Red Hat AMQ Broker 7.12

AMQ Broker 구성

AMQ Broker 7.12와 함께 사용하는 경우

Last Updated: 2025-02-02

Red Hat AMQ Broker 7.12 AMQ Broker 구성

AMQ Broker 7.12와 함께 사용하는 경우

법적 공지

초록

이 가이드에서는 AMQ Broker 구성 방법을 설명합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

1장. 개요

AMQ Broker 구성 파일은 브로커 인스턴스에 대한 중요한 설정을 정의합니다. 브로커의 구성 파일을 편집하여 브로커가 사용자 환경에서 작동하는 방식을 제어할 수 있습니다.

1.1. AMQ BROKER 구성 파일 및 위치

브로커의 모든 구성 파일은 < ***broker_instance_dir> /etc*** 에 저장됩니다. 이러한 구성 파일에서 설정을 편집하여 브로커를 구성할 수 있습니다.

각 브로커 인스턴스는 다음 구성 파일을 사용합니다.

broker.xml

기본 구성 파일입니다. 이 파일을 사용하여 네트워크 연결, 보안 설정, 메시지 주소 등과 같은 브로커의 대부분의 측면을 구성합니다.

bootstrap.xml

AMQ Broker가 브로커 인스턴스를 시작하는 데 사용하는 파일입니다. 이를 사용하여 **broker.xml** 의 위치를 변경하고, 웹 서버를 구성하고, 일부 보안 설정을 설정합니다.

logging.properties

이 파일을 사용하여 브로커 인스턴스의 로깅 속성을 설정합니다.

artemis.profile

이 파일을 사용하여 브로커 인스턴스가 실행되는 동안 사용되는 환경 변수를 설정합니다.

login.config, artemis-users.properties, artemis-roles.properties

보안 관련 파일 이러한 파일을 사용하여 브로커 인스턴스에 대한 사용자 액세스에 대한 인증을 설정합니다.

1.2. 기본 브로커 구성 이해

broker.xml 구성 파일을 편집하여 브로커의 대부분의 기능을 구성합니다. 이 파일에는 브로커를 시작하고 운영하기에 충분한 기본 설정이 포함되어 있습니다. 그러나 일부 기본 설정을 변경하고 사용자 환경에 브로커를 구성하려면 새 설정을 추가해야 할 수도 있습니다.

기본적으로 **broker.xml** 에는 다음 기능에 대한 기본 설정이 포함되어 있습니다.

- 메시지 지속성
- 어셉터
- 보안
- 메시지 주소

기본 메시지 지속성 설정
기본적으로 AMQ Broker 지속성은 디스크의 파일 세트로 구성된 추가 전용 파일 저널을 사용합니다. 저널은 메시지, 트랜잭션 및 기타 정보를 저장합니다.

```
<configuration ...>

  <core ...>

    ...
```

```
      <persistence-enabled>true</persistence-enabled>

      <!-- this could be ASYNCIO, MAPPED, NIO
           ASYNCIO: Linux Libaio
           MAPPED: mmap files
           NIO: Plain Java Files
       -->
      <journal-type>ASYNCIO</journal-type>

      <paging-directory>data/paging</paging-directory>

      <bindings-directory>data/bindings</bindings-directory>

      <journal-directory>data/journal</journal-directory>

      <large-messages-directory>data/large-messages</large-messages-directory>

      <journal-datasync>true</journal-datasync>

      <journal-min-files>2</journal-min-files>

      <journal-pool-files>10</journal-pool-files>

      <journal-file-size>10M</journal-file-size>

      <!--
       This value was determined through a calculation.
       Your system could perform 8.62 writes per millisecond
       on the current journal configuration.
       That translates as a sync write every 115999 nanoseconds.

       Note: If you specify 0 the system will perform writes directly to the disk.
            We recommend this to be 0 if you are using journalType=MAPPED and journal-
datasync=false.
      -->
      <journal-buffer-timeout>115999</journal-buffer-timeout>

      <!--
        When using ASYNCIO, this will determine the writing queue depth for libaio.
       -->
      <journal-max-io>4096</journal-max-io>

      <!-- how often we are looking for how many bytes are being used on the disk in ms -->
      <disk-scan-period>5000</disk-scan-period>

      <!-- once the disk hits this limit the system will block, or close the connection in certain protocols
           that won't support flow control. -->
      <max-disk-usage>90</max-disk-usage>

      <!-- should the broker detect dead locks and other issues -->
      <critical-analyzer>true</critical-analyzer>

      <critical-analyzer-timeout>120000</critical-analyzer-timeout>

      <critical-analyzer-check-period>60000</critical-analyzer-check-period>
```

```
      <critical-analyzer-policy>HALT</critical-analyzer-policy>

      ...

   </core>

</configuration>
```

기본 수락 설정

브로커는 클라이언트에서 사용할 수 있는 포트 및 프로토콜을 정의하는 데 **acceptor** 구성 요소를 사용하여 들어오는 클라이언트 연결을 수신합니다. 기본적으로 AMQ Broker에는 다음과 같이 지원되는 각 메시징 프로토콜에 대한 어셉터가 포함되어 있습니다.

```
<configuration ...>

  <core ...>

    ...

    <acceptors>

      <!-- Acceptor for every supported protocol -->
      <acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,STOMP,HORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- AMQP Acceptor. Listens on default AMQP port for AMQP traffic -->
      <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- STOMP Acceptor -->
      <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEpoll=true</acceptor>

      <!-- HornetQ Compatibility Acceptor. Enables HornetQ Core and STOMP for legacy HornetQ clients. -->
      <acceptor name="hornetq">tcp://0.0.0.0:5445?
anycastPrefix=jms.queue.;multicastPrefix=jms.topic.;protocols=HORNETQ,STOMP;useEpoll=true</acceptor>

      <!-- MQTT Acceptor -->
      <acceptor name="mqtt">tcp://0.0.0.0:1883?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=MQTT;useEpoll=true</acceptor>

    </acceptors>

    ...

  </core>

</configuration>
```

기본 보안 설정
AMQ Broker에는 주소를 기반으로 큐에 보안을 적용하기 위한 유연한 역할 기반 보안 모델이 포함되어 있습니다. 기본 구성에서는 와일드카드를 사용하여 모든 주소에 **amq** 역할을 적용합니다(숫자 기호 **#**로 표시됨).

```
<configuration ...>

  <core ...>

    ...

    <security-settings>
      <security-setting match="#">
        <permission type="createNonDurableQueue" roles="amq"/>
        <permission type="deleteNonDurableQueue" roles="amq"/>
        <permission type="createDurableQueue" roles="amq"/>
        <permission type="deleteDurableQueue" roles="amq"/>
        <permission type="createAddress" roles="amq"/>
        <permission type="deleteAddress" roles="amq"/>
        <permission type="consume" roles="amq"/>
        <permission type="browse" roles="amq"/>
        <permission type="send" roles="amq"/>
        <!-- we need this otherwise ./artemis data imp wouldn't work -->
        <permission type="manage" roles="amq"/>
      </security-setting>
    </security-settings>

    ...

  </core>

</configuration>
```

기본 메시지 주소 설정
AMQ Broker에는 생성된 모든 큐 또는 항목에 적용할 기본 구성 설정 세트를 설정하는 기본 주소가 포함되어 있습니다.

또한 기본 구성은 두 개의 대기열을 정의합니다. **DLQ** (Dead Letter Queue)는 알려진 대상 없이 도착한 메시지를 처리하고 **Expiry Queue** 는 만료가 지난 메시지를 보관하므로 원래 대상으로 라우팅해서는 안 됩니다.

```
<configuration ...>

  <core ...>

    ...

    <address-settings>
      ...
      <!--default for catch all-->
      <address-setting match="#">
        <dead-letter-address>DLQ</dead-letter-address>
        <expiry-address>ExpiryQueue</expiry-address>
        <redelivery-delay>0</redelivery-delay>
        <!-- with -1 only the global-max-size is in use for limiting -->
```

```
        <max-size-bytes>-1</max-size-bytes>
        <message-counter-history-day-limit>10</message-counter-history-day-limit>
        <address-full-policy>PAGE</address-full-policy>
        <auto-create-queues>true</auto-create-queues>
        <auto-create-addresses>true</auto-create-addresses>
        <auto-create-jms-queues>true</auto-create-jms-queues>
        <auto-create-jms-topics>true</auto-create-jms-topics>
      </address-setting>
    </address-settings>

    <addresses>
      <address name="DLQ">
        <anycast>
          <queue name="DLQ" />
        </anycast>
      </address>
      <address name="ExpiryQueue">
        <anycast>
          <queue name="ExpiryQueue" />
        </anycast>
      </address>
    </addresses>

  </core>

</configuration>
```

1.3. 구성 업데이트 다시 로드

기본적으로 브로커는 5000 밀리초마다 구성 파일의 변경 사항을 확인합니다. 브로커가 구성 파일의 "마지막 수정" 타임 스탬프의 변경을 감지하면 브로커는 구성 변경이 수행되었다고 결정합니다. 이 경우 브로커는 구성 파일을 다시 로드하여 변경 사항을 활성화합니다.

브로커가 **broker.xml** 구성 파일을 다시 로드하면 다음 모듈을 다시 로드합니다.

- 주소 설정 및 대기열
  구성 파일이 다시 로드되면 주소 설정에 따라 구성 파일에서 삭제된 주소와 큐를 처리하는 방법이 결정됩니다. **config-delete-addresses** 및 **config-delete-queues** 속성을 사용하여 이 값을 설정할 수 있습니다. 자세한 내용은 부록 B. *주소 설정 구성*의 내용을 참조하십시오.

- 보안 설정
  기존 승인자의 SSL/TLS 키 저장소 및 신뢰 저장소를 다시 로드하여 기존 클라이언트에 영향을 주지 않고 새 인증서를 설정할 수 있습니다. 이전 인증서 또는 다른 인증서가 있는 경우에도 연결된 클라이언트는 계속 메시지를 보내고 받을 수 있습니다.

crlPath 매개변수를 사용하여 구성된 인증서 취소 목록 파일도 다시 로드할 수 있습니다.

- 다이버
  구성 다시 로드는 사용자가 추가한 **새로운** 차이점을 배포합니다. 그러나 구성에서 다양한 제거 또는 < **divert** > 요소 내의 하위 요소에 대한 변경 사항은 브로커를 다시 시작할 때까지 적용되지 않습니다.

다음 절차에서는 브로커가 **broker.xml** 구성 파일의 변경 사항을 확인하는 간격을 변경하는 방법을 보여줍니다.

프로세스

1. &lt ***;broker_instance_dir>*** **/etc/broker.xml** 구성 파일을 엽니다.

2. < **core** > 요소 내에서 < **configuration-file-refresh-period** > 요소를 추가하고 새로 고침 기간(밀리초)을 설정합니다.
   이 예에서는 구성 새로 고침 기간을 60000밀리초로 설정합니다.

```
<configuration>
  <core>
    ...
    <configuration-file-refresh-period>60000</configuration-file-refresh-period>
    ...
  </core>
</configuration>
```

어떤 이유로든 구성 파일에 대한 액세스 권한이 불가능한 경우 Management API 또는 콘솔을 사용하여 구성 파일을 강제로 다시 로드할 수도 있습니다. **ActiveMQServerControl** 의 관리 작업 **reloadConfigurationFile()** 을 사용하여 구성 파일을 다시 로드할 수 있습니다(**ObjectName org.apache.activemq.artemis:broker="*BROKER_NAME*"** 또는 리소스 이름 **서버**)

추가 리소스

- 관리 API 사용 방법을 알아보려면 AMQ Broker 관리에서 관리 API 사용을 참조하십시오.

1.4. 브로커 구성 파일 모듈화

공통 구성 설정을 공유하는 브로커가 여러 개 있는 경우 별도의 파일에 공통 구성을 정의한 다음 각 브로커의 **broker.xml** 구성 파일에 이러한 파일을 포함할 수 있습니다.

브로커 간에 공유할 수 있는 가장 일반적인 구성 설정은 다음과 같습니다.

- 주소
- 주소 설정
- 보안 설정

프로세스

1. 공유하려는 각 **broker.xml** 섹션에 대해 별도의 XML 파일을 생성합니다.
   각 XML 파일은 **broker.xml** 의 단일 섹션(예: 주소 또는 주소 설정 중 하나)만 포함할 수 있습니다.
   최상위 요소도 요소 네임스페이스(**xmlns="urn:activemq:core**")를 정의해야 합니다.

   이 예제에서는 **my-security-settings.xml**:에 정의된 보안 설정 구성을 보여줍니다.

   my-security-settings.xml

```
<security-settings xmlns="urn:activemq:core">
  <security-setting match="a1">
    <permission type="createNonDurableQueue" roles="a1.1"/>
  </security-setting>
  <security-setting match="a2">
```

```
      <permission type="deleteNonDurableQueue" roles="a2.1"/>
    </security-setting>
  </security-settings>
```

2. 공통 구성 설정을 사용해야 하는 각 브로커에 대해 < ***broker_instance_dir***> **/etc/broker.xml** 구성 파일을 엽니다.

3. 열린 각 **broker.xml** 파일에 대해 다음을 수행합니다.

   a. **broker.xml** 시작 시 <configuration> 요소에서 다음 행이 표시되는지 확인합니다.

   ```
   xmlns:xi="http://www.w3.org/2001/XInclude"
   ```

   b. **공유 구성 설정이 포함된 각 XML 파일에 대한 XML 포함을 추가합니다.**

   이 예제에는 my-security-settings.xml 파일이 포함되어 있습니다.

   broker.xml

   ```
   <configuration ...>
     <core ...>
       ...
       <xi:include href="/opt/my-broker-config/my-security-settings.xml"/>
       ...
     </core>
   </configuration>
   ```

   c. **필요한 경우 broker.xml 의 유효성을 검사하고 XML이 스키마에 대해 유효한지 확인합니다.**

   모든 XML 검증기 프로그램을 사용할 수 있습니다. 이 예에서는 xmllint 를 사용하여 artemis-server.xsl 스키마에 대해 broker.xml 의 유효성을 검사합니다.

   ```
   $ xmllint --noout --xinclude --schema /opt/redhat/amq-broker/amq-broker-
   7.2.0/schema/artemis-server.xsd /var/opt/amq-broker/mybroker/etc/broker.xml
   /var/opt/amq-broker/mybroker/etc/broker.xml validates
   ```

추가 리소스

- **XML 포함(XIncludes)에 대한 자세한 내용은 https://www.w3.org/TR/xinclude/ 을 참조하십**

시오.

1.4.1. 모듈식 구성 파일 다시 로드

브로커가 구성 변경 사항을 주기적으로 확인할 때(구성-file-refresh-period에 지정된 빈도에 따라) xi:include 를 통해 broker.xml 구성 파일에 포함된 구성 파일에 대한 변경 사항을 자동으로 감지 하지 않습니다. 예를 들어 broker.xml 에 my-address-settings.xml 이 포함되어 있고 my-address-settings.xml 에 대한 구성을 변경한 경우 브로커는 my-address-settings.xml 의 변경 사항을 자동으로 탐지하지 않고 구성을 다시 로드합니다.

broker.xml 구성 파일과 그 내에 포함된 수정된 구성 파일을 *강제로* 다시 로드하려면 broker.xml 구성 파일의 "마지막 수정" 타임 스탬프가 변경되었는지 확인해야 합니다. 표준 Linux touch 명령을 사용하여 다른 변경 없이 broker.xml 의 마지막 수정 타임스탬프를 업데이트할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ touch -m <broker_instance_dir>/etc/broker.xml
```

또는 관리 API를 사용하여 브로커를 강제로 다시 로드할 수 있습니다. ActiveMQServerControl 의 관리 작업 reloadConfigurationFile() 을 사용하여 구성 파일을 다시 로드할 수 있습니다(ObjectName org.apache.activemq.artemis:broker="*BROKER_NAME*" 또는 리소스 이름 서버)

추가 리소스

- 관리 API 사용 방법을 알아보려면 AMQ Broker 관리에서 관리 API 사용을 참조하십시오.

1.4.2. 외부 XML 엔티티(XXE) 처리 비활성화

broker.xml 파일에 포함된 별도의 파일에서 브로커 구성을 모듈화하지 않으려면 XXE 처리를 비활성화하여 AMQ Broker를 보안 취약점으로부터 보호할 수 있습니다. 모듈식 브로커 구성이 없는 경우 XXE 처리를 비활성화하는 것이 좋습니다.

프로세스

1. &lt ;broker_instance_dir>/etc/artemis.profile 파일을 엽니다.

2. -Dartemis.disableXxe 를 Java 시스템 인수의 JAVA_ARGS 목록에 추가합니다.

```
-Dartemis.disableXxe=true
```

3. Artemis .profile 파일을 저장합니다.

1.5. JAVA CLASSPATH 확장

기본적으로 < *broker_instance_dir* >의 JAR 파일은 디렉터리가 Java classpath의 일부이므로 런타임 시 로드됩니다. AMQ Broker가 < *broker_instance_dir*> /lib 이외의 디렉터리에서 JAR 파일을 로드하려면 해당 디렉터리를 Java 클래스 경로에 추가해야 합니다.

Java 클래스 경로에 디렉터리를 추가하려면 다음 방법 중 하나를 사용할 수 있습니다.

- < broker_instance_dir>/etc/artemis.profile 파일에서 시스템 속성의 JAVA_ARGS 목록에 새로운 속성 artemis.extra.libs 를 추가합니다.
- ARTEMIS_EXTRA_LIBS 환경 변수를 설정합니다.

다음은 두 가지 방법을 사용하여 Java Classpath에 추가된 쉼표로 구분된 디렉터리 목록의 예입니다.

```
-Dartemis.extra.libs=/usr/local/share/java/lib1,/usr/local/share/java/lib2
```

```
export ARTEMIS_EXTRA_LIBS=/usr/local/share/java/lib1,/usr/local/share/java/lib2
```

참고

Artemis .extra.libs Java 시스템 속성이 < broker_instance_dir>/etc/artemis. profile 파일에 구성된 경우 ARTEMIS_ EXTRA_LIBS 환경 변수는 무시됩니다.

1.6. 문서 규칙

이 문서에서는 sudo 명령, 파일 경로 및 교체 가능한 값에 대해 다음 규칙을 사용합니다.

sudo 명령

이 문서에서 sudo 는 root 권한이 필요한 모든 명령에 사용됩니다. sudo 를 사용할 때는 변경 사항이 전체 시스템에 영향을 미칠 수 있으므로 주의해야 합니다.

sudo 사용에 대한 자세한 내용은 sudo 액세스 관리를 참조하십시오.

이 문서에서 파일 경로 사용 정보

이 문서에서 모든 파일 경로는 Linux, UNIX 및 유사한 운영 체제(예: /home/...)에 유효합니다. Microsoft Windows를 사용하는 경우 동등한 Microsoft Windows 경로(예: C:\Users\...)를 사용해야 합니다.

대체 가능한 값

이 문서에서는 사용자 환경과 관련된 값으로 교체해야 하는 교체 가능한 값을 사용하는 경우가 있습니다. 대체 가능한 값은 소문자로 묶고(< >)로 묶고 italics 및 monospace 글꼴 을 사용하여 스타일을 지정할 수 있습니다. 여러 단어가 밑줄(_)으로 구분됩니다.

예를 들어 다음 명령에서 < *install_dir*> 을 고유한 디렉터리 이름으로 바꿉니다.

```
$ <install_dir>/bin/artemis create mybroker
```

2장. 네트워크 연결에서 어셉터 및 커넥터 구성

AMQ Broker에는 네트워크 연결 및 *VM 내 연결의 두 가지 유형의 연결이 있습니다*. 네트워크 연결은 두 당사자가 동일한 서버 또는 물리적 원격지 여부에 관계없이 서로 다른 가상 머신에 있는 경우 사용됩니다. VM 내 연결은 애플리케이션 또는 서버 등 클라이언트가 브로커와 동일한 가상 머신에 상주할 때 사용됩니다.

네트워크 연결은 Netty 를 사용합니다. Netty는 Java IO 또는 NIO, TCP 소켓, SSL/TLS 또는 HTTP 또는 HTTPS를 통한 터널링 등 다양한 방식으로 네트워크 연결을 구성할 수 있는 고성능의 하위 수준 네트워크 라이브러리입니다. Netty를 사용하면 모든 메시징 프로토콜에 단일 포트를 사용할 수 있습니다. 브로커는 사용 중인 프로토콜을 자동으로 감지하고 추가 처리를 위해 들어오는 메시지를 적절한 처리기로 전달합니다.

네트워크 연결의 URI에 따라 해당 유형이 결정됩니다. 예를 들어 URI에 vm 를 지정하면 VM 내 연결이 생성됩니다.

```
<acceptor name="in-vm-example">vm://0</acceptor>
```

또는 URI에 tcp 를 지정하면 네트워크 연결이 생성됩니다. 예를 들면 다음과 같습니다.

