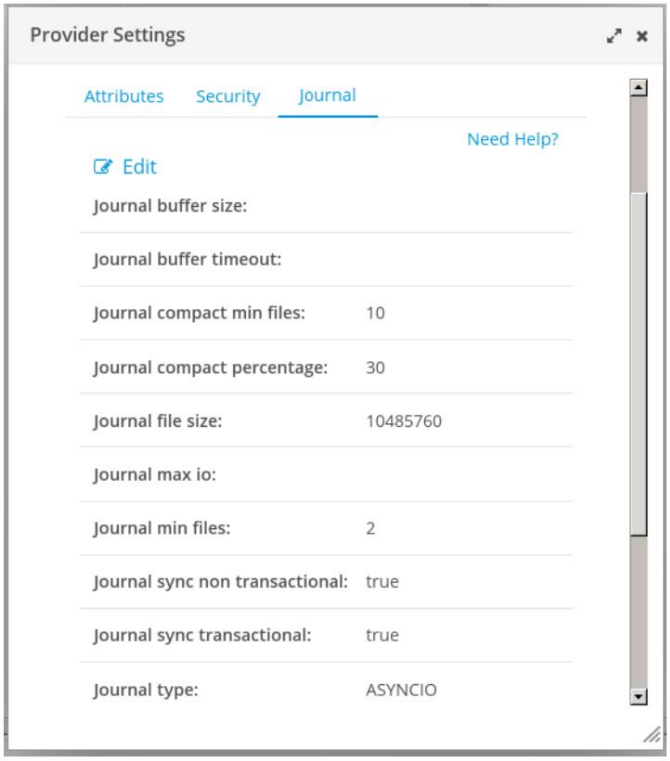

Figure 8.6:
EAP 7 JMS Provider window, messaging journal settings

This course does NOT cover any other tuning parameters related to ActiveMQ memory and disk usage, except for the important journal-type attribute. This attribute accepts two values:

- **NIO – Uses the standard Java NIO API to interact with the file system. This implementation provides extremely good performance and works on any platform with Java 6+ runtime. EAP 7 requires at least Java 8.**
- **ASYNCIO: Uses the native Linux asynchronous I/O (AIO) library. Normally using AIO will offer better performance than using Java NIO.**

The AIO library requires Linux kernel 2.6 or later and the *libaio* RPM package. Both are installed by default on Red Hat Enterprise Linux 7. For other Linux distributions, check with the OS vendor.

The default value for journal-type is ASYNCIO, but if the native AIO library cannot be loaded, ActiveMQ falls back to NIO. In this way, EAP 7's built-in messaging uses the best available alternative and there is no need to modify the journal-type attribute for most installations.

If the administrator wants to make sure that EAP 7 is using the AIO library, check that the server instance logs for a message similar to the following:

```
2016-05-23 15:28:48,994 INFO [org.apache.activemq.artemis.core.server] (ServerService Thread Pool -- 68)
  AMQ221012: Using AIO Journal
```

Other messaging system values

As shown earlier in this chapter, ActiveMQ destinations have very few attributes. But there are many settings that the administrator expects to be able to configure in a production level MOM. For example: security, flow control, redelivery, and so on.

ActiveMQ places those values outside of the targets, in dedicated configuration items, since most real-world applications require many targets with similar values. This is done to ease the administrator's workload and reduce the risk of having inconsistent values for targets in the same application.

There are two configuration items: address-settings and security-settings. The former contains most of the values, while the latter contains only access control attributes.

The names of both elements depend on the syntax of a specific wildcard to match a target name with the name of the element. The wildcard metacharacters are:

- # (hash) – Matches all. For example: the <address-setting name="#"> element matches any destination name, be it a queue or a topic. Another example: stock.us.# matches stock.us.rht and stock.us.nasdaq.rht, but not stock.emea.xyz.

- . (single dot) - Matches the space between words in a wildcard expression. Most of the time, it can be interpreted as a literal point.

- * (asterisk): coincides with a single word. Por ejemplo: stock.us.* coincides with stock.us.rht, but not with stock.us.nasdaq.rht. Another example: stock.*.rht coincides with stock.us.rht and stock.emea.rht, but not with stock.us.nasdaq.rht.

For address-settings and security-settings to match a single target name, use the full target name as the element name.