```
<acceptor name="network-example">tcp://localhost:61617</acceptor>
```

다음 섹션에서는 네트워크 연결 및 VM 내 연결에 필요한 두 가지 중요한 구성 요소인 *어셉터*와 *커넥터*를 설명합니다. 이 섹션에서는 TCP, HTTP, SSL/TLS 네트워크 연결뿐만 아니라 VM 내 연결에 대한 어셉터 및 커넥터를 구성하는 방법을 보여줍니다.

2.1. 승인 정보

*어셉터*는 브로커에 대한 연결 방법을 정의합니다. 각 승인자는 클라이언트가 연결을 만드는 데 사용할 수 있는 포트 및 프로토콜을 정의합니다. 간단한 어셉터 구성은 다음과 같습니다.

```
<acceptors>
  <acceptor name="example-acceptor">tcp://localhost:61617</acceptor>
</acceptors>
```

브로커 구성에 정의된 각 어셉 터 요소는 단일 허용 요소에 포함됩니다. 브로커에 대해 정의할 수 있는 허용자 수에는 상한이 없습니다. 기본적으로 AMQ Broker에는 다음과 같이 지원되는 각 메시징 프로토콜에 대한 어셉터가 포함되어 있습니다.

```
<configuration ...>
  <core ...>
    ...
    <acceptors>
      ...
      <!-- Acceptor for every supported protocol -->
      <acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,STOMP,H
ORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor
>

      <!-- AMQP Acceptor. Listens on default AMQP port for AMQP traffic -->
      <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;a
mqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- STOMP Acceptor -->
      <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEpoll=true
</acceptor>

      <!-- HornetQ Compatibility Acceptor. Enables HornetQ Core and STOMP for legacy
HornetQ clients. -->
      <acceptor name="hornetq">tcp://0.0.0.0:5445?
anycastPrefix=jms.queue.;multicastPrefix=jms.topic.;protocols=HORNETQ,STOMP;useEpoll=t
rue</acceptor>

      <!-- MQTT Acceptor -->
      <acceptor name="mqtt">tcp://0.0.0.0:1883?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=MQTT;useEpoll=true</
acceptor>
    </acceptors>
    ...
  </core>
</configuration>
```

2.2. 어셉터 구성

다음 예제에서는 수락자를 구성하는 방법을 보여줍니다.

프로세스

1. &lt ;*broker_instance_dir*> /etc/broker.xml 구성 파일을 엽니다.

2. acceptor s 요소에서 새 어셉터 요소를 추가합니다. 브로커에 프로토콜 및 포트를 지정합니다. 예를 들면 다음과 같습니다.

```
<acceptors>
  <acceptor name="example-acceptor">tcp://localhost:61617</acceptor>
</acceptors>
```

위 예제에서는 **TCP** 프로토콜에 대한 어셉터를 정의합니다. 브로커는 **TCP**를 사용하는 클라이언트 연결에 대해 포트 **61617**에서 수신 대기합니다.

3. 어셉터에 대해 정의된 **URI**에 키-값 쌍을 추가합니다. 여러 키-값 쌍을 분리하려면 (;)를 사용합니다. 예를 들면 다음과 같습니다.

```
<acceptor name="example-acceptor">tcp://localhost:61617?sslEnabled=true;key-store-path=</path/to/key_store></acceptor>
```

이제 구성이 **TLS/SSL**을 사용하고 필요한 키 저장소의 경로를 정의하는 어셉터를 정의합니다.

추가 리소스

- 어셉터 및 커넥터에 대한 사용 가능한 구성 옵션에 대한 자세한 내용은 부록 **A.** *acceptor* 및 *Connector* 구성 매개변수 을 참조하십시오.

2.3. 커넥터 정보

허용자는 브로커가 연결을 *허용하는* 방법을 정의하지만 클라이언트가 커넥터를 사용하여 브로커에 *연결하는* 방법을 정의합니다.

브로커 자체가 클라이언트 역할을 할 때 커넥터가 브로커에 구성됩니다. 예를 들면 다음과 같습니다.

- 브로커가 다른 브로커에 브리지되는 경우
- 브로커가 클러스터에 참여하는 경우

간단한 커넥터 구성은 다음과 같습니다.

```
<connectors>
  <connector name="example-connector">tcp://localhost:61617</connector>
</connectors>
```

2.4. 커넥터 구성

다음 예제에서는 커넥터를 구성하는 방법을 보여줍니다.

프로세스

1. <*broker_instance_dir*> /etc/broker.xml 구성 파일을 엽니다.

2. 커넥터 요소에서 새 커넥터 요소를 추가합니다. 브로커에 프로토콜 및 포트를 지정합니다. 예를 들면 다음과 같습니다.

```
<connectors>
  <connector name="example-connector">tcp://localhost:61617</connector>
</connectors>
```

위 예제에서는 TCP 프로토콜에 대한 커넥터를 정의합니다. 클라이언트는 커넥터 구성을 사용하여 TCP 프로토콜을 사용하여 포트 61617의 브로커에 연결할 수 있습니다. 브로커 자체는 발신 연결에 이 커넥터를 사용할 수도 있습니다.

3. 커넥터에 대해 정의된 URI에 키-값 쌍을 추가합니다. 여러 키-값 쌍을 분리하려면 (;)를 사용합니다. 예를 들면 다음과 같습니다.

```
<connector name="example-connector">tcp://localhost:61616?
tcpNoDelay=true</connector>
```

이제 구성이 tcpNoDelay 속성 값을 true 로 설정하는 커넥터를 정의합니다. 이 속성의 값을 true 로 설정하면 연결에 대한 나글 알고리즘이 꺼집니다. 나글의 알고리즘은 소규모 데이터 패킷의 전송을 지연하고 대규모 패킷으로 통합함으로써 TCP 연결의 효율성을 개선하는 데 사용되는 알고리즘입니다.

추가 리소스

- 어셉터 및 커넥터에 대한 사용 가능한 구성 옵션에 대한 자세한 내용은 부록 A. *acceptor 및 Connector 구성 매개변수* 을 참조하십시오.

- AMQ Core Protocol JMS 클라이언트에서 브로커 커넥터를 구성하는 방법을 알아보려면 AMQ Core Protocol JMS 문서에서 브로커 커넥터 구성 을 참조하십시오.

2.5. TCP 연결 구성

AMQ Broker는 Netty를 사용하여 차단 Java IO 또는 최신 비차단 Java NIO를 사용하도록 구성할 수 있는 기본, 암호화되지 않은 TCP 기반 연결을 제공합니다. Java NIO는 많은 동시 연결로 더 나은 확장성을 위해 선호됩니다. 그러나 오래된 IO를 사용하면 수천 개의 동시 연결을 지원하는 것에 대해 덜 우려하는 경우 NIO보다 대기 시간을 향상시킬 수 있습니다.

신뢰할 수 없는 네트워크에서 연결을 실행하는 경우 TCP 네트워크 연결이 암호화되지 않음을 알고 있어야 합니다. 보안이 우선 순위인 경우 SSL 또는 HTTPS 구성을 사용하여 이 연결을 통해 전송된 메시지를 암호화하는 것이 좋습니다. 자세한 내용은 5.1절. “연결 보안”를 참조하십시오.

TCP 연결을 사용하는 경우 모든 연결이 클라이언트에서 시작됩니다. 브로커는 클라이언트에 대한 연결을 시작하지 않습니다. 이 기능은 한 방향에서 연결을 강제로 시작하는 방화벽 정책과 잘 작동합니다.

TCP 연결의 경우 호스트와 커넥터 URI의 포트는 연결에 사용되는 주소를 정의합니다.

다음 예제에서는 TCP 연결을 구성하는 방법을 보여줍니다.

사전 요구 사항

- 어셉터 및 커넥터 구성에 대해 잘 알고 있어야 합니다. 자세한 내용은 다음을 참조하십시오.
  - 2.2절. “어셉터 구성”
  - 2.4절. “커넥터 구성”

프로세스

1. <*broker_instance_dir*>/etc/broker.xml 구성 파일을 엽니다.
2. 새 어셉터를 추가하거나 기존 승인자를 수정합니다. 연결 URI에서 프로토콜로 tcp 를 지정합니다. 브로커에 IP 주소 또는 호스트 이름과 포트를 모두 포함합니다. 예를 들면 다음과 같습니다.

```
<acceptors>
  <acceptor name="tcp-acceptor">tcp://10.10.10.1:61617</acceptor>
  ...
```

```
</acceptors>
```

이전 예제를 기반으로 브로커는 IP 주소 10.10.10.1 에서 61617 포트에 연결된 클라이언트의 TCP 통신을 허용합니다.

3. (선택 사항) 유사한 방식으로 커넥터를 구성할 수 있습니다. 예를 들면 다음과 같습니다.

```
<connectors>
  <connector name="tcp-connector">tcp://10.10.10.2:61617</connector>
  ...
</connectors>
```

이전 예제의 커넥터는 지정된 IP 및 포트 10.10.10.2:61617 에 TCP를 연결할 때 클라이언트 또는 브로커 자체에서도 참조합니다.

추가 리소스

- TCP 연결에 사용 가능한 구성 옵션에 대한 자세한 내용은 부록 A. *acceptor 및 Connector 구성 매개변수* 을 참조하십시오.

2.6. HTTP 연결 구성

HTTP 연결은 HTTP 프로토콜을 통해 패킷을 터널링하며 방화벽에서 HTTP 트래픽만 허용하는 시나리오에서 유용합니다. AMQ Broker는 HTTP가 사용되는지 자동으로 감지하므로 HTTP에 대한 네트워크 연결을 구성하는 것은 TCP에 대한 연결을 구성하는 것과 동일합니다.

사전 요구 사항

- 어셉터 및 커넥터 구성에 대해 잘 알고 있어야 합니다. 자세한 내용은 다음을 참조하십시오.
  - 2.2절. “어셉터 구성”
  - 2.4절. “커넥터 구성”

프로세스

1. &lt ;*broker_instance_dir*> /etc/broker.xml 구성 파일을 엽니다.

2.

새 어셉터를 추가하거나 기존 승인자를 수정합니다. 연결 URI에서 프로토콜로 tcp 를 지정합니다. 브로커에 IP 주소 또는 호스트 이름과 포트를 모두 포함합니다. 예를 들면 다음과 같습니다.

```
<acceptors>
 <acceptor name="http-acceptor">tcp://10.10.10.1:80</acceptor>
 ...
</acceptors>
```

이전 예제를 기반으로 브로커는 IP 주소 10.10.10.1 에서 포트 80 에 연결된 클라이언트의 HTTP 통신을 허용합니다. 브로커는 HTTP 프로토콜이 사용 중인지를 자동으로 감지하고 그에 따라 클라이언트와 통신합니다.

3.

(선택 사항) 유사한 방식으로 커넥터를 구성할 수 있습니다. 예를 들면 다음과 같습니다.

```
<connectors>
 <connector name="http-connector">tcp://10.10.10.2:80</connector>
 ...
</connectors>
```

이전 예에 표시된 커넥터를 사용하여 브로커는 IP 주소 10.10.10.2 의 포트 80 에서 아웃바운드 HTTP 연결을 생성합니다.

추가 리소스

- HTTP 연결은 TCP와 동일한 구성 매개 변수를 사용하지만 일부도 있습니다. HTTP 연결에 사용 가능한 모든 구성 옵션에 대한 자세한 내용은 부록 A. *acceptor 및 Connector 구성 매개변수* 을 참조하십시오.
- HTTP 사용 방법을 보여주는 전체 작업 예제는 JMS HTTP 예제 를 참조하십시오.

2.7. 보안 네트워크 연결 구성

TLS/SSL을 사용하여 네트워크 연결을 보호할 수 있습니다. 자세한 내용은 5.1절. “연결 보안”의 내용을 참조하십시오.

2.8. VM 내 연결 구성

예를 들어 HA(고가용성) 구성의 일부로 여러 브로커가 동일한 가상 머신에 공동 배치되는 경우 VM 내 연결을 사용할 수 있습니다. VM 내 연결은 브로커와 동일한 JVM에서 실행되는 로컬 클라이언트에서도

사용할 수 있습니다.

사전 요구 사항

- 어셉터 및 커넥터 구성에 대해 잘 알고 있어야 합니다. 자세한 내용은 다음을 참조하십시오.
  - 2.2절. “어셉터 구성”
  - 2.4절. “커넥터 구성”

프로세스

1. &lt ;*broker_instance_dir*> /etc/broker.xml 구성 파일을 엽니다.

2. 새 어셉터를 추가하거나 기존 승인자를 수정합니다. 연결 URI에서 vm 을 프로토콜로 지정합니다. 예를 들면 다음과 같습니다.

```
<acceptors>
  <acceptor name="in-vm-acceptor">vm://0</acceptor>
  ...
</acceptors>
```

이전 예제의 수락자를 기반으로 브로커는 ID가 0 인 브로커의 연결을 수락합니다. 다른 브로커는 동일한 가상 머신에서 실행 중이어야 합니다.

3. (선택 사항) 유사한 방식으로 커넥터를 구성할 수 있습니다. 예를 들면 다음과 같습니다.

```
<connectors>
  <connector name="in-vm-connector">vm://0</connector>
  ...
</connectors>
```

이전 예제의 커넥터는 클라이언트와 동일한 가상 시스템에서 실행 중인 ID가 0 인 브로커에 대한 VM 내 연결을 설정할 수 있는 방법을 정의합니다. 클라이언트는 애플리케이션 또는 다른 브로커일 수 있습니다.

3장. 네트워크 연결에서 메시징 프로토콜 구성

AMQ Broker에는 플러그형 프로토콜 아키텍처가 있으므로 네트워크 연결에 대해 하나 이상의 프로토콜을 쉽게 활성화할 수 있습니다.

브로커는 다음 프로토콜을 지원합니다.

- **AMQP**
- **MQTT**
- **OpenWire**
- **STOMP**

참고

위의 프로토콜 외에도 브로커는 **"Core"**라는 자체 네이티브 프로토콜도 지원합니다. 이 프로토콜의 이전 버전은 **"HornetQ"**로 알려졌으며 **Red Hat JBoss Enterprise Application Platform**에서 사용되었습니다.

3.1. 메시징 프로토콜을 사용하도록 네트워크 연결 구성

프로토콜을 네트워크 연결과 연결해야 사용할 수 있습니다. (네트워크 연결 생성 및 구성 방법에 대한 자세한 내용은 *2장. 네트워크 연결에서 어셉터 및 커넥터 구성* 을 참조하십시오.) < ***broker_instance_dir*** > 파일에 있는 기본 구성에는 이미 정의된 여러 연결이 포함되어 있습니다. 편의를 위해 **AMQ Broker**에는 지원되는 각 프로토콜에 대한 **acceptor**와 모든 프로토콜을 지원하는 기본 수락자가 포함되어 있습니다.

기본 어셉터 개요

다음은 **broker.xml** 구성 파일에 기본적으로 포함된 어셉터입니다.

```
<configuration>
 <core>
  ...
  <acceptors>
```

```
      <!-- All-protocols acceptor -->
      <acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,STOMP,H
ORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredits=300</acceptor
>

      <!-- AMQP Acceptor. Listens on default AMQP port for AMQP traffic -->
      <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpoll=true;a
mqpCredits=1000;amqpLowCredits=300</acceptor>

      <!-- STOMP Acceptor -->
      <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEpoll=true
</acceptor>

      <!-- HornetQ Compatibility Acceptor. Enables HornetQ Core and STOMP for legacy
HornetQ clients. -->
      <acceptor name="hornetq">tcp://0.0.0.0:5445?
anycastPrefix=jms.queue.;multicastPrefix=jms.topic.;protocols=HORNETQ,STOMP;useEpoll=t
rue</acceptor>

      <!-- MQTT Acceptor -->
      <acceptor name="mqtt">tcp://0.0.0.0:1883?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=MQTT;useEpoll=true</
acceptor>

    </acceptors>
    ...
  </core>
</configuration>
```

지정된 네트워크 연결에서 프로토콜을 활성화하는 유일한 요구 사항은 **protocols** 매개변수를 수락자의 **URI**에 추가하는 것입니다. 매개 변수의 값은 쉼표로 구분된 프로토콜 이름 목록이어야 합니다. **protocol** 매개변수가 **URI**에서 생략되면 모든 프로토콜이 활성화됩니다.

예를 들어 **AMQP** 프로토콜을 사용하여 포트 **3232**에서 메시지를 수신하기 위한 어셉터를 생성하려면 다음 단계를 따르십시오.

1. **&lt ;*broker_instance_dir*> /etc/broker.xml** 구성 파일을 엽니다.

2. **< acceptors>** 스탠자에 다음 행을 추가합니다.

```
<acceptor name="ampq">tcp://0.0.0.0:3232?protocols=AMQP</acceptor>
```

기본 어셉터의 추가 매개변수

최소한의 수락자 구성에서는 프로토콜을 연결 **URI**의 일부로 지정합니다. 그러나 **broker.xml** 구성 파일

의 기본 어셉터에는 몇 가지 추가 매개변수가 구성되어 있습니다. 다음 표에는 기본 수락자에 대해 구성된 추가 매개변수가 자세히 설명되어 있습니다.

표 3.1. 어셉터의 추가 매개변수

수락자	매개변수	설명
all-protocols acceptor AMQP STOMP	tcpSendBufferSize	TCP 전송 버퍼의 크기(바이트)입니다. 기본값은 **32768** 입니다.
	tcpReceiveBufferSize	TCP 수신 버퍼의 크기(바이트)입니다. 기본값은 **32768** 입니다. TCP 버퍼 크기는 네트워크의 대역폭 및 대기 시간에 따라 조정해야 합니다. 요약하면 TCP send/receive buffer size는 다음과 같이 계산되어야 합니다. buffer_size = bandwidth * RTT. 여기서 대역폭은 초당 바이트 단위이고 네트워크 라운드 트립 시간(RTT)은 초 단위입니다. RTT는 **ping** 유틸리티를 사용하여 쉽게 측정할 수 있습니다. 빠른 네트워크의 경우 기본값에서 버퍼 크기를 늘릴 수 있습니다.
all-protocols acceptor AMQP STOMP HornetQ MQTT	useEpoll	이를 지원하는 시스템(Linux)을 사용하는 경우 Netty epoll을 사용합니다. Netty 네이티브 전송은 NIO 전송보다 더 나은 성능을 제공합니다. 이 옵션의 기본값은 **true** 입니다. 옵션을 **false** 로 설정하면 NIO가 사용됩니다.
all-protocols acceptor AMQP	amqpCredits	총 메시지 크기에 관계없이 AMQP 생산자가 보낼 수 있는 최대 메시지 수입니다. 기본값은 **1000** 입니다. AMQP 메시지를 차단하는 데 사용할 수 있는 방법에 대한 자세한 내용은 7.3.2절. “AMQP 생산자 차단” 에서 참조하십시오.
all-protocols acceptor AMQP	amqpLowCredits	브로커가 프로듀서를 보충하는 임계값을 낮추십시오. 기본값은 **300** 입니다. 생산자가 이 임계값에 도달하면 브로커는 **amqpCredits** 값을 복원하기 위해 생산자를 충분히 보냅니다. AMQP 메시지를 차단하는 데 사용할 수 있는 방법에 대한 자세한 내용은 7.3.2절. “AMQP 생산자 차단” 에서 참조하십시오.

수락자	매개변수	설명
HornetQ 호환성 허용기	anycastPrefix	클라이언트가 **anycast** 및 **멀티 캐스트를** 모두 사용하는 주소에 연결할 때 **anycast** 라우팅 유형을 지정하는 데 사용하는 접두사입니다. 기본값은 **jms.queue** 입니다. 주소에 연결할 때 라우팅 유형을 지정하도록 클라이언트를 활성화하도록 접두사를 구성하는 방법에 대한 자세한 내용은 4.6절. "허용되는 구성에 라우팅 유형 추가" 을 참조하십시오.
	multicastPrefix	**anycast** 및 **multicast** 를 모두 사용하는 주소에 연결할 때 클라이언트가 사용하는 멀티 캐스트 라우팅 유형을 지정하는 데 사용하는 접두사입니다. 기본값은 **jms.topic** 입니다. 주소에 연결할 때 라우팅 유형을 지정하도록 클라이언트를 활성화하도록 접두사를 구성하는 방법에 대한 자세한 내용은 4.6절. "허용되는 구성에 라우팅 유형 추가" 을 참조하십시오.

추가 리소스

- **Netty 네트워크 연결에 대해 구성할 수 있는 다른 매개변수에 대한 자세한 내용은 부록 A. *acceptor 및 Connector 구성 매개변수* 을 참조하십시오.**

3.2. 네트워크 연결에서 AMQP 사용

브로커는 AMQP 1.0 사양을 지원합니다. AMQP 링크는 소스와 대상(즉, 클라이언트와 브로커) 간의 메시지를 위한 단방향 프로토콜입니다.

프로세스

1. **&lt ;*broker_instance_dir*> /etc/broker.xml 구성 파일을 엽니다.**

2. **다음 예와 같이 AMQP의 값이 AMQP인 protocols 매개변수를 포함하여 AMQP 클라이언트를 수신하도록 어셉 터를 추가하거나 구성합니다.**

```
<acceptors>
  <acceptor name="amqp-acceptor">tcp://localhost:5672?protocols=AMQP</acceptor>
  ...
</acceptors>
```

이전 예에서 브로커는 기본 AMQP 포트인 포트 5672에서 AMQP 1.0 클라이언트를 허용합니다.

AMQP 링크에는 두 개의 끝점, 즉 발신자와 수신자가 있습니다. 발신자가 메시지를 전송할 때 브로커는 브로커의 대상으로 전달할 수 있도록 내부 형식으로 변환합니다. 수신자는 브로커의 대상에 연결하고 메시지를 제공하기 전에 AMQP로 다시 변환합니다.

AMQP 링크가 동적인 경우 임시 큐가 생성되고 원격 소스 또는 원격 대상 주소가 임시 큐의 이름으로 설정됩니다. 링크가 동적이 아닌 경우 큐에 원격 대상 또는 소스의 주소가 사용됩니다. 원격 대상 또는 소스가 없으면 예외가 전송됩니다.

링크 대상은 또한 기본 세션을 롤백하거나 커밋하는 데 사용되는 코디네이터일 수도 있습니다.

참고

AMQP는 세션당 여러 트랜잭션, amqp:multi-txns-per-ssn 을 사용할 수 있지만 현재 버전의 AMQ Broker는 세션당 단일 트랜잭션만 지원합니다.

참고

AMQP 내의 분산 트랜잭션(XA)의 세부 사항은 사양 1.0 버전에서는 제공되지 않습니다. 환경에 분산 트랜잭션 지원이 필요한 경우 AMQ Core Protocol JMS를 사용하는 것이 좋습니다.

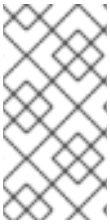
프로토콜 및 해당 기능에 대한 자세한 내용은 AMQP 1.0 사양을 참조하십시오.

3.2.1. AMQP 링크를 주제로 사용

JMS와 달리 AMQP 프로토콜은 주제를 포함하지 않습니다. 그러나 대기열의 소비자가 아닌 AMQP 소비자 또는 수신자를 서브스크립션으로 처리할 수 있습니다. 기본적으로 jms.topic 접두사가 있는 주소에 연결하는 수신 링크는 서브스크립션으로 처리되며 서브스크립션 큐가 생성됩니다. 서브스크립션 큐는 Terminus Durability 구성 방법에 따라 다음 표에서 캡처된 대로 지속성 또는 휘발성으로 구성됩니다.

표 3.2. 서브스크립션 대기열

멀티 캐스트 전용 큐에 대해 이러한 종류의 서브스크립션을 생성하려면 다음을 수행합니다.	Terminus Durability를 이...로 설정합니다.
Cryostat	UNSETTLED_STATE 또는 CONFIGURATION
취소할 수 없음	NONE



참고

Cryostat 큐의 이름은 컨테이너 ID와 링크 이름으로 구성됩니다(예: my-container-id:my-link-name).

AMQ Broker는 또한 qpid-jms 클라이언트를 지원하며, 주소에 사용되는 접두사와 관계없이 주제 사용을 준수합니다.

3.2.2. AMQP 보안 구성

브로커는 AMQP SASL 인증을 지원합니다. 브로커에서 SASL 기반 인증을 구성하는 방법에 대한 자세한 내용은 보안 을 참조하십시오.

3.3. 네트워크 연결로 MQTT 사용

브로커는 MQTT v3.1.1 및 v5.0 (및 이전 v3.1 코드 메시지 형식)을 지원합니다. MQTT는 서버에 대한 경량 클라이언트, 게시/서브스크립션 메시징 프로토콜입니다. MQTT는 메시징 오버헤드 및 네트워크 트래픽뿐만 아니라 클라이언트의 코드 풋프린트를 줄입니다. 이러한 이유로 MQTT는 센서 및 액추에이터와 같은 제한된 장치에 가장 적합하며 IoT(사물 인터넷)의 사실상 표준 통신 프로토콜이 빠르게 증가하고 있습니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. MQTT 프로토콜이 활성화된 어셉터를 추가합니다. 예를 들면 다음과 같습니다.

```
<acceptors>
  <acceptor name="mqtt">tcp://localhost:1883?protocols=MQTT</acceptor>
  ...
</acceptors>
```

MQTT에는 다음과 같은 여러 유용한 기능이 포함되어 있습니다.

서비스 품질

각 메시지는 연결된 서비스 품질을 정의할 수 있습니다. 브로커는 정의된 최고 수준의 서비스 수준에서 구독자에게 메시지를 전달하려고 합니다.

보존된 메시지

메시지는 특정 주소에 대해 유지될 수 있습니다. 해당 주소에 대한 새 구독자는 클라이언트가 연결되기 전에 보존된 메시지가 전송되어도 다른 메시지보다 먼저 유지된 메시지를 수신합니다.

보존된 메시지는 **sys.mqtt.<*topic name*>**라는 큐에 저장되며 클라이언트가 보존된 메시지를 삭제할 때까지 큐에 남아 있거나, 메시지가 만료될 때까지 만료가 구성될 때까지 큐에 남아 있습니다. 큐가 비어 있으면 명시적으로 삭제할 때까지 큐가 제거되지 않습니다. 예를 들어 다음 구성은 큐를 삭제합니다.

```
<address-setting match="$sys.mqtt.retain.#">
  <auto-delete-queues>true</auto-delete-queues>
  <auto-delete-addresses>true</auto-delete-addresses>
</address-setting>
```

와일드카드 서브스크립션

MQTT 주소는 파일 시스템의 계층 구조와 유사하게 계층 구조입니다. 클라이언트는 특정 주제 또는 계층 구조의 전체 분기를 구독할 수 있습니다.

will Messages

클라이언트는 연결 패킷의 일부로 **"will message"**를 설정할 수 있습니다. 클라이언트가 비정상적으로 연결이 끊어지면 브로커는 **will** 메시지를 지정된 주소에 게시합니다. 다른 구독자는 해당 메시지를 수신하고 그에 따라 응답할 수 있습니다.

MQTT 프로토콜에 대한 자세한 내용은 사양을 참조하십시오.

- **MQTT 3.11 사양**
- **MQTT 5.0 사양**

3.3.1. MQTT 속성 구성

MQTT 수락자에 키-값 쌍을 추가하여 연결 속성을 구성할 수 있습니다. 예를 들면 다음과 같습니다.

```
<acceptors>
  <acceptor name="mqtt">tcp://localhost:1883?
protocols=MQTT;receiveMaximum=50000;topicAliasMaximum=50000;maximumPacketSize;13
4217728;
```

```
serverKeepAlive=30;closeMqttConnectionOnPublishAuthorizationFailure=false</acceptor>
  ...
</acceptors>
```

receiveMaximum

승인이 필요하기 전에 브로커가 클라이언트에서 수신할 수 있는 최대 **QoS 1** 및 **2** 메시지 수를 지정하여 흐름 제어를 활성화합니다. 기본값은 **65535** 입니다. 값 **-1** 은 클라이언트에서 브로커로의 흐름 제어를 비활성화합니다. 이는 값을 **0**으로 설정하는 것과 동일하지만 **CONNACK** 패킷의 크기가 줄어듭니다.

topicAliasMaximum

클라이언트에 대해 브로커가 지원하는 최대 별칭 수를 지정합니다. 기본값은 **65535** 입니다. 값 **-1**을 사용하면 브로커가 클라이언트에 항목 별칭 제한을 알릴 수 없습니다. 이는 값을 **0**으로 설정하는 것과 동일하지만 **CONNACK** 패킷의 크기가 줄어듭니다.

maximumPacketSize

브로커가 클라이언트에서 허용할 수 있는 최대 패킷 크기를 지정합니다. 기본값은 **268435455** 입니다. 값 **-1**을 사용하면 브로커가 클라이언트에 최대 패킷 크기를 알릴 수 없으므로 들어오는 패킷 크기에 제한이 적용되지 않습니다.

serverKeepAlive

브로커가 비활성 클라이언트 연결을 열린 상태로 유지하는 기간을 지정합니다. 구성된 값은 클라이언트에 대해 구성된 **keep-alive** 값보다 작거나 클라이언트에 구성된 값이 **0**인 경우에만 연결에 적용됩니다. 기본값은 **60** 초입니다. 값 **-1** 은 브로커가 항상 클라이언트의 활성 값을 허용합니다(해당 값이 **0**인 경우에도).

closeMqttConnectionOnPublishAuthorizationFailure

기본적으로 권한 부여 부족으로 인해 **PUBLISH** 패킷이 실패하면 브로커는 네트워크 연결을 종료합니다. 브로커가 네트워크 연결을 닫는 대신 긍정적인 승인을 보내려면 **closeMqttConnectionOnPublishAuthorizationFailure** 를 **false** 로 설정합니다.

3.4. 네트워크 연결과 함께 OPENWIRE 사용

브로커는 **JMS** 클라이언트가 브로커와 직접 통신할 수 있는 **OpenWire** 프로토콜 을 지원합니다. 이 프로토콜을 사용하여 이전 버전의 **AMQ Broker**와 통신합니다.

현재 **AMQ Broker**는 표준 **JMS API**만 사용하는 **OpenWire** 클라이언트를 지원합니다.

프로세스

1. &lt ;*broker_instance_dir*> /etc/broker.xml 구성 파일을 엽니다.

2. 다음 예와 같이 **protocol** 매개변수의 일부로 **OPENWIRE** 를 포함하도록 수락자를 추가하거나 수정합니다.

```
<acceptors>
  <acceptor name="openwire-acceptor">tcp://localhost:61616?
protocols=OPENWIRE</acceptor>
  ...
</acceptors>
```

이전 예에서 브로커는 들어오는 **OpenWire** 명령의 포트 **61616**에서 수신 대기합니다.

자세한 내용은 **Openwire 예제** 를 참조하십시오.

3.5. 네트워크 연결로 STOMP 사용

STOMP 는 **STOMP** 클라이언트가 **STOMP** 브로커와 통신할 수 있도록 하는 텍스트 지향 유선 프로토콜입니다. 브로커는 **STOMP 1.0, 1.1** 및 **1.2**를 지원합니다. **STOMP** 클라이언트는 여러 언어와 플랫폼에서 사용할 수 있으므로 상호 운용성에 적합합니다.

프로세스

1. &lt ;*broker_instance_dir*> /etc/broker.xml 구성 파일을 엽니다.

2. 기존 어셉 터를 구성하거나 새 어셉터를 생성하고 다음과 같이 **STOMP** 값을 사용하여 **protocols** 매개변수를 포함합니다.

```
<acceptors>
  <acceptor name="stomp-acceptor">tcp://localhost:61613?protocols=STOMP</acceptor>
  ...
</acceptors>
```

이전 예에서 브로커는 기본값인 **61613** 포트에서 **STOMP** 연결을 허용합니다.

STOMP를 사용하여 브로커를 구성하는 방법의 예는 **STOMP 예제** 를 참조하십시오.

3.5.1. STOMP 제한 사항

STOMP를 사용하는 경우 다음과 같은 제한 사항이 적용됩니다.

1. 브로커는 현재 가상 호스팅을 지원하지 않으므로 CONNECT 프레임의 호스트 헤더가 무시됩니다.

2. 메시지 확인은 트랜잭션되지 않습니다. ACK 프레임은 트랜잭션의 일부일 수 없으며 트랜잭션 헤더가 설정된 경우 무시됩니다.

3.5.2. STOMP 메시지에 대한 ID 제공

JMS 소비자 또는 QueueBrowser를 통해 STOMP 메시지를 수신하는 경우 메시지에는 기본적으로 JMSMessageID 와 같은 JMS 속성이 포함되지 않습니다. 그러나 브로커 paramater를 사용하여 들어오는 각 STOMP 메시지에 메시지 ID를 설정할 수 있습니다.

프로세스

1. &lt ;*broker_instance_dir&gt*; /etc/broker.xml 구성 파일을 엽니다.

2. 다음 예와 같이 STOMP 연결에 사용되는 acceptor 에 대해 stompEnableMessageId 매개변수를 true 로 설정합니다.

```
<acceptors>
  <acceptor name="stomp-acceptor">tcp://localhost:61613?
protocols=STOMP;stompEnableMessageId=true</acceptor>
  ...
</acceptors>
```

stompEnableMessageId 매개변수를 사용하면 이 수락자를 사용하여 전송된 각 stomp 메시지에 추가 속성이 추가되었습니다. 속성 키는 amq-message-id 이고 값은 다음 예와 같이 "STOMP" 접두사가 붙은 내부 메시지 ID의 문자열 표현입니다.

```
amq-message-id : STOMP12345
```

구성에 stompEnableMessageId 가 지정되지 않은 경우 기본값은 false 입니다.

3.5.3. 실시간 연결 시간 설정

STOMP 클라이언트는 연결을 닫기 전에 **DISCONNECT** 프레임을 보내야 합니다. 이를 통해 브로커는 세션 및 소비자와 같은 서버 측 리소스를 닫을 수 있습니다. 그러나 **DISCONNECT** 프레임을 보내지 않고 **STOMP** 클라이언트가 종료되거나 실패하는 경우 브로커는 클라이언트가 아직 활성 상태인지 여부를 즉시 알 수 없습니다. 따라서 **STOMP** 연결은 **"TTL (Time to Live)" (TTL) 1** 분을 갖도록 구성됩니다. 즉, **1**분 이상 유휴 상태인 경우 브로커가 **STOMP** 클라이언트로의 연결을 중지합니다.

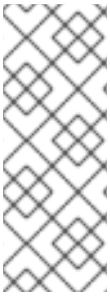
프로세스

1. **&lt ;*broker_instance_dir*> /etc/broker.xml** 구성 파일을 엽니다.

2. 다음 예와 같이 **STOMP** 연결에 사용되는 수락 자의 **URI**에 **connectionTTL** 매개변수를 추가합니다.

```
<acceptors>
  <acceptor name="stomp-acceptor">tcp://localhost:61613?
protocols=STOMP;connectionTTL=20000</acceptor>
  ...
</acceptors>
```

이전 예에서 **stomp-acceptor** 를 사용하는 모든 **stomp** 연결은 **TTL**이 **20**초로 설정됩니다.



참고

STOMP 프로토콜 버전 **1.0**에는 하트비트 프레임이 포함되어 있지 않습니다. 따라서 데이터가 **connection-ttl** 내에서 전송되었는지 확인하는 사용자의 책임이거나 브로커는 클라이언트가 종료되었다고 가정하고 서버 측 리소스를 정리해야 합니다. 버전 **1.1**에서는 **heart-beats**를 사용하여 **stomp** 연결의 라이프 사이클을 유지할 수 있습니다.

브로커 기본 시간 덮어쓰기

앞서 언급했듯이 **STOMP** 연결의 기본 **TTL**은 **1**분입니다. 브로커 구성에 **connection-ttl-override** 속성을 추가하여 이 값을 덮어쓸 수 있습니다.

프로세스

1. **&lt ;*broker_instance_dir*> /etc/broker.xml** 구성 파일을 엽니다.

2. **connection-ttl-override** 속성을 추가하고 새 기본값에 대한 값을 밀리초 단위로 제공합니다. 아래와 같이 **&lt ;core& gt;** 스탠자 내부에 있습니다.

-

```
<configuration ...>
  ...
  <core ...>
    ...
    <connection-ttl-override>30000</connection-ttl-override>
    ...
  </core>
<configuration>
```

이전 예에서 **STOMP** 연결의 기본 **Time to Live(TTL)**는 **30**초, **30000** 밀리초로 설정됩니다.

3.5.4. JMS에서 STOMP 메시지 전송 및 사용

STOMP는 주로 텍스트 지향 프로토콜입니다. **JMS**와 상호 운용하기 쉽도록 **STOMP** 구현에서는 **content-length** 헤더가 있는지 확인하여 **STOMP** 메시지를 **JMS**에 매핑하는 방법을 결정합니다.

표 3.3. STOMP 메시지를 JMS에 매핑

STOMP 메시지가 ...에 매핑되도록 하려면	메시지는...이어야 합니다.
JMS textMessage	**content-length** 헤더를 포함하지 않습니다.
JMS BytesMessage	**content-length** 헤더를 포함합니다.

JMS 메시지를 **STOMP**에 매핑할 때 동일한 논리가 적용됩니다. **STOMP** 클라이언트는 **content-length** 헤더가 있는지 확인하여 메시지 본문의 유형을 확인할 수 있습니다(문자열 또는 바이트).

메시지 헤더에 대한 자세한 내용은 **STOMP** 사양을 참조하십시오.

3.5.5. STOMP 대상을 AMQ Broker 주소 및 큐에 매핑

메시지를 보내고 구독할 때 **STOMP** 클라이언트는 일반적으로 대상 헤더를 포함합니다. 대상 이름은 브로커의 대상에 매핑되는 문자열 값입니다. **AMQ Broker**에서 이러한 대상은 주소 및 큐에 매핑됩니다. 대상 프레임에 대한 자세한 내용은 **STOMP** 사양을 참조하십시오.

예를 들어 다음 메시지(헤드 및 본문 포함)를 전송하는 **STOMP** 클라이언트를 예로 들 수 있습니다.

```
SEND
destination:/my/stomp/queue
```

```
hello queue a
^@
```

이 경우 브로커는 **/my/stomp/queue** 주소와 연결된 모든 큐에 메시지를 전달합니다.

예를 들어 **SEND** 프레임을 사용하여 **STOMP** 클라이언트가 메시지를 보내면 지정된 대상이 주소에 매핑됩니다.

클라이언트가 **SUBSCRIBE** 또는 **UNSUBSCRIBE** 프레임을 보낼 때와 동일한 방식으로 작동하지만, 이 경우 **AMQ Broker**는 대상 을 큐에 매핑합니다.

```
SUBSCRIBE
destination: /other/stomp/queue
ack: client

^@
```

이전 예에서 브로커는 대상 을 **/other/stomp/queue** 에 매핑합니다.

STOMP 대상을 **JMS** 대상에 매핑

JMS 대상은 브로커 주소 및 큐에도 매핑됩니다. **STOMP**를 사용하여 **JMS** 대상에 메시지를 보내려면 **STOMP** 대상이 동일한 규칙을 따라야 합니다.

- **jms.queue**에 의해 큐 이름 앞에 따라 **JMS Queue** 를 보내거나 구독합니다. 예를 들어, **JMS Queue** 주문에 메시지를 보내려면 **STOMP** 클라이언트에서 프레임을 보내야 합니다.

```
SEND
destination:jms.queue.orders
hello queue orders
^@
```

- **jms.topic**에 의해 주제 이름 앞에 따라 **JMS Topic** 을 보내거나 구독합니다. . 예를 들어, 주식 **JMS Topic**을 구독하려면 **STOMP** 클라이언트는 다음과 유사한 프레임을 보내야 합니다.