Note that the destination name is NOT the JNDI name. This is the same name used internally by ActiveMQ to refer to the destination. For an ActiveMQ integrated within EAP 7, the internal name is the name of the object prefixed with jms.queue. for a jms-queue resource or the jms.topic prefix. for a jms-topic resource. Note the period at the end of the two prefixes.

A jms-topic also gets a suffix based on the subscriber ID, so an address-setting that refers to jms-topic should always add the # suffix.

The following table shows the name of a target resource and the name for an address-setting that matches that target specifically:

destination name of ActiveMQ	ActiveMQ address value name
jms-queue=MyQueue	address-settings=jms.queue.MyQueue
jms-topic=MyTopic	address-settings=jms.topic.MyTopic.#

An address-setting can configure a number of different functions; see the EAP product documentation for a description and usage examples for

each of its attributes. The following examples are included for syntax examples only and use some of the attributes related to control flow.

- The following example matches only the queue named jms.queue.Volatile and limits memory usage for all messages to 100 KiB. If more messages are published that increase memory usage above the threshold, the new messages are silently deleted:

```
[domain@localhost:9990 /] cd /profile=full/subsystem=messaging-activemq/server=default [domain@localhost:9990
server=default] ./address-setting=jms.queue.Volatile:add(\ max-size-bytes=102400, address-full-
policy=DROP)
```

- The following example matches all sensor application targets (all follow a naming convention prefixed with app.sensors) and limits memory usage for each target to 150 KiB. But instead of deleting messages, publishers are prevented from publishing any more messages until some messages are consumed and memory is freed:

```
[domain@localhost:9990 /] cd /profile=full/subsystem=messaging-activemq/server=default [domain@localhost:9990
server=default] ./address-setting=jms.queue.app.sensors.*:add(\ max-size-bytes=153600, address-full-
policy=BLOCK)
```

Two important caveats when configuring ActiveMQ address-settings are:

1. Many independent functions are configured by the same object. When creating an address-setting object focused on a single function, as in the previous examples, attributes related to other functions get their default values. This can have unexpected side effects, such as a feature enabled for all targets in the EAP defaults, being disabled for some targets.

2. Multiple address-settings objects can match the same destination. The more specific matches usually supersede less specific ones. Test carefully if the merged configuration has the expected behavior.

The administration console provides a page for creating and configuring address setting objects along with the Queues/Topics page and is illustrated by the following figure:

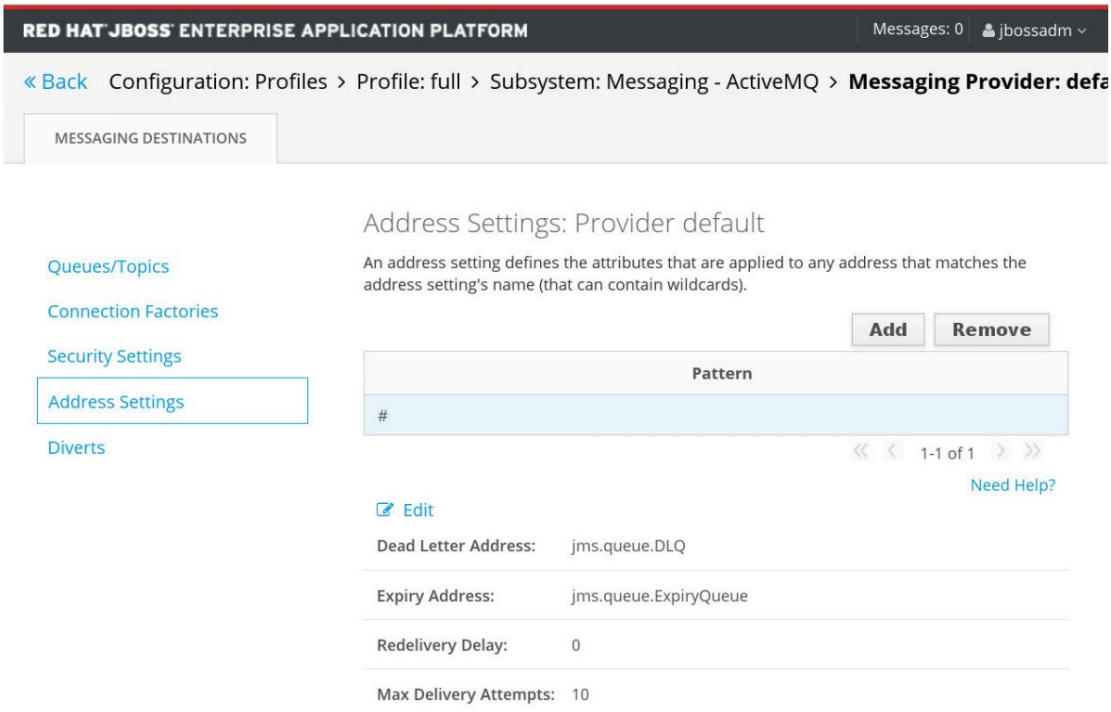

Figure
8.7: EAP 7 Messaging Provider - Address Values Page

messaging security

Messaging security in EAP 7 addresses the built-in MOM, that is, ActiveMQ and the API of Etc.:

- **MOM settings define how to authenticate local and remote clients, and what permissions grant those customers for different destinations.**
- **The JMS API focuses on how to pass client credentials to the MOM.**

Security for the embedded MOM is configured as attributes for the server=default object within the messaging-activemq subsystem. The security domain attribute selects the EAP security domain to use to authenticate client user credentials and obtain their roles.

The default configuration points to the other security domain, which validates user credentials using the ApplicationRealm configured in the standalone.xml or host.xml file. For more information about configuring users and roles in ApplicationRealm, see the Securing JBoss EAP chapter later in this book.

Security is NOT used for applications connecting to the built-in ActiveMQ, as the defaults have the override-in-vm-security attribute set to true. Unless this attribute is set to false, only remote clients are required to provide authentication credentials and are subject to authorization checks.

Most production environments require changing the default messaging-activemq subsystem configurations to use a security domain

different, backed by a database or an LDAP server. They probably also require local client authentication to be enabled.

To configure permissions for messaging destinations, the administrator configures security-setting objects. These objects work in much the same way as address-setting objects by using their names as wildcards to match internal ActiveMQ destination names.

A security-setting has child objects of type role, whose attributes define the permissions that the named role has. The standard EAP configuration defines a single security-setting=# that contains a single role=guest child element that allows any user to publish and consume messages to any destination.

The following command inspects the default security-setting to display the roles specified by the object:

```
[standalone@localhost:9990 /] cd /subsystem=messaging-activemq/server=default
[standalone@localhost:9990 server=default] ./security-setting=#:read-resource {

        "outcome" => "success",
        "result" => {"role" => {"guest" => undefined}}
}
```

The output above shows that only the guest role was assigned permissions via the default security-setting.

To display the permissions assigned to the role, inspect the role=guest child element, as in the following example:

```
[standalone@localhost:9990 server=default] ./security-setting=#/role=guest:read-resource {

        "outcome" => "success",
        "result" =>
                { "consume" => true,
                "create-durable-queue" => false, "create-
                non-durable-queue" => true, "delete-durable-
                queue" => false, "delete-non-durable-queue"
                => true, "manage" => false, "send" => true


        }
}
```

An alternative is to pass the recursive=true argument to the read-resource operation in security-setting:

```
[standalone@localhost:9990 server=default] ./security-setting=#:read-resource(\ recursive=true) {


        "outcome" => "success",
        "result" => {"role" => {"guest" => { "consume" =>
                true, "create-durable-
                queue" => false, "create-non-durable-queue"
                => true, "delete-durable-queue" => false, "delete-
                non-durable-queue" => true, "manage" =>
                false,
```

```
                "send" => true
        }}}
}
```

Note that a single security-setting object can grant permissions for many roles, and multiple security-setting objects can match the same target. Unlike the address-setting object, where multiple matches can affect the same destination, the more specific security-setting object is used alone. In this way, a more specific security-setting object can deny permissions granted by a more generic one.

Production environments likely require removal of the default permissions assigned to the guest role to implement the secure by default philosophy. The following command does it:

```
[standalone@localhost:9990 server=default] ./security-setting=#/role=guest:remove
```

The following example shows how to add permissions to a specific role:

```
[standalone@localhost:9990 server=default] ./security-setting=#/role=sender:add(\ send=true)
```

The example above shows how to add the sender role permission to post messages to any destination, and nothing else.