```
SUBSCRIBE
destination:jms.topic.stocks
^@
```

4장. 주소 및 큐 구성

4.1. 주소, 큐 및 라우팅 유형

AMQ Broker에서 주소 지정 모델은 세 가지 주요 개념, 즉 *주소*,*큐* 및 *라우팅 유형*으로 구성됩니다.

주소는 메시징 끝점을 나타냅니다. 구성 내에서 일반적인 주소에는 고유한 이름, 하나 이상의 대기열 및 라우팅 유형이 제공됩니다.

큐 는 주소와 연결되어 있습니다. 주소당 큐가 여러 개 있을 수 있습니다. 들어오는 메시지가 주소와 일치하면 구성된 라우팅 유형에 따라 하나 이상의 대기열로 메시지가 전송됩니다. 큐를 자동으로 생성하고 삭제하도록 구성할 수 있습니다. 주소(및 이에 따라 연결된 대기열)를 **Cryostat**로 구성할 수도 *있습니다*. **queue**의 메시지도 지속되는 한 **Cryostat** 큐의 메시지는 충돌하거나 브로커를 다시 시작할 수 있습니다. 반대로, 실행 불가능한 큐의 메시지는 메시지 자체가 영구적인 경우에도 충돌하거나 브로커를 다시 시작할 수 없습니다.

라우팅 유형에 따라 주소와 연결된 큐로 메시지를 보내는 방법이 결정됩니다. **AMQ Broker**에서는 표에 표시된 것처럼 두 가지 다른 라우팅 유형으로 주소를 구성할 수 있습니다.

표 **4.1.** 주소 라우팅 유형

메시지가...로 라우팅되는 경우	이 라우팅 유형 사용...
지점 간 방식으로 일치하는 주소 내의 단일 큐	**anycast**
게시-서브스크립션 방식으로 일치하는 주소 내의 모든 대기열	**멀티 캐스트**

참고

주소에는 하나 이상의 라우팅 유형이 있어야 합니다.

주소당 두 개 이상의 라우팅 유형을 정의할 수 있지만 이는 권장되지 않습니다.

주소에 두 라우팅 유형이 정의되어 있고 클라이언트에 기본 설정이 표시되지 않는 경우 브로커는 기본적으로 멀티캐스트 라우팅 유형으로 설정됩니다.

추가 리소스

- 구성에 대한 자세한 내용은 다음을 수행합니다.
  - **anycast** 라우팅 유형을 사용하는 지점 간 메시징을 참조하십시오. 4.3절. “지점 간 메시징에 대한 주소 구성”
  - 멀티 캐스트 라우팅 유형을 사용한 게시-서브스크립션 메시징 참조 4.4절. “게시-서브스크립션 메시징을 위한 주소 구성”

4.1.1. 주소 및 큐 이름 지정 요구 사항

주소 및 큐를 구성할 때 다음 요구 사항을 유의하십시오.

- 클라이언트가 사용하는 유선 프로토콜에 관계없이 클라이언트가 큐에 연결할 수 있도록 하려면 주소와 큐 이름에 다음 문자가 포함되어 서는 안 됩니다.

  & :: , ? >

- 숫자 기호(#) 및 별표(*) 문자는 와일드카드 표현식용으로 예약되어 있으며 주소 및 큐 이름에 사용해서는 안 됩니다. 자세한 내용은 4.2.1절. “AMQ Broker 와일드카드 구문”의 내용을 참조하십시오.
- 주소 및 큐 이름에는 공백을 포함하지 않아야 합니다.
- 주소 또는 큐 이름으로 단어를 분리하려면 구성된 구분 기호 문자를 사용합니다. 기본 구분 기호 문자는 마침표(.)입니다. 자세한 내용은 4.2.1절. “AMQ Broker 와일드카드 구문”의 내용을 참조하십시오.

4.2. 주소 세트에 주소 설정 적용

AMQ Broker에서는 일치하는 주소 이름을 나타내는 와일드카드 표현식을 사용하여 **address-setting** 요소에 지정된 구성을 주소 집합에 적용할 수 있습니다.

다음 섹션에서는 와일드카드 표현식을 사용하는 방법에 대해 설명합니다.

4.2.1. AMQ Broker 와일드카드 구문

AMQ Broker는 주소 설정에서 와일드카드를 표시하기 위해 특정 구문을 사용합니다. 와일드카드는 보안 설정 및 소비자를 생성할 때 사용할 수도 있습니다.

- 와일드카드 표현식에는 마침표(.)로 구분된 단어가 포함되어 있습니다.
- 숫자 기호(#)와 별표(*) 문자도 특별한 의미가 있으며 다음과 같이 한 단어로 대체할 수 있습니다.
  - 숫자 기호 문자는 "0개 이상의 단어로 이루어진 모든 시퀀스와 일치"를 의미합니다. 표현식의 끝에 이 값을 사용합니다.
  - 별표 문자는 "단일한 단어 일치"를 의미합니다. 표현식 내 어디에서나 사용하십시오.

일치는 문자별 문자가 아니라 각 구분 기호 경계에서 수행됩니다. 예를 들어 이름에 있는 큐와 일치하도록 구성된 **address-setting** 요소는 **my queue** 라는 큐와 일치하지 않습니다.

둘 이상의 **address-setting** 요소가 주소가 일치하는 경우 브로커는 기준선으로 가장 구체적인 일치의 구성을 사용하여 오버레이 구성을 사용합니다. 리터럴 표현식은 와일드카드보다 다르며 별표(*)는 숫자 기호(#)보다 더 구체적입니다. 예를 들어 **my.destination** 과 **my.*** 둘 다 **my.destination** 주소와 일치합니다. 이 경우 브로커는 와일드카드 표현식이 리터럴보다 작기 때문에 먼저 **my.*** 아래의 구성을 적용합니다. 다음으로 브로커는 **my.destination address** 설정 요소의 구성을 오버레이하여 **my.*** 와 공유된 모든 구성을 덮어씁니다. 예를 들어 다음 구성에서 **my.destination** 과 연결된 큐에는 **max-delivery-attempts** 가 **3** 으로 설정되고 **last-value-queue** 가 **false** 로 설정됩니다.

```
<address-setting match="my.*">
   <max-delivery-attempts>3</max-delivery-attempts>
   <last-value-queue>true</last-value-queue>
</address-setting>
<address-setting match="my.destination">
   <last-value-queue>false</last-value-queue>
</address-setting>
```

다음 표의 예제에서는 와일드카드를 사용하여 주소 집합과 일치하는 방법을 보여줍니다.

표 **4.2.** 와일드카드를 사용하여 일치하는 주소

예	설명
#	**broker.xml** 에서 사용되는 기본 **address-setting** 입니다. 모든 주소와 일치합니다. 이 catch-all을 계속 적용하거나 필요에 따라 각 주소 또는 주소 그룹에 대해 새 **address-setting** 을 추가할 수 있습니다.
news.europe.#	**news.europe**,**news.europe.sport**,**news.europe.politics.fr** 과 일치하지만 **news.usa** 또는 **europe** 는 일치하지 않습니다.
news.*	**news.europe** 및 **news.usa** 와 일치하지만 **news.europe.sport** 는 일치하지 않습니다.
news.*.sport	**news.europe.sport** 및 **news.usa.sport** 와 일치하지만 **news.europe.fr.sport** 는 일치하지 않습니다.

4.2.2. 리터럴 일치 구성

*리터럴 일치에서 와일드카드 문자는 와일드카드가 포함된 주소와 일치하도록 리터럴 문자로 처리됩니다. 예를 들어 리터럴 일치의 해시(#) 문자는 **orders.retail** 또는 **orders.wholesale**과 같은 일치하는 주소 없이 **orders.#**과 일치할 수 있습니다.**For example, the hash (#) character in a literal match can match an address of orders.# without matching addresses such as orders.retail or orders.wholesale.***

프로세스

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. *리터럴 일치를 구성하기 전에 **literal-match-markers** 매개변수를 사용하여 리터럴 일치를 구분하는 문자를 정의합니다. 다음 예제에서는 리터럴 일치를 제한 해제하는 데 사용됩니다.*

```
<core>
  ...
  <literal-match-markers>()</literal-match-markers>
  ...
</core>
```

3. *리터럴 일치를 구분하는 마커를 정의한 후 **address** 설정 **match** 매개 변수에 마커를 포함하여 일치 항목을 지정합니다. 다음 예제에서는 **orders.#** 이라는 주소에 대해 리터럴 일치를 구성하여 해당 특정 주소에 대한 지표를 활성화합니다.*

```
<address-settings>
  <address-setting match="(orders.#)">
```

```
    <enable-metrics>true</enable-metrics>
  </address-setting>
</address-settings>
```

4.2.3. 브로커 와일드카드 구문 구성

다음 절차에서는 와일드카드 주소에 사용되는 구문을 사용자 지정하는 방법을 보여줍니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. 아래 예제 와 같이 **<wildcard-addresses >** 섹션을 구성에 추가합니다.

```
<configuration>
 <core>
  ...
  <wildcard-addresses> //
   <enabled>true</enabled> //
   <delimiter>,</delimiter> //
   <any-words>@</any-words> //
   <single-word>$</single-word>
  </wildcard-addresses>
  ...
 </core>
</configuration>
```

enabled

true 로 설정하면 브로커에 사용자 지정 설정을 사용하도록 지시합니다.

구분 기호

기본값 대신 구분 기호 로 사용할 사용자 지정 문자를 제공합니다 .

any-words

임의의 단어에 대한 값으로 제공되는 문자는 **'0**개 이상의 단어 와 일치하는 시퀀스'를 의미하고 기본값 **#** 을 대체합니다. 표현식 끝에 이 문자를 사용합니다.

단일 단어

단일 단어에 대한 값으로 제공되는 문자는 ' 단일한 단어 '와 일치하는 데 사용되며 기본값 *.를 대체합니다. 표현식 내 어느 곳에나 이 문자를 사용합니다.

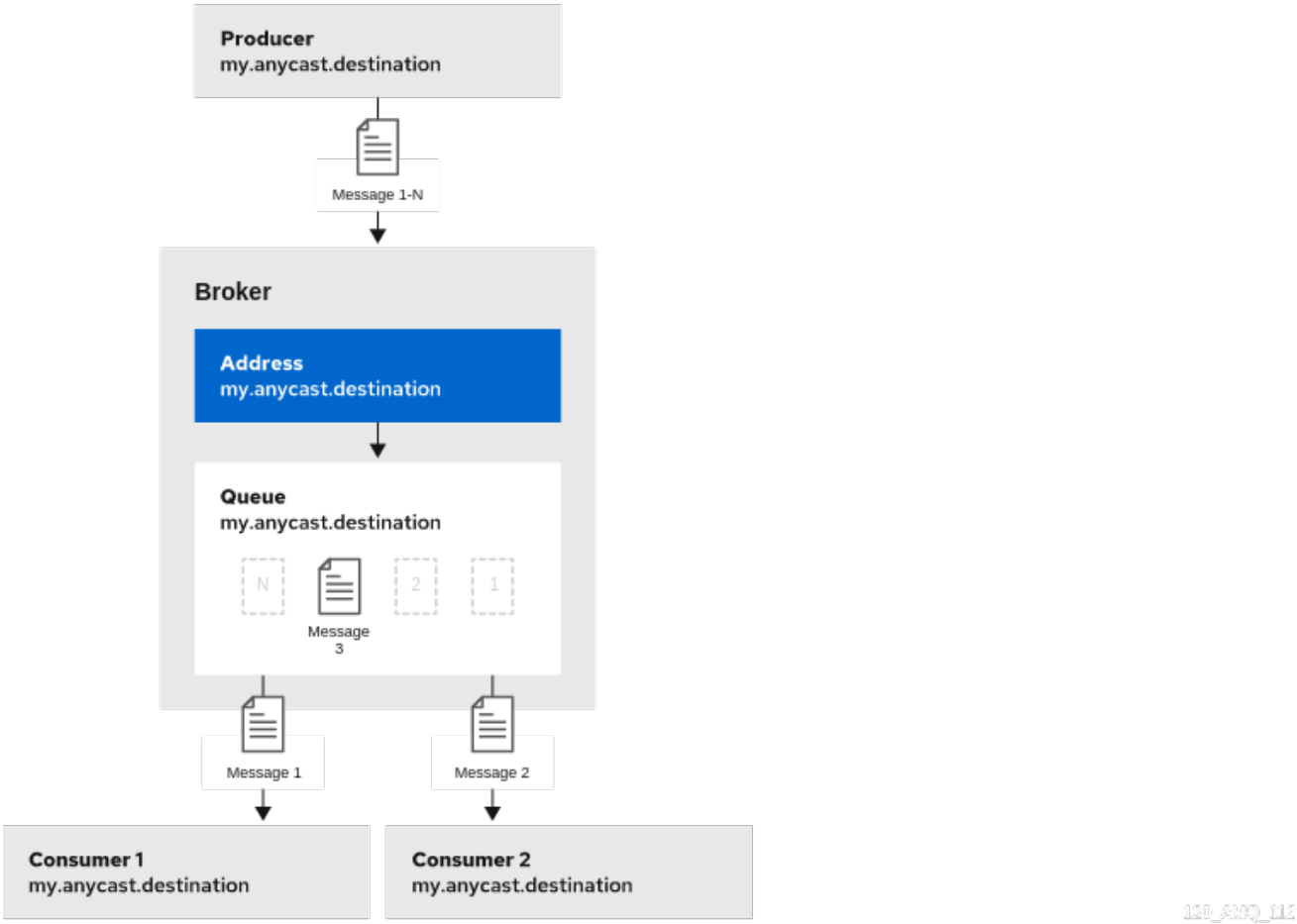
4.3. 지점 간 메시징에 대한 주소 구성

점대점 메시징은 생산자가 보낸 메시지에는 하나의 소비자만 있는 일반적인 시나리오입니다. **AMQP** 및 **JMS** 메시지 생산자와 소비자는 예를 들어 점대점 메시징 대기열을 사용할 수 있습니다. 주소와 연결된 큐가 지점 간 방식으로 메시지를 수신하도록 브로커 구성에서 지정된 주소 요소에 대한 **anycast** 라우팅 유형을 정의합니다.

anycast 를 사용하여 주소에서 메시지를 수신하면 브로커는 주소와 연결된 큐를 찾고 메시지를 라우팅합니다. 그러면 소비자가 해당 대기열의 메시지를 사용하도록 요청할 수 있습니다. 여러 소비자가 동일한 큐에 연결하면 소비자가 동일하게 처리할 수 있는 메시지가 소비자 간에 배포됩니다.

다음 그림은 점대점 메시징의 예를 보여줍니다.

그림 **4.1. point-to-point** 메시징



4.3.1. 기본 지점 간 메시징 구성

다음 절차에서는 지점 간 메시징에 대해 단일 큐로 주소를 구성하는 방법을 보여줍니다.

프로세스

1. <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. 주소의 선택한 큐 요소 주위에 anycast 구성 요소를 래핑합니다. address 및 queue 요소 모두에 대한 name 속성 값이 동일한지 확인합니다. 예를 들면 다음과 같습니다.

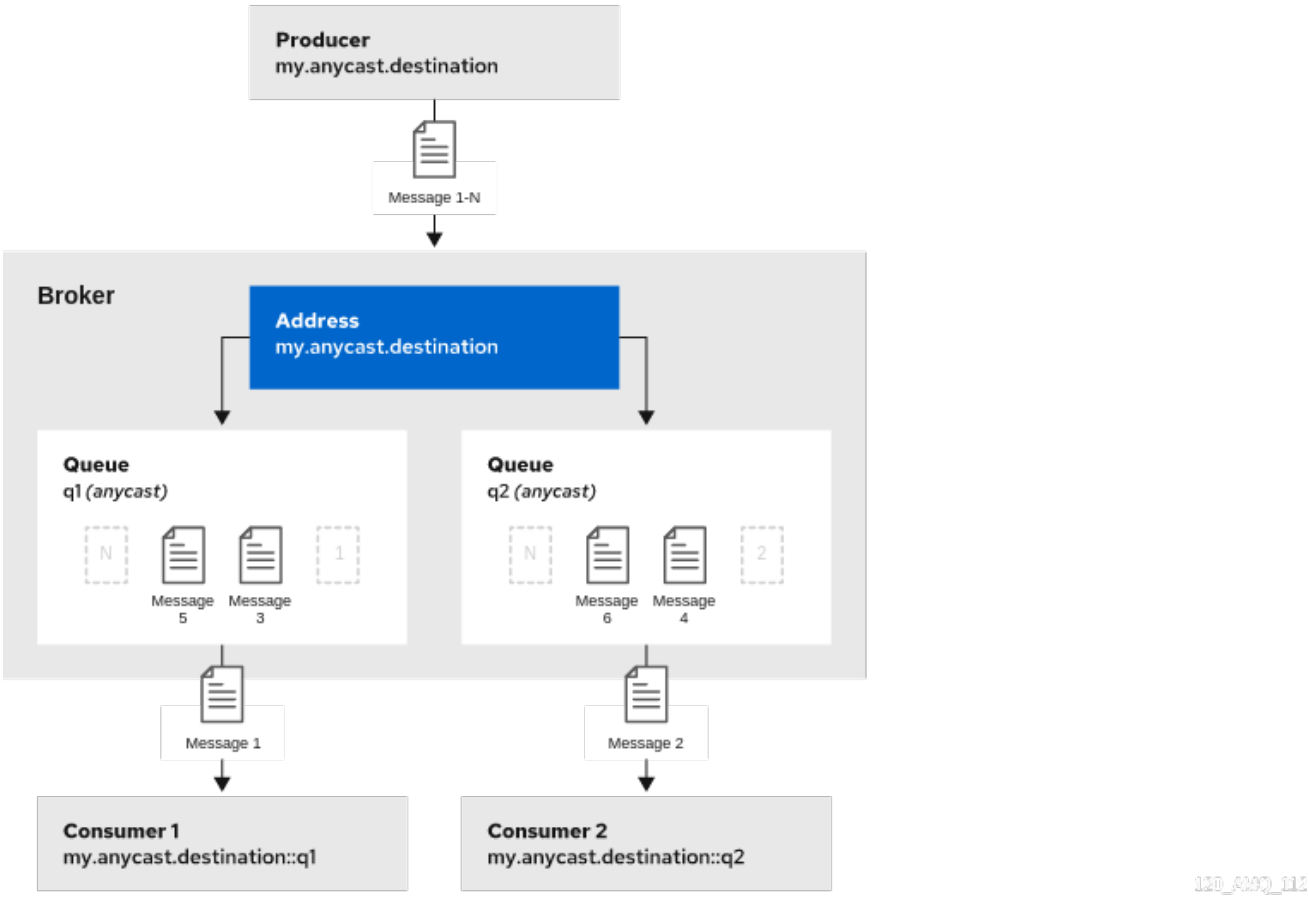
```
<configuration ...>
 <core ...>
  ...
  <address name="my.anycast.destination">
   <anycast>
    <queue name="my.anycast.destination"/>
   </anycast>
  </address>
 </core>
</configuration>
```

4.3.2. 여러 큐에 대한 지점 간 메시징 구성

anycast 라우팅 유형을 사용하는 주소에 두 개 이상의 큐를 정의할 수 있습니다. 브로커는 연결된 모든 큐에 대해 anycast 주소로 전송된 메시지를 균등하게 배포합니다. FQN(Fully Qualified Queue Name) 을 지정하면 클라이언트를 특정 큐에 연결할 수 있습니다. 둘 이상의 소비자가 동일한 큐에 연결하면 브로커는 메시지를 소비자 간에 균등하게 배포합니다.

다음 그림은 두 개의 큐를 사용하는 점대점 메시징의 예를 보여줍니다.

그림 4.2. 두 개의 큐를 사용한 점대점 메시징



다음 절차에서는 여러 큐가 있는 주소에 대해 점대점 메시징을 구성하는 방법을 보여줍니다.

프로세스

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. ***address** 요소의 큐 요소 주위에 **anycast** 구성 요소를 래핑합니다. 예를 들면 다음과 같습니다.*

```
<configuration ...>
  <core ...>
    ...
    <address name="my.anycast.destination">
      <anycast>
        <queue name="q1"/>
        <queue name="q2"/>
      </anycast>
    </address>
  </core>
</configuration>
```

클러스터의 여러 브로커에 대해 위에 미러링된 것과 같은 구성이 있는 경우 클러스터는 생산자 및 소비자에 불투명한 방식으로 지점 간 메시징을 로드 밸런싱할 수 있습니다. 정확한 동작은 클러스터에 대한 메시지 로드 밸런싱 정책이 구성된 방법에 따라 달라집니다.

추가 리소스

- 다음에 대한 자세한 내용은 다음을 수행합니다.
  - 정규화된 큐 이름 지정은 **4.9절. "정규화된 큐 이름 지정"** 에서 참조하십시오.
  - 브로커 클러스터에 대한 메시지 로드 밸런싱을 구성하는 방법은 **14.1.1절. "브로커 클러스터의 메시지 부하를 분산하는 방법"** 을 참조하십시오.

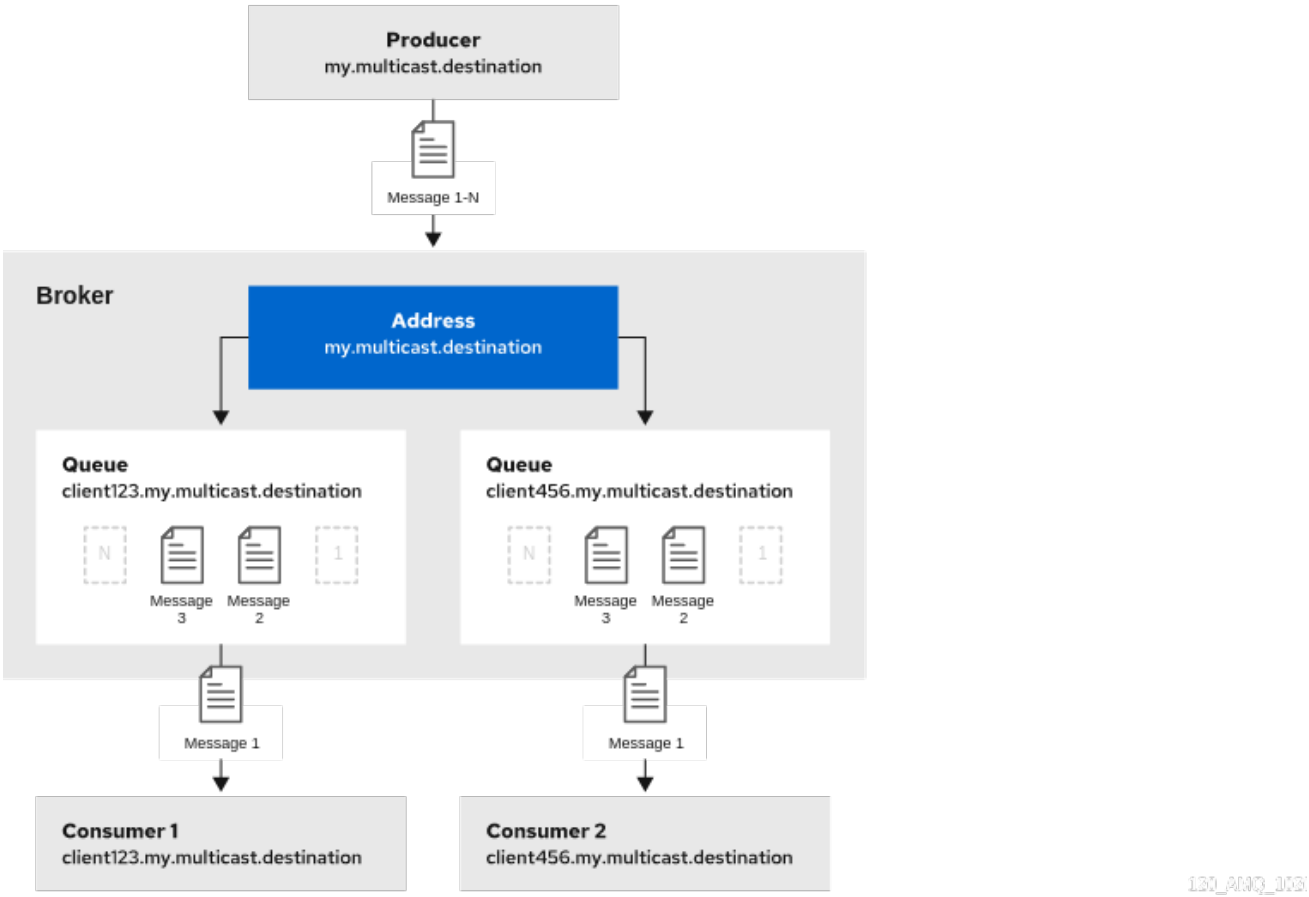
4.4. 게시-서브스크립션 메시징을 위한 주소 구성

게시-서브스크립션 시나리오에서는 주소가 등록된 모든 소비자에게 메시지가 전송됩니다. **JMS** 주제와 **MQTT** 서브스크립션은 게시-서브스크립션 메시징의 두 가지 예입니다. 게시-서브스크립션 방식으로 주소 수신 메시지와 연결된 큐가 브로커 구성에서 지정된 주소 요소에 대한 멀티 캐스트 라우팅 유형을 정의하도록 합니다.

멀티캐스트 라우팅 유형이 있는 주소에서 메시지를 수신하면 브로커는 해당 주소와 연결된 각 큐로 메시지 사본을 라우팅합니다. 복사 오버헤드를 줄이기 위해 각 큐는 전체 복사가 아닌 메시지에 대한 참조만 전송됩니다.

다음 그림은 게시-서브스크립션 메시징의 예를 보여줍니다.

그림 4.3. 게시-서브스크립션 메시징



다음 절차에서는 게시-서브스크립션 메시징의 주소를 구성하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** *구성 파일을 엽니다.*

2. *주소에 빈 멀티 캐스트 구성 요소를 추가합니다.*

```
<configuration ...>
  <core ...>
    ...
    <address name="my.multicast.destination">
      <multicast/>
    </address>
  </core>
</configuration>
```

3. *(선택 사항) 주소에 하나 이상의 큐 요소를 추가하고 멀티 캐스트 요소를 래핑합니다. 일반적으로 브로커는 클라이언트에서 요청한 각 서브스크립션에 대해 큐를 자동으로 생성하므로 이 단*

계가 필요하지 않습니다.

```
<configuration ...>
  <core ...>
    ...
    <address name="my.multicast.destination">
      <multicast>
        <queue name="client123.my.multicast.destination"/>
        <queue name="client456.my.multicast.destination"/>
      </multicast>
    </address>
  </core>
</configuration>
```

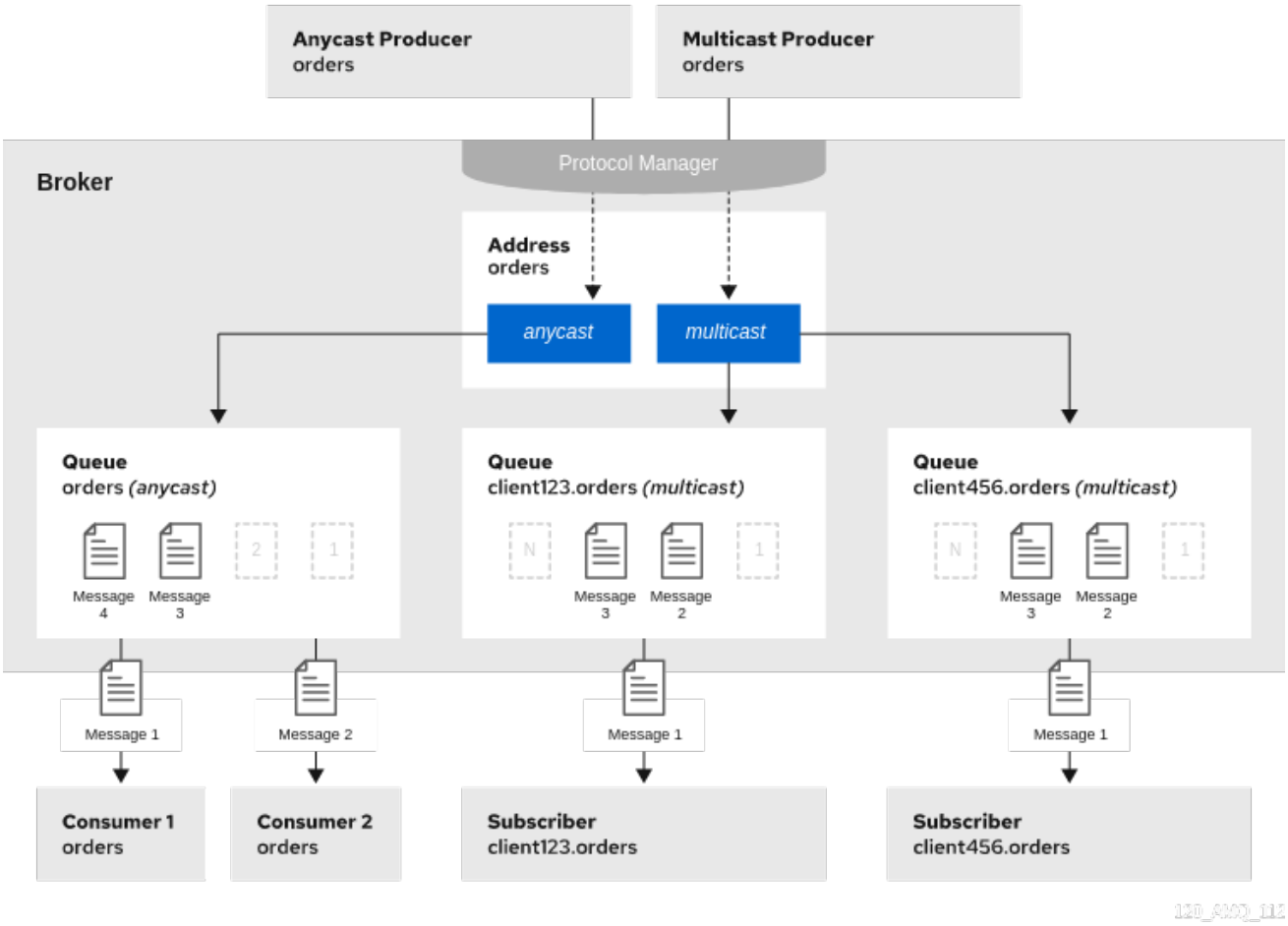
4.5. 지점 간 및 게시-서브스크립션 메시징의 주소 구성

point-to-point 및 **publish-subscribe** 의미 체계를 둘 다 사용하여 주소를 구성할 수도 있습니다.

point-to-point 및 **publish-subscribe** 의미 체계를 모두 사용하는 주소를 구성하는 것은 일반적으로 권장되지 않습니다. 그러나 **orders** 라는 **JMS** 큐와 **orders** 라는 **JMS** 주제도 원하는 경우 유용할 수 있습니다. 다양한 라우팅 유형을 사용하면 클라이언트 연결에 대해 주소가 고유하게 표시됩니다. 이 경우 **JMS** 큐 프로듀서에서 보낸 메시지는 **anycast** 라우팅 유형을 사용합니다. **JMS** 주제 프로듀서에서 보낸 메시지는 멀티캐스트 라우팅 유형을 사용합니다. **JMS** 주제 소비자가 브로커에 연결하면 자체 서브스크립션 큐에 연결됩니다. 그러나 **JMS** 대기열 소비자는 **anycast** 큐에 연결되어 있습니다.

다음 그림은 함께 사용되는 **point-to-point** 및 **publish-subscribe** 메시징의 예를 보여줍니다.

그림 4.4. point-to-point 및 publish-subscribe 메시징



다음 절차에서는 점대점 및 게시-서브스크립션 메시징 모두에 대해 주소를 구성하는 방법을 보여줍니다.

참고

*이 시나리오의 동작은 사용 중인 프로토콜에 따라 달라집니다. **JMS**의 경우 주제와 큐 생산자와 소비자 간에 명확한 구분이 있어 논리를 간단하게 만들 수 있습니다. **AMQP**와 같은 다른 프로토콜은 이러한 차이를 구분하지 않습니다. **AMQP**를 통해 전송되는 메시지는 **anycast** 및 **multicast** 및 **consumers** 기본값 둘 다에서 **anycast** 로 라우팅됩니다. 자세한 내용은 **3장. 네트워크 연결에서 메시징 프로토콜 구성**의 내용을 참조하십시오.*

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** *구성 파일을 엽니다.*

2. **address** *요소의 큐 요소 주위에* **anycast** *구성 요소를 래핑합니다. 예를 들면 다음과 같습니다.*

-

```
<configuration ...>
 <core ...>
  ...
  <address name="orders">
   <anycast>
    <queue name="orders"/>
   </anycast>
  </address>
 </core>
</configuration>
```

3. *주소에 빈 멀티 캐스트 구성 요소를 추가합니다.*

```
<configuration ...>
 <core ...>
  ...
  <address name="orders">
   <anycast>
    <queue name="orders"/>
   </anycast>
   <multicast/>
  </address>
 </core>
</configuration>
```

참고

일반적으로 브로커는 필요에 따라 서브스크립션 큐를 생성하므로 멀티캐스트 요소 내에 특정 큐 요소를 나열할 필요가 없습니다.

4.6. 허용되는 구성에 라우팅 유형 추가

*일반적으로 **anycast** 및 **multicast** 를 모두 사용하는 주소에서 메시지를 수신하면 **anycast** 대기열 중 하나가 메시지와 모든 멀티 캐스트 대기열을 수신합니다. 그러나 클라이언트는 주소에 연결할 때 특수 접두사를 지정하여 임의의 캐스트 또는 멀티 캐스트를 사용하여 연결할지 여부를 지정할 수 있습니다. 접두사는 브로커 구성의 수락자 **URL** 내에서 **anycastPrefix** 및 **multicastPrefix** 매개변수를 사용하여 지정된 사용자 지정 값입니다.*

다음 절차에서는 지정된 허용자에 대한 접두사를 구성하는 방법을 보여줍니다.

프로세스

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2.

*지정된 수락자의 경우 **anycast** 접두사를 구성하려면 구성된 **URL**에 **anycastPrefix** 를 추가합니다. 사용자 지정 값을 설정합니다. 예를 들면 다음과 같습니다.*

```
<configuration ...>
  <core ...>
    ...
      <acceptors>
        <!-- Acceptor for every supported protocol -->
        <acceptor name="artemis">tcp://0.0.0.0:61616?
protocols=AMQP;anycastPrefix=anycast://</acceptor>
      </acceptors>
    ...
  </core>
</configuration>
```

*이전 구성에 따라 **acceptor**는 **anycast** 접두사에 **anycast://** 를 사용하도록 구성되어 있습니다. 클라이언트가 **anycast** 큐 중 하나에 메시지를 보내야하는 경우 클라이언트 코드는 **anycast://<my.destination>/** 을 지정할 수 있습니다.*

3.

*지정된 수락자의 경우 멀티캐스트 접두사를 구성하려면 구성된 **URL**에 멀티 캐스트 접두사를 추가합니다. 사용자 지정 값을 설정합니다. 예를 들면 다음과 같습니다.*

```
<configuration ...>
  <core ...>
    ...
      <acceptors>
        <!-- Acceptor for every supported protocol -->
        <acceptor name="artemis">tcp://0.0.0.0:61616?
protocols=AMQP;multicastPrefix=multicast://</acceptor>
      </acceptors>
    ...
  </core>
</configuration>
```

*이전 구성을 기반으로 허용자는 멀티 캐스트 접두사에 **multicast://** 를 사용하도록 구성됩니다. 클라이언트에 멀티 캐스트 큐로만 전송된 메시지가 필요한 경우 클라이언트 코드는 **multicast://<my.destination>/** 을 지정할 수 있습니다.*

4.7. 서브스크립션 대기열 구성

*대부분의 경우 클라이언트가 먼저 주소에 가입하도록 요청할 때 프로토콜 관리자가 자동으로 서브스크립션 큐를 생성하므로 서브스크립션 대기열을 수동으로 생성할 필요가 없습니다. 자세한 내용은 **4.8.3**절. "프로토콜 관리자 및 주소"를 참조하십시오. **Cryostat** 서브스크립션의 경우 생성된 큐 이름은 일반적으로 클라이언트 **ID**와 주소를 연결합니다.*

다음 섹션에서는 필요한 경우 서브스크립션 대기열을 수동으로 생성하는 방법을 보여줍니다.

4.7.1. Cryostat 서브스크립션 큐 구성

큐가 **Cryostat** 서브스크립션으로 구성되면 브로커는 비활성 구독자에 대한 메시지를 저장하고 다시 연결할 때 구독자에게 전달합니다. 따라서 클라이언트는 구독한 후 큐에 전달된 각 메시지를 수신할 수 있습니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 선택한 큐에 **Cryostat** 구성 요소를 추가합니다. **true** 값을 설정합니다.

```
<configuration ...>
 <core ...>
  ...
  <address name="my.durable.address">
   <multicast>
    <queue name="q1">
     <durable>true</durable>
    </queue>
   </multicast>
  </address>
 </core>
</configuration>
```

참고

대기열은 **default** 요소를 포함하고 **value**를 **true** 로 설정하는 것은 기본적으로 내구성이 있으므로 큐를 만드는 데 엄격하게 필요하지는 않습니다. 그러나 명시적으로 요소를 포함 하면 나중에 필요한 경우 큐의 동작을 **non-durable**로 변경할 수 있습니다.

4.7.2. 공유 대상이 아닌 서브스크립션 대기열 구성

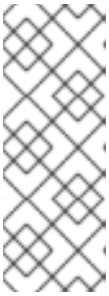
둘 이상의 소비자가 한 번에 큐에 연결하지 못하도록 브로커를 구성할 수 있습니다. 따라서 이러한 방식으로 구성된 큐에 대한 서브스크립션은 "비공유"로 간주됩니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 선택한 각 큐에 **Cryostat** 구성 요소를 추가합니다. **true** 값을 설정합니다.

```
<configuration ...>
 <core ...>
  ...
  <address name="my.non.shared.durable.address">
   <multicast>
    <queue name="orders1">
     <durable>true</durable>
    </queue>
    <queue name="orders2">
     <durable>true</durable>
    </queue>
   </multicast>
  </address>
 </core>
</configuration>
```



참고

대기열은 **default** 요소를 포함하고 **value**를 **true** 로 설정하는 것은 기본적으로 내구성이 있으므로 큐를 만드는 데 엄격하게 필요하지는 않습니다. 그러나 명시적으로 요소를 포함 하면 나중에 필요한 경우 큐의 동작을 **non-durable**로 변경할 수 있습니다.

3. 선택한 각 큐에 **max-consumers** 속성을 추가합니다. 값을 **1** 로 설정합니다.

```
<configuration ...>
 <core ...>
  ...
  <address name="my.non.shared.durable.address">
   <multicast>
    <queue name="orders1" max-consumers="1">
     <durable>true</durable>
    </queue>
    <queue name="orders2" max-consumers="1">
     <durable>true</durable>
    </queue>
   </multicast>
  </address>
 </core>
</configuration>
```

4.7.3. 실행 불가능한 서브스크립션 대기열 구성

실행 불가능한 서브스크립션은 일반적으로 임시 대기열을 생성하고 삭제하는 관련 프로토콜 관리자에 의해 관리됩니다.

그러나 실행 불가능한 서브스크립션 큐와 같이 작동하는 대기열을 수동으로 생성하려면 대기열에서 **purge-on-no-consumers** 속성을 사용할 수 있습니다. **purge-on-no-consumers** 를 **true** 로 설정하면 소비자가 연결될 때까지 큐에서 메시지를 수신하지 않습니다. 또한 마지막 소비자가 대기열에서 연결이 끊어지면 큐가 제거됩니다(즉, 해당 메시지가 제거됨). 새 소비자가 큐에 연결될 때까지 큐는 추가 메시지를 수신하지 않습니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 선택한 각 큐에 **purge-on-no-consumers** 속성을 추가합니다. **true** 값을 설정합니다.

```
<configuration ...>
  <core ...>
    ...
    <address name="my.non.durable.address">
        <multicast>
            <queue name="orders1" purge-on-no-consumers="true"/>
        </multicast>
    </address>
  </core>
</configuration>
```

4.8. 주소 및 큐를 자동으로 생성 및 삭제

브로커를 구성하여 자동으로 주소와 큐를 생성하고 더 이상 사용하지 않는 후 삭제할 수 있습니다. 이렇게 하면 클라이언트가 연결할 수 있기 전에 각 주소를 미리 구성하지 않아도 됩니다.

4.8.1. 자동 큐 생성 및 삭제를 위한 구성 옵션

다음 표에는 큐 및 주소를 자동으로 생성 및 삭제하도록 **address-setting** 요소를 구성할 때 사용할 수 있는 구성 요소가 나열되어 있습니다.

표 **4.3.** 큐 및 주소를 자동으로 생성 및 삭제하는 구성 요소

address-setting 을...로 원하는 경우	이 구성 추가...
클라이언트가 메시지를 전송하거나 존재하지 않는 주소로 매핑된 큐에서 메시지를 사용하려고 할 때 주소를 생성합니다.	**auto-create-addresses**
클라이언트가 큐로 메시지를 보내거나 큐에서 메시지를 사용하려고 할 때 큐를 생성합니다.	**auto-create-queues**
더 이상 큐가 없는 경우 자동으로 생성된 주소를 삭제합니다.	**auto-delete-addresses**
큐에 소비자 및 0개의 메시지가 있는 경우 자동으로 생성된 큐를 삭제합니다.	**auto-delete-queues**
클라이언트가 지정하지 않는 경우 특정 라우팅 유형을 사용합니다.	**default-address-routing-type**

4.8.2. 주소 및 큐의 자동 생성 및 삭제 구성

다음 절차에서는 주소 및 큐의 자동 생성 및 삭제를 구성하는 방법을 보여줍니다.

프로세스

1. *<broker_instance_dir>/etc/broker.xml 구성 파일을 엽니다.*

2. *자동 생성 및 삭제를 위해 address-setting 을 구성합니다. 다음 예제에서는 이전 표에 언급된 모든 구성 요소를 사용합니다.*