Specific permissions, such as send and consume, and their meanings, are introduced later in this course in the Securing JBoss EAP chapter.

Handling message delivery failures

An important aspect of MOM administration is handling failures to consume messages. Delivery failures can affect performance, as a consumer can get stuck in a loop, trying to consume the same message over and over again, and throwing errors for each attempt to process the message contents.

This scenario can occur because messages not recognized as consumed are NOT discarded; messages are left at the destination, to be consumed later. This is similar to a database rolling back a failed transaction to ensure data consistency; the message read operation is rolled back because its included transaction was aborted.

Most of the time, this is the right thing to do as the failure may be related to something that should be working fine, such as an offline database server, and deleting the message may result in the loss of important business transactions, such as bill payments. an order. Trying again later allows the application to recover from unexpected crashes.

However, if the failure occurs only with some messages and not with others, it is recommended to delete the failed messages. This saves hardware resources for processing messages that were not affected by current problems and allows you to deal with failed messages later, after the problems have been fixed, using an exception workflow. .

For example, an application that processes payments for an online store might accept three payment options: credit card, bank transfer, and gift card. To process credit card payments, an external web service must be called. Sometimes this service experiences technical difficulties, but other payment options are not affected and do not need to wait until the credit card service comes back online.

ActiveMQ address-setting objects allow the administrator to define:

- **A delay in resending a failed message, hoping that the problem will not occur again on the next attempt, using the redelivery-delay attribute.**
- **A destination to send the message after a number of failed delivery attempts, usando los atributos max-delivery-attempts y dead-letter-address.**

The default messaging-activemq values include the following address value:

```
[standalone@localhost:9990 server=default] ./address-setting=#:read-resource {

    "outcome" => "success",
    "result" => {
        ...
        "dead-letter-address" => "jms.queue.DLQ",
        ...
        "max-delivery-attempts" => 10,
        ...
        "redelivery-delay" => 0L,
        ...
```

This means that for all destinations (name of address-setting object is #), messages that fail to be consumed after 10 attempts (max-delivery-attempts) are passed to destination jms.queue.DLQ (dead-letter -address). There are no delays between delivery attempts (redelivery-delay).

Important

Remember one important caveat: the destination names used to match an address-setting element and for the dead-letter address attribute are not JNDI names as perceived by the JEE application. They are also not the name of the destination resources, as perceived by the CLI. They are internal ActiveMQ destination addresses.

The rules for generating ActiveMQ internal names were presented earlier in this chapter. Another way to get an internal destination name is by using the queue-address runtime attribute of the jms-queue or jms-topic objects.

Sorting messages from many applications onto the same default dead-letter queue (DLQ) is probably not easy, and most applications require a custom address-setting object that specifies a different forwarding policy.

For example, to have a 5 second delay between delivery attempts to destination jms queue=MyQueue and move messages to jms-queue=MyDLQ after three attempts, use the following command:

```
[standalone@localhost:9990 server=default] ./address-setting=jms.queue.MyQueue:add(\ redelivery-delay=5000,
dead-letter-address=jms.queue.MyDLQ, max-delivery-attempts=3)
```

A concept closely related to DLQ is the concept of expiration. It means that some messages may not be useful, or may require processing by different business rules, if not consumed within a certain time frame.

ActiveMQ allows an administrator to set up an expiration queue using the expiry-address attribute of the address-setting object. The JMS API specifies how an application can provide a message expiration interval. An administrator can also use the expiry-delay attribute to define a default interval for messages to expire. This default interval does NOT override the one specified by the application.

For example, to have a default expiration interval of 20 minutes (20 minutes x 60 seconds x 1000 ms = 1200000ms) for messages from destination jms queue=MyQueue and move messages that took longer to consume to jms queue=MyExpiry, use the following command:

```
[standalone@localhost:9990 server=default] ./address-setting=jms.queue.MyQueue:add(\ expiry-delay=1200000,
expiry-address=jms.queue.MyExpiry)
```

If the redelivery and expiration options of the previous two examples are required, a single address-setting object must be created by merging all the attributes: address-setting names must be unique, even if multiple address-setting object names can match the same internal destination address.

It was already shown in this chapter that address-setting objects can be configured using the administration console, and thus, redelivery and expiration values can be configured using the web administration interface instead of the CLI.