```
<configuration ...>
 <core ...>
  ...
  <address-settings>
   <address-setting match="activemq.#">
    <auto-create-addresses>true</auto-create-addresses>
    <auto-delete-addresses>true</auto-delete-addresses>
    <auto-create-queues>true</auto-create-queues>
    <auto-delete-queues>true</auto-delete-queues>
    <default-address-routing-type>ANYCAST</default-address-routing-type>
   </address-setting>
  </address-settings>
  ...
 </core>
</configuration>
```

address-setting

address-setting 요소의 구성은 와일드카드 주소 activemq.# 과 일치하는 모든 주소 또는 큐에 적용됩니다.

auto-create-addresses

클라이언트가 아직 존재하지 않는 주소에 연결을 요청하면 브로커는 주소를 생성합니다.

auto-delete-addresses

자동으로 생성된 주소는 더 이상 연결된 큐가 없는 경우 삭제됩니다.

auto-create-queues

클라이언트가 아직 존재하지 않는 큐에 연결하도록 요청하면 브로커는 큐를 생성합니다.

auto-delete-queues

자동으로 생성된 큐는 더 이상 소비자 또는 메시지가 없는 경우 삭제됩니다.

default-address-routing-type

연결할 때 클라이언트가 라우팅 유형을 지정하지 않으면 브로커는 주소에 메시지를 전달할 때 **ANYCAST** 를 사용합니다. 기본값은 **MULTICAST** 입니다.

추가 리소스

- 다음에 대한 자세한 내용은 다음을 수행합니다.
  - 주소를 구성할 때 사용할 수 있는 와일드카드 구문은 4.2절. “주소 세트에 주소 설정 적용” 를 참조하십시오.
  - 라우팅 유형: 4.1절. “주소, 큐 및 라우팅 유형” 을 참조하십시오.

4.8.3. 프로토콜 관리자 및 주소

프로토콜 관리자 라는 구성 요소는 프로토콜별 개념을 **AMQ Broker** 주소 모델, 큐 및 라우팅 유형에 사용되는 개념에 매핑합니다. 특정 상황에서 프로토콜 관리자가 브로커에 큐를 자동으로 생성할 수 있습니다.

예를 들어, 클라이언트가 주소 **/house/ room1/lights** 및 **/house/ room2/lights** 를 사용하여 **MQTT** 서브스크립션 패킷을 전송하는 경우, **MQTT** 프로토콜 관리자는 두 주소에 멀티캐스트 의미가 필요하다는 것을 이해하게 됩니다. 따라서 프로토콜 관리자는 먼저 두 주소에 대해 멀티캐스트 가 활성화되어 있는지

확인합니다. 그렇지 않은 경우 동적으로 생성하려고 합니다. 프로토콜 관리자가 성공하면 클라이언트에서 요청한 각 서브스크립션에 대해 특수 서브스크립션 대기열을 생성합니다.

각 프로토콜은 약간 다르게 작동합니다. 아래 표는 다양한 유형의 큐에 대한 구독 프레임이 요청될 때 일반적으로 발생하는 내용을 설명합니다.

표 4.4. 다양한 대기열 유형에 대한 프로토콜 관리자 작업

큐가 이 유형의 경우...	프로토콜 관리자의 일반적인 작업은...입니다.
Cryostat 서브스크립션 큐	적절한 주소를 찾고 **멀티캐스트** 의미 체계가 활성화되어 있는지 확인합니다. 그런 다음 클라이언트 ID를 사용하고 주소를 해당 라우팅 유형으로 **멀티 캐스트** 로 사용하여 특수 서브스크립션 큐를 생성합니다. 특수 이름을 사용하면 프로토콜 관리자가 클라이언트의 연결을 끊고 나중에 다시 연결할 때 필요한 클라이언트 서브스크립션 대기열을 신속하게 식별할 수 있습니다. 클라이언트가 구독을 취소하면 큐가 삭제됩니다.
임시 서브스크립션 대기열	적절한 주소를 찾고 **멀티캐스트** 의미 체계가 활성화되어 있는지 확인합니다. 그런 다음 **멀티캐스트** 라우팅 유형을 사용하여 이 주소 아래에 임의의 UUID(읽기 UUID) 이름을 사용하여 큐를 생성합니다. 클라이언트의 연결을 끊으면 큐가 삭제됩니다.
점대점 대기열	적절한 주소를 찾고 **anycast** 라우팅 유형이 활성화되어 있는지 확인합니다. 이 경우 주소와 동일한 이름으로 큐를 찾는 것을 목표로 합니다. 존재하지 않는 경우 사용 가능한 첫 번째 큐를 찾습니다. 존재하지 않는 경우 큐를 자동으로 생성합니다(자동 생성이 활성화됨). 큐 소비자는 이 큐에 바인딩됩니다. 큐가 자동으로 생성되면 소비자와 메시지가 없으면 자동으로 삭제됩니다.

4.9. 정규화된 큐 이름 지정

내부적으로 브로커는 주소에 대한 클라이언트 요청을 특정 큐에 매핑합니다. 브로커는 메시지를 보낼 대기열 또는 메시지를 수신하기 위한 큐를 대신하여 클라이언트를 결정합니다. 그러나 클라이언트가 대기열 이름을 직접 지정해야 하는 고급 사용 사례가 있을 수 있습니다. 이러한 상황에서 클라이언트는 정규화된 큐 이름 (FQQN)을 사용할 수 있습니다. FQQN에는 주소 이름과 큐 이름이 모두 포함됩니다. ::.

다음 절차에서는 여러 큐가 있는 주소에 연결할 때 FQQN을 지정하는 방법을 보여줍니다.

사전 요구 사항

- *아래 예제와 같이 두 개 이상의 큐로 구성된 주소가 있습니다.*

```
<configuration ...>
  <core ...>
    ...
    <addresses>
      <address name="my.address">
        <anycast>
          <queue name="q1" />
          <queue name="q2" />
        </anycast>
      </address>
    </addresses>
  </core>
</configuration>
```

프로세스

- *클라이언트 코드에서 브로커의 연결을 요청할 때 주소 이름과 큐 이름을 모두 사용합니다. 두 개의 콜론(::)을 사용하여 이름을 구분합니다. 예를 들면 다음과 같습니다.*

```
String FQQN = "my.address::q1";
Queue q1 session.createQueue(FQQN);
MessageConsumer consumer = session.createConsumer(q1);
```

4.10. 분할된 대기열 구성

*대기열의 부분적인 순서만 필요한 대기열에서 메시지 처리를 위한 일반적인 패턴은 대기열 샤딩 을 사용하는 것입니다. 즉, 단일 논리 대기열 역할을 하지만 여러 기본 물리적 대기열에서 지원하는 **anycast** 주소를 정의합니다.*

프로세스

1. *&lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.*

2. *address 요소를 추가하고 name 속성을 설정합니다. 예를 들면 다음과 같습니다.*

```
<configuration ...>
  <core ...>
    ...
    <addresses>
      <address name="my.sharded.address"></address>
    </addresses>
  </core>
</configuration>
```

3.

anycast 라우팅 유형을 추가하고 원하는 수의 **sharded** 큐를 포함합니다. 아래 예제에서는 **q1,q2** 및 **q3** 큐가 임의의 캐스트 대상으로 추가됩니다.

```
<configuration ...>
  <core ...>
    ...
    <addresses>
      <address name="my.sharded.address">
        <anycast>
          <queue name="q1" />
          <queue name="q2" />
          <queue name="q3" />
        </anycast>
      </address>
    </addresses>
</core>
</configuration>
```

이전 구성에 따라 **my.sharded.address** 로 전송된 메시지는 **q1,q2** 및 **q3** 에 걸쳐 동일하게 배포됩니다. 클라이언트는 **FQN(Fully Qualified Queue Name)**을 사용할 때 특정 물리적 큐에 직접 연결할 수 있으며 해당 대기열로만 전송된 메시지를 수신할 수 있습니다.

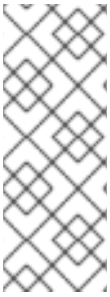
특정 메시지를 특정 큐에 연결하기 위해 클라이언트는 각 메시지에 대한 메시지 그룹을 지정할 수 있습니다. 브로커는 메시지를 동일한 큐로 그룹화하고 하나의 소비자는 모두 처리합니다.

추가 리소스

- 다음에 대한 자세한 내용은 다음을 수행합니다.
  - 정규화된 큐 이름, 참조 **4.9절. “정규화된 큐 이름 지정”**
  - 메시지 그룹화는 **AMQ Core Protocol JMS** 문서에서 메시지 그룹 사용을 참조하십시오.

4.11. 마지막 값 대기열 구성

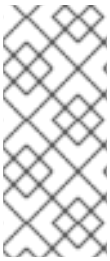
마지막 값 큐 는 마지막 값 키 값이 동일한 최신 메시지가 큐에 배치될 때 큐에서 메시지를 삭제하는 대기열 유형입니다. 이 동작을 통해 마지막 값 큐는 동일한 키의 메시지의 마지막 값만 유지합니다.



참고

큐로 전송된 메시지가 호출되면 마지막 값 대기열이 예상대로 작동하지 않습니다. 마지막 값 큐가 있는 주소의 **address-full-policy** 매개변수 값을 **DROP,BLOCK** 또는 **FAIL** 로 설정하여 이러한 큐로 전송된 메시지가 호출되지 않도록 합니다. 자세한 내용은 *7.2절. “메시지 삭제 구성”*의 내용을 참조하십시오.

마지막 값 큐의 간단한 사용 사례는 특정 주식에 대한 최신 값만 관심 있는 주식 가격을 모니터링하는 것입니다.



참고

구성된 마지막 값 키가 없는 메시지가 마지막 값 큐로 전송되면 브로커는 이 메시지를 "일반" 메시지로 처리합니다. 이러한 메시지는 구성된 마지막 값 키가 있는 새 메시지가 도달하는 경우 큐에서 제거되지 않습니다.

마지막 값 큐를 개별적으로 구성하거나 주소 집합과 연결된 모든 큐에 대해 구성할 수 있습니다.

다음 절차에서는 이러한 방식으로 마지막 값 큐를 구성하는 방법을 보여줍니다.

4.11.1. 마지막 값 대기열을 개별적으로 구성

다음 절차에서는 마지막 값 큐를 개별적으로 구성하는 방법을 보여줍니다.

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 지정된 큐의 경우 **last-value-key** 키를 추가하고 사용자 지정 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<address name="my.address">
   <multicast>
      <queue name="prices1" last-value-key="stock_ticker"/>
   </multicast>
</address>
```

3. 또는 **_AMQ_LVQ_NAME** 의 기본 마지막 값 키 이름을 사용하는 마지막 값 큐를 구성할 수 있습니다. 이렇게 하려면 지정된 큐에 **last-value** 키를 추가합니다. 값을 **true** 로 설정합니다. 예를

들면 다음과 같습니다.

```
<address name="my.address">
   <multicast>
      <queue name="prices1" last-value="true"/>
   </multicast>
</address>
```

4.11.2. 주소에 대한 마지막 값 대기열 구성

다음 절차에서는 주소 또는 주소 집합에 대한 마지막 값 큐를 구성하는 방법을 보여줍니다.

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **address-setting** 요소에서 일치하는 주소에 대해 **default-last-value-key** 를 추가합니다. 사용자 지정 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<address-setting match="lastValue">
  <default-last-value-key>stock_ticker</default-last-value-key>
</address-setting>
```

이전 구성에 따라 **lastValue** 주소와 연결된 모든 대기열은 **possibility _ticker** 의 마지막 값 키를 사용합니다. 기본적으로 **default-last-value-key** 값은 설정되지 않습니다.

3. 주소 집합에 대한 마지막 값 큐를 구성하려면 주소 와일드카드를 지정할 수 있습니다. 예를 들면 다음과 같습니다.

```
<address-setting match="lastValue.*">
  <default-last-value-key>stock_ticker</default-last-value-key>
</address-setting>
```

4. 또는 주소 또는 주소 집합과 연결된 모든 대기열을 **_AMQ_LVQ_NAME** 의 기본 마지막 값 키 이름을 사용하도록 구성할 수 있습니다. 이렇게 하려면 **default-last-value-key** 대신 **default-last-value-queue** 를 추가합니다. 값을 **true** 로 설정합니다. 예를 들면 다음과 같습니다.

```
<address-setting match="lastValue">
  <default-last-value-queue>true</default-last-value-queue>
</address-setting>
```

추가 리소스

- 주소를 구성할 때 사용할 수 있는 와일드카드 구문에 대한 자세한 내용은 4.2절. “주소 세트에 주소 설정 적용” 을 참조하십시오.

4.11.3. 마지막 값 큐 동작의 예

이 예에서는 마지막 값 큐의 동작을 보여줍니다.

broker.xml 구성 파일에서 다음과 같은 구성을 추가했다고 가정합니다.

```
<address name="my.address">
   <multicast>
      <queue name="prices1" last-value-key="stock_ticker"/>
   </multicast>
</address>
```

앞의 구성에서는 **price1** 이라는 큐를 생성하고 마지막 값 키는 **sec _ticker** 를 생성합니다.

클라이언트가 두 개의 메시지를 전송한다고 가정합니다. 각 메시지의 속성 **sec _ticker** 에 대해 **ATN** 의 값이 동일합니다. 각 메시지에는 **price_ price** 라는 속성에 대해 다른 값이 있습니다. 각 메시지는 동일한 큐, **price1** 로 전송됩니다.

```
TextMessage message = session.createTextMessage("First message with last value property
set");
message.setStringProperty("stock_ticker", "ATN");
message.setStringProperty("stock_price", "36.83");
producer.send(message);
```

```
TextMessage message = session.createTextMessage("Second message with last value
property set");
message.setStringProperty("stock_ticker", "ATN");
message.setStringProperty("stock_price", "37.02");
producer.send(message);
```

score _ticker 마지막 값 키(이 경우 **ATN**)에 대해 동일한 값을 가진 두 개의 메시지가 **price1** 큐에 도착하면 첫 번째 메시지가 삭제되고 최신 메시지만 큐에 남아 있습니다. 명령줄에서 다음 행을 입력하여 이 동작을 검증할 수 있습니다.

```
TextMessage messageReceived = (TextMessage)messageConsumer.receive(5000);
System.out.format("Received message: %s\n", messageReceived.getText());
```

이 예에서는 두 번째 메시지 모두 마지막 값 키에 대해 동일한 값을 사용하고 두 번째 메시지가 첫 번째

메시지 뒤에 큐에서 수신되었으므로 표시되는 출력은 두 번째 메시지입니다.

4.11.4. 마지막 값 큐에 대해 비차단적 소비 적용

소비자가 큐에 연결하면 일반 동작은 해당 소비자에게 전송된 메시지가 소비자가 독점적으로 취득한다는 것입니다. 소비자가 메시지 수신을 승인하면 브로커는 대기열에서 메시지를 제거합니다.

일반 사용 동작 대신 중단 되지 않은 소비를 적용하도록 큐를 구성할 수 있습니다. 이 경우 대기열에서 메시지를 소비자에게 보내면 다른 소비자가 계속 메시지를 수신할 수 있습니다. 또한 메시지는 소비자가 사용한 경우에도 큐에 남아 있습니다. 이러한 비영향 소비 동작을 적용하면 소비자를 대기열 브라우저 라고 합니다.

비분명적 소비를 강제하는 것은 마지막 값 큐에 유용한 구성입니다. 이는 큐가 항상 특정 마지막 값 키에 대한 최신 값을 가지도록 하기 때문입니다.

다음 절차에서는 마지막 값 큐에 대해 비차단 소비를 적용하는 방법을 보여줍니다.

사전 요구 사항

- 이미 주소 또는 주소 집합과 연결된 모든 큐에 대해 마지막 값 대기열을 개별적으로 구성했습니다. 자세한 내용은 다음을 참조하십시오.
  - 4.11.1절. “마지막 값 대기열을 개별적으로 구성”
  - 4.11.2절. “주소에 대한 마지막 값 대기열 구성”

프로세스

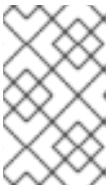
1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.
2. 이전에 큐를 마지막 값 큐로 개별적으로 구성한 경우 비분명 키를 추가합니다. 값을 **true** 로 설정합니다. 예를 들면 다음과 같습니다.

```
<address name="my.address">
  <multicast>
    <queue name="orders1" last-value-key="stock_ticker" non-destructive="true" />
```

```
    </multicast>
  </address>
```

3. 이전에 마지막 값 큐에 대한 주소 또는 주소 세트를 구성한 경우 **default-non-destructive** 키를 추가합니다. 값을 **true** 로 설정합니다. 예를 들면 다음과 같습니다.

```
<address-setting match="lastValue">
  <default-last-value-key>stock_ticker </default-last-value-key>
  <default-non-destructive>true</default-non-destructive>
</address-setting>
```



참고

기본적으로 **default-non-destructive** 의 값은 **false** 입니다.

4.12. 만료된 메시지를 만료 주소로 이동

마지막 값 큐 이외의 큐의 경우 파괴적이지 않은 소비자만 있는 경우 브로커는 대기열에서 메시지를 삭제하지 않으므로 큐 크기가 시간이 지남에 따라 증가합니다. 큐 크기가 제한되지 않은 이러한 증가를 방지하려면 메시지가 만료되는 시기를 구성하고 브로커가 만료된 메시지를 이동하는 주소를 지정할 수 있습니다.

4.12.1. 메시지 만료 구성

다음 절차에서는 메시지 만료를 구성하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **core** 요소에서 **message-expiry-scan-period** 를 설정하여 브로커가 만료된 메시지를 검색하는 빈도를 지정합니다.

```
<configuration ...>
  <core ...>
    ...
    <message-expiry-scan-period>1000</message-expiry-scan-period>
    ...
```

이전 구성을 기반으로 브로커는 **1000** 밀리초마다 만료된 메시지의 대기열을 검색합니다.

3. 일치하는 주소 또는 주소 집합에 대한 **address-setting** 요소에서 만료 주소를 지정합니다. 또한 메시지 만료 시간을 설정합니다. 예를 들면 다음과 같습니다.

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="stocks">
        ...
        <expiry-address>ExpiryAddress</expiry-address>
        <expiry-delay>10</expiry-delay>
        ...
      </address-setting>
      ...
    <address-settings>
<configuration ...>
```

expiry-address

일치하는 주소 또는 주소에 대한 만료 주소입니다. 이전 예에서 브로커는 주식 주소에 대한 만료된 메시지를 **ExpiryAddress** 라는 만료 주소로 보냅니다.

expiry-delay

브로커가 기본 만료 시간을 사용하는 메시지에 적용되는 만료 시간(밀리초)입니다. 기본적으로 메시지에는 만료 시간이 **0** 이며 이는 만료되지 않습니다. 만료 시간이 기본값보다 큰 메시지의 경우 **expiration -delay** 는 적용되지 않습니다.

예를 들어 이전 예제와 같이 주소에 **expiry-delay** 를 **10** 으로 설정했다고 가정합니다. 기본 만료 시간이 **0** 인 메시지가 이 주소의 큐에 도달하는 경우 브로커는 메시지의 만료 시간을 **0** 에서 **10** 으로 변경합니다. 그러나 만료 시간 **20** 을 사용하는 다른 메시지가 도달하는 경우 만료 시간은 변경되지 않습니다. **expiry-delay**를 **-1** 로 설정하면 이 기능이 비활성화됩니다. 기본적으로 **expiry-delay** 는 **-1** 로 설정됩니다.

4. 또는 **expiry-delay** 의 값을 지정하는 대신 최소 및 최대 만료 지연 값을 지정할 수 있습니다. 예를 들면 다음과 같습니다.

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="stocks">
```

```
        ...
        <expiry-address>ExpiryAddress</expiry-address>
        <min-expiry-delay>10</min-expiry-delay>
        <max-expiry-delay>100</max-expiry-delay>
        ...
      </address-setting>
      ...
    <address-settings>
<configuration ...>
```

min-expiry-delay

브로커가 메시지에 적용되는 최소 만료 시간(밀리초)입니다.

max-expiry-delay

브로커가 메시지에 적용되는 최대 만료 시간(밀리초)입니다.

*브로커는 다음과 같이 **min-expiry-delay** 및 **max-expiry-delay** 값을 적용합니다.*

- *기본 만료 시간이 **0** 인 메시지의 경우 브로커는 만료 시간을 **max-expiry-delay** 의 지정된 값으로 설정합니다. **max-expiry-delay** 의 값을 지정하지 않은 경우 브로커는 만료 시간을 **min-expiry-delay** 의 지정된 값으로 설정합니다. **min-expiry-delay** 에 대한 값을 지정하지 않은 경우 브로커는 메시지의 만료 시간을 변경하지 않습니다.*
- ***max-expiry-delay** 이상의 만료 시간이 있는 메시지의 경우 브로커는 만료 시간을 **max-expiry-delay** 의 지정된 값으로 설정합니다.*
- ***min-expiry-delay** 의 값보다 낮은 메시지의 경우 브로커는 만료 시간을 **min-expiry-delay** 의 지정된 값으로 설정합니다.*
- ***min-expiry-delay** 값과 **max-expiry-delay** 사이의 만료가 있는 메시지의 경우 브로커는 메시지의 만료 시간을 변경하지 않습니다.*
- ***expiry-delay** (즉, 기본값 **-1**이외의 값)에 대한 값을 지정하면 **min-expiry-delay** 및 **max-expiry-delay** 에 대해 지정하는 모든 값을 덮어씁니다.*
- ***min-expiry-delay** 및 **max-expiry-delay** 의 기본값은 **-1** (즉, 비활성화됨)입니다.*

5.

구성 파일의 **address** 요소에서 이전에 **expiry-address** 에 지정된 주소를 구성합니다. 이 주소에 큐를 정의합니다. 예를 들면 다음과 같습니다.

```
<addresses>
  ...
  <address name="ExpiryAddress">
    <anycast>
      <queue name="ExpiryQueue"/>
    </anycast>
  </address>
  ...
</addresses>
```

이전 예제 구성은 만료 큐인 **ExpiryQueue** 를 만료 주소인 **ExpiryAddress** 와 연결합니다.

4.12.2. 만료 리소스 자동 생성

일반적인 사용 사례는 원래 주소에 따라 만료된 메시지를 분리하는 것입니다. 예를 들어 주식 이라는 주소에서 만료된 메시지를 **EXP.stocks** 라는 만료 큐로 라우팅하도록 선택할 수 있습니다. 마찬가지로 주문이라는 주소에서 만료된 메시지를 **EXP. orders** 라는 만료 큐로 라우팅할 수 있습니다.

이러한 유형의 라우팅 패턴을 사용하면 만료된 메시지를 쉽게 추적, 검사 및 관리할 수 있습니다. 그러나 이러한 패턴은 주로 자동으로 생성된 주소와 큐를 사용하는 환경에서 구현하기 어렵습니다. 이러한 유형의 환경에서는 관리자가 만료된 메시지를 유지하기 위해 주소와 큐를 수동으로 생성하는 데 필요한 추가 노력을 기울이지 않습니다.

해결 방법으로 지정된 주소 또는 주소 집합에 대해 만료된 메시지를 처리하기 위해 자동으로 리소스(즉, 주소 및 대기열)를 생성하도록 브로커를 구성할 수 있습니다. 다음 절차에서는 예제를 보여줍니다.

사전 요구 사항

- 지정된 주소 또는 주소 집합에 대한 만료 주소를 이미 구성했습니다. 자세한 내용은 4.12.1절. “메시지 만료 구성”의 내용을 참조하십시오.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 이전에 구성 파일에 추가한 **& lt;address-setting >** 요소를 찾아 일치하는 주소 또는 주소 집합에 대한 만료 주소를 정의합니다. 예를 들면 다음과 같습니다.

```
<configuration ...>

  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="stocks">
        ...
        <expiry-address>ExpiryAddress</expiry-address>
        ...
      </address-setting>
      ...
    <address-settings>
<configuration ...>
```

3.

< address-setting > 요소에서 브로커에 만료 리소스(즉, 주소 및 큐)를 자동으로 생성하도록 지시하는 구성 항목과 이러한 리소스의 이름을 지정하는 방법을 추가합니다. 예를 들면 다음과 같습니다.

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="stocks">
        ...
        <expiry-address>ExpiryAddress</expiry-address>
        <auto-create-expiry-resources>true</auto-create-expiry-resources>
        <expiry-queue-prefix>EXP.</expiry-queue-prefix>
        <expiry-queue-suffix></expiry-queue-suffix>
        ...
      </address-setting>
      ...
    <address-settings>
<configuration ...>
```

auto-create-expiry-resources

브로커가 만료된 메시지를 수신하기 위해 만료 주소와 큐를 자동으로 생성하는지 여부를 지정합니다. 기본값은 **false**입니다.

매개변수 값이 **true** 로 설정되면 브로커는 만료 주소 및 관련 만료 큐를 정의하는 **<address>** 요소를 자동으로 생성합니다. 자동으로 생성된 **< address>** 요소의 **name** 값은 에 **<expiry-address>** 지정된 **name** 값과 일치합니다.

자동으로 생성된 만료 큐에는 멀티캐스트 라우팅 유형이 있습니다. 기본적으로 브로커는 만료된 메시지가 원래 전송된 주소와 일치하도록 만료 큐의 이름을 지정합니다 (예: 주식).

브로커는 **_AMQ_ORIG_ADDRESS** 속성을 사용하는 만료 대기열에 대한 필터도 정의합니다. 이 필터를 사용하면 만료 큐가 해당 원래 주소로 전송된 메시지만 수신합니다.

expiry-queue-prefix

브로커가 자동으로 생성된 만료 큐의 이름에 적용되는 접두사입니다. 기본값은 **EXP**입니다.

접두사 값을 정의하거나 기본값을 유지하는 경우 만료 큐의 이름은 접두사 및 원래 주소(예: **EXP.stocks**)입니다.

expiry-queue-suffix

브로커가 자동으로 생성된 만료 대기열 이름에 적용되는 접미사입니다. 기본값은 정의되지 않습니다(즉, 브로커는 접미사를 적용하지 않음).

자체적으로 큐 이름(예: **AMQ Broker Core Protocol JMS** 클라이언트를 사용하는 경우) 또는 정규화된 큐 이름(예: 다른 **JMS** 클라이언트를 사용하는 경우)을 사용하여 만료 대기열에 직접 액세스할 수 있습니다.

참고

만료 주소 및 큐가 자동으로 생성되므로 자동으로 생성된 주소 및 큐 삭제와 관련된 주소 설정이 이러한 만료 리소스에도 적용됩니다.

추가 리소스

- 자동으로 생성된 주소 및 큐의 자동 삭제를 구성하는 데 사용되는 주소 설정에 대한 자세한 내용은 4.8.2절. “주소 및 큐의 자동 생성 및 삭제 구성” 을 참조하십시오.

4.13. DEAD LETTER ADDRESS로 전달되지 않은 메시지 이동

클라이언트에 메시지를 전달하지 못하면 브로커가 메시지를 지속적으로 전송하려고 시도하지 않을 수 있습니다. 무한 전송 시도를 방지하기 위해 **dead letter address** 와 하나 이상의 **asscociated dead letter** 큐 를 정의할 수 있습니다. 지정된 수의 전달 시도 후 브로커는 원래 큐에서 전달되지 않은 메시지를 제거하고 구성된 **dead letter** 주소로 메시지를 보냅니다. 시스템 관리자는 나중에 **dead letter queue**에서 전달되지 않은 메시지를 사용하여 메시지를 검사할 수 있습니다.

지정된 큐에 대해 **dead letter address**를 구성하지 않으면 브로커는 지정된 수의 전달 시도 후 큐에서 전달되지 않은 메시지를 영구적으로 제거합니다.

dead letter 큐에서 사용되는 전달되지 않은 메시지의 속성은 다음과 같습니다.

_AMQ_ORIG_ADDRESS

메시지의 원래 주소를 지정하는 string 속성

_AMQ_ORIG_QUEUE

메시지의 원래 큐를 지정하는 문자열 속성

4.13.1. dead letter address 구성

다음 절차에서는 dead letter address 및 관련 dead letter 큐를 구성하는 방법을 보여줍니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. 큐 이름과 일치하는 < address-setting > 요소에서 dead letter address name 및 최대 전달 시도 수를 설정합니다. 예를 들면 다음과 같습니다.

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="exampleQueue">
        <dead-letter-address>DLA</dead-letter-address>
        <max-delivery-attempts>3</max-delivery-attempts>
      </address-setting>
    ...
    <address-settings>
<configuration ...>
```

match

브로커가 이 address-setting 섹션에서 구성을 적용하는 주소입니다. < address-setting > 요소의 match 특성에 와일드카드 표현식을 지정할 수 있습니다. 와일드카드 표현식은 < address-setting > 요소에 구성된 dead letter 설정을 일치하는 주소 세트와 연결하려는 경우에 유용합니다.

dead-letter-address

dead letter address의 이름입니다. 이 예에서 브로커는 큐 exampleQueue 에서 dead

letter 주소 DLA 로 전달되지 않은 메시지를 이동합니다.

max-delivery-attempts

전달되지 않은 메시지를 구성된 dead letter 주소로 이동하기 전에 브로커가 수행한 최대 전달 시도 수입니다. 이 예에서 브로커는 전달 시도가 실패한 세 번의 배달 시도 후 전달되지 않은 메시지를 dead letter address로 이동합니다. 기본값은 10 입니다. 브로커가 무한 수의 재전송 시도를 만들려면 값 -1 을 지정합니다.

3. *address 섹션에서 dead letter address 인 DLA 의 address 요소를 추가합니다. dead letter 큐를 dead letter 주소와 연결하려면 큐 의 name 값을 지정합니다. 예를 들면 다음과 같습니다.*

```
<configuration ...>
  <core ...>
    ...
    <addresses>
      <address name="DLA">
        <anycast>
          <queue name="DLQ" />
        </anycast>
      </address>
    ...
    </addresses>
  </core>
</configuration>
```

이전 구성에서 DLQ 라는 dead letter 큐를 dead letter address인 DLA 와 연결합니다.

추가 리소스

- *주소 설정에서 와일드카드를 사용하는 방법에 대한 자세한 내용은 4.2절. "주소 세트에 주소 설정 적용" 을 참조하십시오.*

4.13.2. dead letter queue automatically 생성

일반적인 사용 사례는 원래 주소에 따라 전달되지 않은 메시지를 분리하는 것입니다. 예를 들어 주식이라는 주소에서 전달되지 않은 메시지를 DLQ.stocks 라는 연결된 dead letter 큐가 있는 DLA.stocks 라는 dead letter queue로 라우팅하도록 선택할 수 있습니다. 마찬가지로 주문이라는 주소에서 전달되지 않은 메시지를 DLA. orders 라는 dead letter 주소로 라우팅할 수 있습니다.

이러한 유형의 라우팅 패턴을 사용하면 전달되지 않은 메시지를 쉽게 추적, 검사 및 관리할 수 있습니다. 그러나 이러한 패턴은 주로 자동으로 생성된 주소와 큐를 사용하는 환경에서 구현하기 어렵습니다. 이

러한 유형의 환경에 대한 시스템 관리자는 전송되지 않은 메시지를 유지하기 위해 수동으로 주소와 큐를 생성하는 데 필요한 추가 노력을 원하지 않을 수 있습니다.

해결 방법으로 다음 절차에 표시된 대로 전달되지 않은 메시지를 처리하기 위해 수신자와 큐를 자동으로 생성하도록 브로커를 구성할 수 있습니다.

사전 요구 사항

- 큐 또는 큐 집합에 대한 **dead letter** 주소를 이미 구성했습니다. 자세한 내용은 **4.13.1절. "dead letter address 구성"**의 내용을 참조하십시오.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 일치하는 큐 또는 큐 집합에 대한 **dead letter** 주소를 정의하기 위해 이전에 추가한 **<address-setting >** 요소를 찾습니다. 예를 들면 다음과 같습니다.

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="exampleQueue">
        <dead-letter-address>DLA</dead-letter-address>
        <max-delivery-attempts>3</max-delivery-attempts>
      </address-setting>
    ...
    <address-settings>
<configuration ...>
```

3. **< address-setting >** 요소에서 브로커에 **dead letter** 리소스(즉, 주소 및 큐)를 자동으로 생성하도록 지시하는 구성 항목과 이러한 리소스의 이름을 지정하는 방법을 추가합니다. 예를 들면 다음과 같습니다.

```
<configuration ...>
  <core ...>
    ...
    <address-settings>
      ...
      <address-setting match="exampleQueue">
        <dead-letter-address>DLA</dead-letter-address>
        <max-delivery-attempts>3</max-delivery-attempts>
        <auto-create-dead-letter-resources>true</auto-create-dead-letter-resources>
```

```
        <dead-letter-queue-prefix>DLQ.</dead-letter-queue-prefix>
        <dead-letter-queue-suffix></dead-letter-queue-suffix>
      </address-setting>
    ...
    <address-settings>
<configuration ...>
```

auto-create-dead-letter-resources

브로커가 전달되지 않은 메시지를 수신하기 위해 **dead letter address** 및 **queue**를 자동으로 생성하는지 여부를 지정합니다. 기본값은 **false**입니다.

auto-create-dead-letter-resources 가 **true** 로 설정된 경우 브로커는 **dead letter address** 및 관련 **dead letter** 큐를 정의하는 **<address>** 요소를 자동으로 생성합니다. 자동으로 생성된 **< address >** 요소의 이름은 **< dead-letter-address>**에 지정하는 **name** 값과 일치합니다.

브로커가 자동으로 생성된 **< address >** 요소에서 정의하는 **dead letter** 큐에는 멀티캐스트 라우팅 유형이 있습니다. 기본적으로 브로커는 배달되지 않은 메시지의 원래 주소와 일치하도록 **dead letter** 큐의 이름을 지정합니다(예: 주식).

브로커는 **_AMQ_ORIG_ADDRESS** 속성을 사용하는 **dead letter queue**에 대한 필터도 정의합니다. 이 필터를 사용하면 **dead letter** 큐가 해당 원래 주소로 전송된 메시지만 수신합니다.

dead-letter-queue-prefix

브로커가 자동으로 생성된 **dead letter** 큐의 이름에 적용되는 접두사입니다. 기본값은 **DLQ**입니다.

접두사 값을 정의하거나 기본값을 유지하는 경우 **dead letter** 큐의 이름은 접두사와 원래 주소(예: **DLQ.stocks**)의 연결입니다.

dead-letter-queue-suffix

브로커가 자동으로 생성된 **dead letter** 큐에 적용되는 접미사입니다. 기본값은 정의되지 않습니다(즉, 브로커는 접미사를 적용하지 않음).

4.14. 만료된 또는 전달되지 않은 AMQP 메시지의 주석 및 속성

브로커가 만료된 또는 전달되지 않은 **AMQP** 메시지를 구성한 만료 또는 **dead letter** 큐로 이동하기 전에 브로커는 주석과 속성을 메시지에 적용합니다. 클라이언트는 이러한 속성 또는 주석을 기반으로 필터를 생성하여 만료 또는 배달 못 한 큐에서 사용할 특정 메시지를 선택할 수 있습니다.

참고

브로커가 적용되는 속성은 내부 속성입니다. 이러한 속성은 정기적으로 사용하기 위해 클라이언트에 노출되지 않지만 필터의 클라이언트에서 지정할 수 있습니다.

다음 표는 브로커가 만료된 AMQP 메시지 또는 전달되지 않은 AMQP 메시지에 적용되는 주석 및 내부 속성을 보여줍니다.

표 4.5. 만료된 또는 전달되지 않은 AMQP 메시지에 적용되는 주석 및 속성

주석 이름	내부 속성 이름	설명
x-opt-ORIG-MESSAGE-ID	_AMQ_ORIG_MESSAGE_ID	메시지가 만료 또는 배달 못 한 큐로 이동하기 전에 원본 메시지 ID입니다.
x-opt-ACTUAL-EXPIRY	_AMQ_ACTUAL_EXPIRY	마지막 epoch가 시작된 이후의 시간(밀리초)으로 지정된 메시지 만료 시간입니다.
x-opt-ORIG-QUEUE	_AMQ_ORIG_QUEUE	만료된 또는 전달되지 않은 메시지의 원래 큐 이름입니다.
x-opt-ORIG-ADDRESS	_AMQ_ORIG_ADDRESS	만료된 또는 전달되지 않은 메시지의 원래 주소 이름입니다.

추가 리소스

- *주석에 따라 AMQP 메시지를 필터링하도록 AMQP 클라이언트를 구성하는 예는 13.3절. "주석에 속성을 기반으로 AMQP 메시지 필터링" 를 참조하십시오.*

4.15. 대기열 비활성화

브로커 구성에서 큐를 수동으로 정의하는 경우 큐는 기본적으로 활성화됩니다.

그러나 클라이언트가 구독할 수 있도록 큐를 정의하려고 하지만 메시지 라우팅에 큐를 사용할 준비가 되지 않은 경우가 있을 수 있습니다. 또는 큐로 메시지 흐름을 중지하려고 하지만 클라이언트가 큐에 바인딩된 상태로 유지하려는 상황이 있을 수 있습니다. 이러한 경우 큐를 비활성화할 수 있습니다.

다음 예제에서는 브로커 구성에 정의된 큐를 비활성화하는 방법을 보여줍니다.

사전 요구 사항

- 브로커 구성에서 주소 및 관련 큐를 정의하는 방법에 대해 잘 알고 있어야 합니다. 자세한 내용은 4장. 주소 및 큐 구성의 내용을 참조하십시오.

프로세스

1. <broker_instance_dir>/etc/broker.xml 구성 파일을 엽니다.

2. 이전에 정의한 큐의 경우 **enabled** 속성을 추가합니다. 큐를 비활성화하려면 이 속성의 값을 **false** 로 설정합니다. 예를 들면 다음과 같습니다.

```
<addresses>
  <address name="orders">
    <multicast>
      <queue name="orders" enabled="false"/>
    </multicast>
  </address>
</addresses>
```

활성화된 속성의 기본값은 **true** 입니다. 값을 **false** 로 설정하면 큐로 메시지 라우팅이 비활성화됩니다.

참고

주소의 모든 대기열을 비활성화하면 해당 주소로 전송된 모든 메시지가 자동으로 삭제됩니다.

4.16. 큐에 연결된 소비자 수 제한

max-consumers 특성을 사용하여 특정 큐에 연결된 소비자 수를 제한합니다. **max-consumers** 플래그를 **1** 로 설정하여 배타적 소비자를 생성합니다. 기본값은 **-1** 이며, 이는 무제한 소비자 수를 설정합니다.

다음 절차에서는 큐에 연결할 수 있는 소비자 수에 제한을 설정하는 방법을 보여줍니다.

프로세스

1. <broker_instance_dir>/etc/broker.xml 구성 파일을 엽니다.

2. 지정된 큐의 경우 **max-consumers** 키를 추가하고 값을 설정합니다.

```
<configuration ...>
  <core ...>
    ...
    <addresses>
      <address name="my.address">
        <anycast>
          <queue name="q3" max-consumers="20"/>
        </anycast>
      </address>
    </addresses>
  </core>
</configuration>
```

이전 구성을 기반으로 **20** 소비자만 동시에 대기열 **q3** 에 연결할 수 있습니다.

3. 전용 소비자를 생성하려면 **max-consumers** 를 **1** 로 설정합니다.

```
<configuration ...>
  <core ...>
    ...
    <address name="my.address">
      <anycast>
        <queue name="q3" max-consumers="1"/>
      </anycast>
    </address>
  </core>
</configuration>
```

4. 무제한 소비자 수를 허용하려면 **max-consumers** 를 **-1** 로 설정합니다.

```
<configuration ...>
  <core ...>
    ...
    <address name="my.address">
      <anycast>
        <queue name="q3" max-consumers="-1"/>
      </anycast>
    </address>
  </core>
</configuration>
```

4.17. 전용 대기열 구성

배타적 대기열은 모든 메시지를 한 번에 하나의 소비자로만 라우팅하는 특수 대기열입니다. 이 구성은 동일한 소비자가 모든 메시지를 직렬로 처리하려는 경우에 유용합니다. 큐에 대한 소비자가 여러 개 있는 경우 하나의 소비자만 메시지를 수신합니다. 해당 소비자가 큐에서 연결을 끊으면 다른 소비자가 선택됩니다.

4.17.1. 개별적으로 전용 대기열 구성

다음 절차에서는 지정된 큐를 배타적으로 구성하는 방법을 보여줍니다.

프로세스

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. *지정된 큐의 경우 전용 키를 추가합니다. 값을 **true** 로 설정합니다.*

```
<configuration ...>
 <core ...>
  ...
  <address name="my.address">
   <multicast>
    <queue name="orders1" exclusive="true"/>
   </multicast>
  </address>
 </core>
</configuration>
```

4.17.2. 주소에 대한 전용 대기열 구성

다음 절차에서는 연결된 모든 큐가 배타적이므로 주소 또는 주소 집합 을 구성하는 방법을 보여줍니다.

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. ***address-setting** 요소에서 일치하는 주소에 대해 **default-exclusive-queue** 키를 추가합니다. 값을 **true** 로 설정합니다.*

```
<address-setting match="myAddress">
   <default-exclusive-queue>true</default-exclusive-queue>
</address-setting>
```

*이전 구성을 기반으로 **myAddress** 주소와 연결된 모든 대기열은 배타적입니다. 기본적으로*

default-exclusive-queue 값은 false 입니다.

3. 주소 집합에 대한 전용 대기열을 구성하려면 주소 와일드카드를 지정할 수 있습니다. 예를 들면 다음과 같습니다.

```
<address-setting match="myAddress.*">
   <default-exclusive-queue>true</default-exclusive-queue>
</address-setting>
```

추가 리소스

- 주소를 구성할 때 사용할 수 있는 와일드카드 구문에 대한 자세한 내용은 4.2절. “주소 세트에 주소 설정 적용” 을 참조하십시오.

4.18. 임시 큐에 특정 주소 설정 적용

예를 들어 JMS를 사용하는 경우 브로커는 UUID(Universally unique identifier)를 주소 이름과 큐 이름으로 할당하여 임시 큐를 생성합니다.

기본 < address-setting match="#"& gt;는 구성된 주소 설정을 임시 큐를 포함하여 모든 큐에 적용합니다. 특정 주소 설정을 임시 큐에만 적용하려면 선택적으로 아래에 설명된 대로 temporary-queue-namespace 를 지정할 수 있습니다. 그런 다음 네임스페이스와 일치하는 주소 설정을 지정하고 브로커는 해당 설정을 모든 임시 큐에 적용할 수 있습니다.

임시 큐가 생성되고 임시 큐 네임스페이스가 존재하는 경우 브로커는 temporary-queue-namespace 값 앞에 있고 구성된 구분 기호(기본값 .)를 주소 이름에 추가합니다. 이를 사용하여 일치하는 주소 설정을 참조합니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

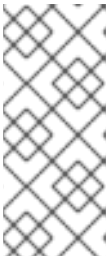
2. temporary-queue-namespace 값을 추가합니다. 예를 들면 다음과 같습니다.

```
<temporary-queue-namespace>temp-example</temporary-queue-namespace>
```

3. 임시 큐 네임스페이스에 해당하는 match 값을 사용하여 address-setting 요소를 추가합니다. 예를 들면 다음과 같습니다.

```
<address-settings>
  <address-setting match="temp-example.#">
    <enable-metrics>false</enable-metrics>
  </address-setting>
</address-settings>
```

이 예제에서는 브로커가 생성한 모든 임시 큐에서 메트릭을 비활성화합니다.



참고

임시 큐 네임스페이스를 지정해도 임시 큐에는 영향을 미치지 않습니다. 예를 들어, 네임스페이스는 임시 큐의 이름을 변경하지 않습니다. 네임스페이스는 임시 큐를 참조하는 데 사용됩니다.

추가 리소스

- 주소 설정에서 와일드카드를 사용하는 방법에 대한 자세한 내용은 4.2절. “주소 세트에 주소 설정 적용” 을 참조하십시오.

4.19. 링 대기열 구성

일반적으로 AMQ Broker의 대기열은 FIFO(first-in, first-out) 의미 체계를 사용합니다. 즉, 브로커는 큐에 메시지를 추가하고 헤드에서 제거합니다. 링 큐는 고정된 지정된 수의 메시지를 보유하는 특수한 유형의 큐입니다. 브로커는 새 메시지가 도착했을 때 큐의 헤드에서 메시지를 제거하여 고정된 큐 크기를 유지하지만 큐에는 지정된 수의 메시지가 이미 있습니다.

예를 들어 크기가 3 으로 구성된 링 큐와 메시지 A,B,C, D 를 순차적으로 전송하는 생산자를 고려하십시오. 메시지 C 가 큐에 도착하면 큐의 메시지 수가 구성된 링 크기에 도달했습니다. 이 시점에서 메시지 A 는 큐의 헤드에 있고 메시지 C는 테두리에 있습니다.At this point, message A is at the head of the queue, while message C is at the tail. D 메시지가 큐에 도착하면 브로커는 메시지를 큐에 추가합니다. 브로커는 고정된 큐 크기를 유지하기 위해 큐 헤드(즉, 메시지 A)에서 메시지를 제거합니다. 이제 메시지 B 가 대기열의 헤드에 있습니다.

4.19.1. 링 대기열 구성

다음 절차에서는 링 큐를 구성하는 방법을 보여줍니다.

프로세스

1. <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

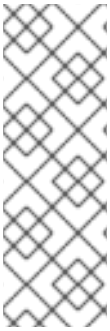
2. 명시적 링 크기 세트가 없는 일치하는 주소에 있는 모든 큐에 대한 기본 링 크기를 정의하려면 address-setting 요소에 default-ring-size 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<address-settings>
  <address-setting match="ring.#">
    <default-ring-size>3</default-ring-size>
  </address-setting>
</address-settings>
```

default-ring-size 매개변수는 자동 생성된 큐의 기본 크기를 정의하는 데 특히 유용합니다. default-ring-size 의 기본값은 -1 입니다(즉, 크기 제한 없음).

3. 특정 큐에 링 크기를 정의하려면 ring-size 키를 큐 요소에 추가합니다. 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<addresses>
  <address name="myRing">
    <anycast>
      <queue name="myRing" ring-size="5" />
    </anycast>
  </address>
</addresses>
```



참고

브로커가 실행되는 동안 ring-size 값을 업데이트할 수 있습니다. 브로커는 업데이트를 동적으로 적용합니다. 새 ring-size 값이 이전 값보다 작으면 브로커는 대기열 헤드에서 메시지를 즉시 삭제하여 새 크기를 적용하지 않습니다. 큐로 전송된 새 메시지는 여전히 이전 메시지의 삭제가 강제 적용되지만, 클라이언트에서 정상적인 메시지 소비를 통해 큐가 새로운, 축소된 크기에 도달하지 않습니다.

4.19.2. 링 대기열 문제 해결

이 섹션에서는 링 대기열의 동작이 구성과 다른 상황에 대해 설명합니다.

In-delivery 메시지 및 롤백

메시지가 소비자에게 전달될 때 메시지는 기술적으로 큐에 존재하지 않지만 아직 승인되지 않은 "내부" 상태에 있습니다. 메시지는 소비자가 승인할 때까지 전달 중 상태로 유지됩니다. 전송 중 상태에 남아

있는 메시지는 링 대기열에서 제거할 수 없습니다.

브로커는 전달 중 메시지를 제거할 수 없으므로 클라이언트는 링 크기 구성이 허용하는 것보다 링 큐로 더 많은 메시지를 보낼 수 있습니다. 예를 들어 다음 시나리오를 고려하십시오.

1. 생산자는 **ring-size="3"** 로 구성된 링 큐에 세 개의 메시지를 보냅니다.

2. 모든 메시지는 즉시 소비자에게 전달됩니다.

   이 시점에서 **messageCount= 3** 을 제공하고**Count= 3** 을 제공합니다.

3. 생산자는 다른 메시지를 큐에 보냅니다. 그런 다음 메시지가 소비자에게 전달됩니다.

   이제 **messageCount = 4** 를 제공하고**Count = 4** 를 제공합니다. 메시지 수 **4** 는 구성된 링 크기 **3** 보다 큽니다. 그러나 브로커는 큐에서 전송 중 메시지를 제거할 수 없기 때문에 이 상황을 허용할 의무가 있습니다.

4. 이제 메시지를 승인하지 않고 소비자가 닫혀 있다고 가정합니다.

   이 경우 승인되지 않은 네 개의 메시지가 브로커에 다시 취소되고 사용된 역순으로 큐의 헤드에 추가됩니다. 이 작업을 수행하면 큐가 구성된 링 크기에 도달합니다. 링 대기열은 헤드의 메시지에 따라 대기열의 메시지를 선호하기 때문에 대기열은 대기열의 헤드에 다시 추가된 마지막 메시지였기 때문에 생산자가 보낸 첫 번째 메시지를 삭제합니다. 트랜잭션 또는 코어 세션 롤백은 동일한 방식으로 처리됩니다.

코어 클라이언트를 직접 사용하거나 **AMQ Core Protocol JMS** 클라이언트를 사용하는 경우 **consumerWindowSize** 매개변수 값(**1024 * 1024**바이트)의 값을 줄임으로써 전달 시 메시지 수를 최소화할 수 있습니다.

예약된 메시지

예약된 메시지가 큐로 전송되면 메시지는 일반 메시지와 같이 큐에 즉시 추가되지 않습니다. 대신 브로커는 중간 버퍼에 예약된 메시지를 보유하고 메시지의 세부 정보에 따라 큐 헤드에 전달할 메시지를 예약합니다. 그러나 예약된 메시지는 큐의 메시지 수에 계속 반영됩니다. 전송 중 메시지와 마찬가지로 이 동작으로 인해 브로커가 링 큐 크기를 적용하지 않는 것처럼 보일 수 있습니다. 예를 들어 다음 시나리오를 고려하십시오.

1. **12:00**에 생산자는 **ring-size="3"** 로 구성된 링 큐로 메시지를 전송합니다. 이 메시지는

12:05로 예정되어 있습니다.

이 시점에서 messageCount= 1 및 scheduledCount= 1.

2. *12:01에서 생산자는 메시지 B 를 동일한 링 큐로 보냅니다.*

   이제 messageCount= 2 및 scheduledCount= 1 입니다.

3. *12:02에서 생산자는 메시지 C 를 동일한 링 큐로 보냅니다.*

   이제 messageCount= 3 및 scheduledCount= 1.

4. *12:03에서 생산자는 메시지 D 를 동일한 링 큐로 보냅니다.*

   이제 messageCount= 4 및 scheduledCount= 1 입니다.

큐의 메시지 수는 이제 구성된 링 크기인 3 보다 큰 4 개입니다. 그러나 예약된 메시지는 아직 큐에 기술적으로 표시되지 않습니다(즉, 브로커에 있고 큐에 배치되도록 예약됨). 예약된 전송 시간 12:05에서 브로커는 큐의 헤드에 메시지를 넣습니다. 그러나 링 큐가 이미 구성된 크기에 도달했으므로 예약된 메시지 A 가 즉시 제거됩니다.

페이지가 지정된 메시지

전달된 예약된 메시지 및 메시지와 유사하게 페이지 지정된 메시지는 브로커가 적용하는 링 큐 크기로 계산되지 않습니다. 메시지는 큐 수준이 아닌 주소 수준에서 실제로 페이지링되기 때문입니다. 페이지 지정된 메시지는 큐의 messageCount 값에 반영되지만 큐에 기술적으로 반영되지 않습니다.

링 큐가 있는 주소에 페이징을 사용하지 않는 것이 좋습니다. 대신 전체 주소가 메모리에 적합할 수 있는지 확인합니다. 또는 address-full-policy 매개변수를 DROP,BLOCK 또는 FAIL 값으로 구성합니다.

추가 리소스

- *브로커는 소급 주소를 구성할 때 링 큐의 내부 인스턴스를 생성합니다. 자세한 내용은 4.20절. "소급 주소 구성" 에서 참조하십시오.*

4.20. 소급 주소 구성

주소를 소급 으로 구성하면 아직 주소에 바인딩된 큐가 없는 경우를 포함하여 해당 주소로 전송된 메시지를 유지할 수 있습니다. 나중에 큐가 생성되고 주소에 바인딩되면 브로커는 해당 큐에 메시지를 소급 배포합니다. 주소가 **retroactive**으로 구성되지 않고 아직 큐가 바인딩되지 않은 경우 브로커는 해당 주소로 전송된 메시지를 삭제합니다.

소급 주소를 구성할 때 브로커는 링 큐라는 대기열 유형의 내부 인스턴스를 생성합니다. 링 큐는 고정된 지정된 수의 메시지를 보유하는 특수한 유형의 큐입니다. 큐가 지정된 크기에 도달하면 큐에 도달하는 다음 메시지는 큐에서 가장 오래된 메시지를 강제 실행합니다. 소급 주소를 구성할 때 내부 링 대기열의 크기를 간접적으로 지정합니다. 기본적으로 내부 큐는 멀티캐스트 라우팅 유형을 사용합니다.

소급 주소에서 사용하는 내부 링 큐는 관리 **API**를 통해 노출됩니다. 큐 비우기와 같은 메트릭을 검사하고 일반적인 관리 작업을 수행할 수 있습니다. 링 큐는 또한 주소의 전체 메모리 사용량에 기여하며, 이는 메시지 페이징과 같은 동작에 영향을 미칩니다.

다음 절차에서는 주소를 소급으로 구성하는 방법을 보여줍니다.

프로세스

1. **<broker_instance_dir>/etc/broker.xml** 구성 파일을 엽니다.

2. **address-setting** 요소에서 **retroactive-message-count** 매개변수 값을 지정합니다. 지정한 값은 브로커가 보존할 메시지 수를 정의합니다. 예를 들면 다음과 같습니다.

```
<configuration>
 <core>
  ...
  <address-settings>
    <address-setting match="orders">
      <retroactive-message-count>100</retroactive-message-count>
    </address-setting>
  </address-settings>
...
 </core>
</configuration>
```

참고

broker.xml 구성 파일 또는 관리 API에서 브로커가 실행되는 동안 retroactive-message-count 값을 업데이트할 수 있습니다. 그러나 이 매개변수의 값을 줄이는 경우 링 큐를 통해 소급 주소가 구현되기 때문에 추가 단계가 필요합니다. ring-size 매개변수가 감소된 링 큐는 새 ring-size 값을 얻기 위해 큐에서 메시지를 자동으로 삭제하지 않습니다. 이 동작은 의도하지 않은 메시지 손실에 대한 보호 기능입니다. 이 경우 관리 API를 사용하여 링 큐의 메시지 수를 수동으로 줄여야 합니다.

추가 리소스

- 링 큐에 대한 자세한 내용은 4.19절. “링 대기열 구성” 을 참조하십시오.

4.21. 내부 관리 주소 및 큐에 대한 권고 메시지 비활성화

기본적으로 AMQ Broker는 OpenWire 클라이언트가 브로커에 연결된 경우 주소 및 큐에 대한 권고 메시지를 생성합니다. 권고 메시지는 브로커가 생성한 내부 관리 주소로 전송됩니다. 이러한 주소는 사용자가 배포한 주소 및 큐와 동일한 디스플레이 내에서 AMQ Management Console에 표시됩니다. 유용한 정보를 제공하지만 브로커가 많은 수의 대상을 관리하면 권고 메시지가 바람직하지 않은 결과를 초래할 수 있습니다. 예를 들어, 메시지는 메모리 사용량을 늘리거나 연결 리소스를 소모할 수 있습니다. 또한 권고 메시지를 보내기 위해 생성된 모든 주소를 표시하려고 할 때 AMQ 관리 콘솔이 혼동될 수 있습니다. 이러한 상황을 방지하려면 다음 매개변수를 사용하여 브로커에서 권고 메시지의 동작을 구성할 수 있습니다.

supportAdvisory

권고 메시지 또는 false 를 생성하여 비활성화하려면 이 옵션을 true 로 설정합니다. 기본값은 true입니다.

suppressInternalManagementObjects

권고 메시지를 true 로 설정하여 registry 및 AMQ Management Console과 같은 관리 서비스에 권고 메시지를 노출하지 않도록 합니다. 기본값은 true입니다.

다음 절차에서는 브로커에서 권고 메시지를 비활성화하는 방법을 보여줍니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. OpenWire 커넥터의 경우 구성된 URL에 supportAdvisory 및 suppressInternalManagementObjects 매개변수를 추가합니다. 이 섹션의 앞부분에서 설명한

대로 값을 설정합니다. 예를 들면 다음과 같습니다.

```
<acceptor name="artemis">tcp://127.0.0.1:61616?
protocols=CORE,AMQP,OPENWIRE;supportAdvisory=false;suppressInternalManagem
entObjects=false</acceptor>
```

4.22. 주소 및 큐 처리

페더레이션을 사용하면 브로커가 공통 클러스터에 있을 필요 없이 브로커 간 메시지를 전송할 수 있습니다. 브로커는 독립 실행형 또는 별도의 클러스터에 있을 수 있습니다. 또한 소스 및 대상 브로커는 서로 다른 관리 도메인에 있을 수 있습니다. 즉, 브로커의 구성, 사용자 및 보안 설정이 다를 수 있습니다. 브로커가 다른 버전의 **AMQ Broker**를 사용하고 있을 수도 있습니다.

예를 들어, 페더레이션은 한 클러스터에서 다른 클러스터로 안정적으로 메시지를 보내는 데 적합합니다. 이러한 전송은 클라우드 인프라의 영역 또는 인터넷을 통해 전송될 수 있습니다. 소스 브로커에서 대상 브로커로의 연결이 끊어지면(예: 네트워크 장애로 인해) 소스 브로커는 대상 브로커가 다시 온라인 상태가 될 때까지 연결을 다시 설정하려고 합니다. 대상 브로커가 다시 온라인 상태가 되면 메시지 전송이 재개됩니다.

관리자는 주소 및 큐 정책을 사용하여 페더레이션을 관리할 수 있습니다. 정책 구성은 특정 주소 또는 큐와 일치하거나, 구성과 일치하는 와일드카드 표현식을 주소 또는 큐 집합과 일치시킬 수 있습니다. 따라서 일치하는 세트에서 큐 또는 주소가 추가되거나 제거될 때 페더레이션을 동적으로 적용할 수 있습니다. 정책에는 특정 주소 및 큐를 포함 및/또는 제외하는 여러 표현식이 포함될 수 있습니다. 또한 브로커 또는 브로커 클러스터에 여러 정책을 적용할 수 있습니다.

AMQ Broker에서 두 가지 주요 페더레이션 옵션은 주소 페더레이션 및 큐 페더레이션 입니다. 이러한 옵션은 다음 섹션에 설명되어 있습니다.

참고

브로커는 페더레이션 및 로컬 전용 구성 요소에 대한 구성을 포함할 수 있습니다. 즉, 브로커에 페더레이션을 구성하면 브로커에 모든 것을 통합 할 필요가 없습니다.

4.22.1. 주소 페더레이션 정보

주소 페더레이션은 연결된 브로커 간의 전체 멀티 캐스트 배포 패턴과 같습니다. 예를 들어 **BrokerA**의 주소로 전송되는 모든 메시지는 해당 브로커의 모든 큐로 전달됩니다. 또한 각 메시지는 **BrokerB** 로 전달되고 연결된 모든 대기열도 제공됩니다.

주소 페더레이션은 브로커를 원격 브로커의 주소에 동적으로 연결합니다. 예를 들어 로컬 브로커가 원

격 브로커의 주소에서 메시지를 가져오려는 경우 원격 주소에 큐가 자동으로 생성됩니다. 그런 다음 원격 브로커의 메시지가 이 큐에 사용됩니다. 마지막으로 메시지는 원래 로컬 주소에 직접 게시된 것처럼 로컬 브로커의 해당 주소로 복사됩니다.

원격 브로커는 페더레이션이 주소를 만들 수 있도록 재구성할 필요가 없습니다. 그러나 로컬 브로커에는 원격 주소에 대한 권한이 부여되어야 합니다.

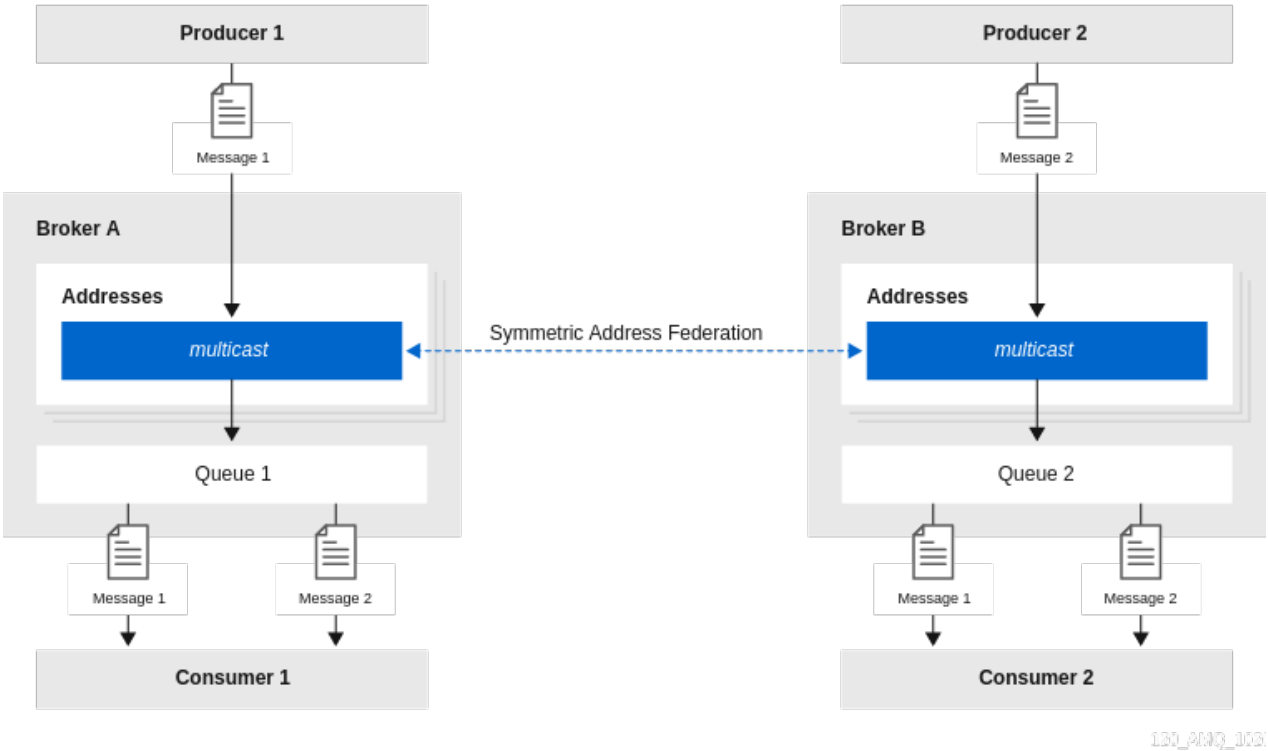
4.22.2. 주소 페더레이션을 위한 공통 토폴로지

주소 페더레이션 사용을 위한 몇 가지 일반적인 토폴로지는 아래에 설명되어 있습니다.

대칭 토폴로지

대칭 토폴로지에서 생산자 및 소비자는 각 브로커에 연결됩니다. 큐와 소비자는 생산자가 게시한 메시지를 수신할 수 있습니다. 대칭 토폴로지의 예는 다음과 같습니다.

그림 4.5. 대칭 토폴로지의 주소 페더레이션

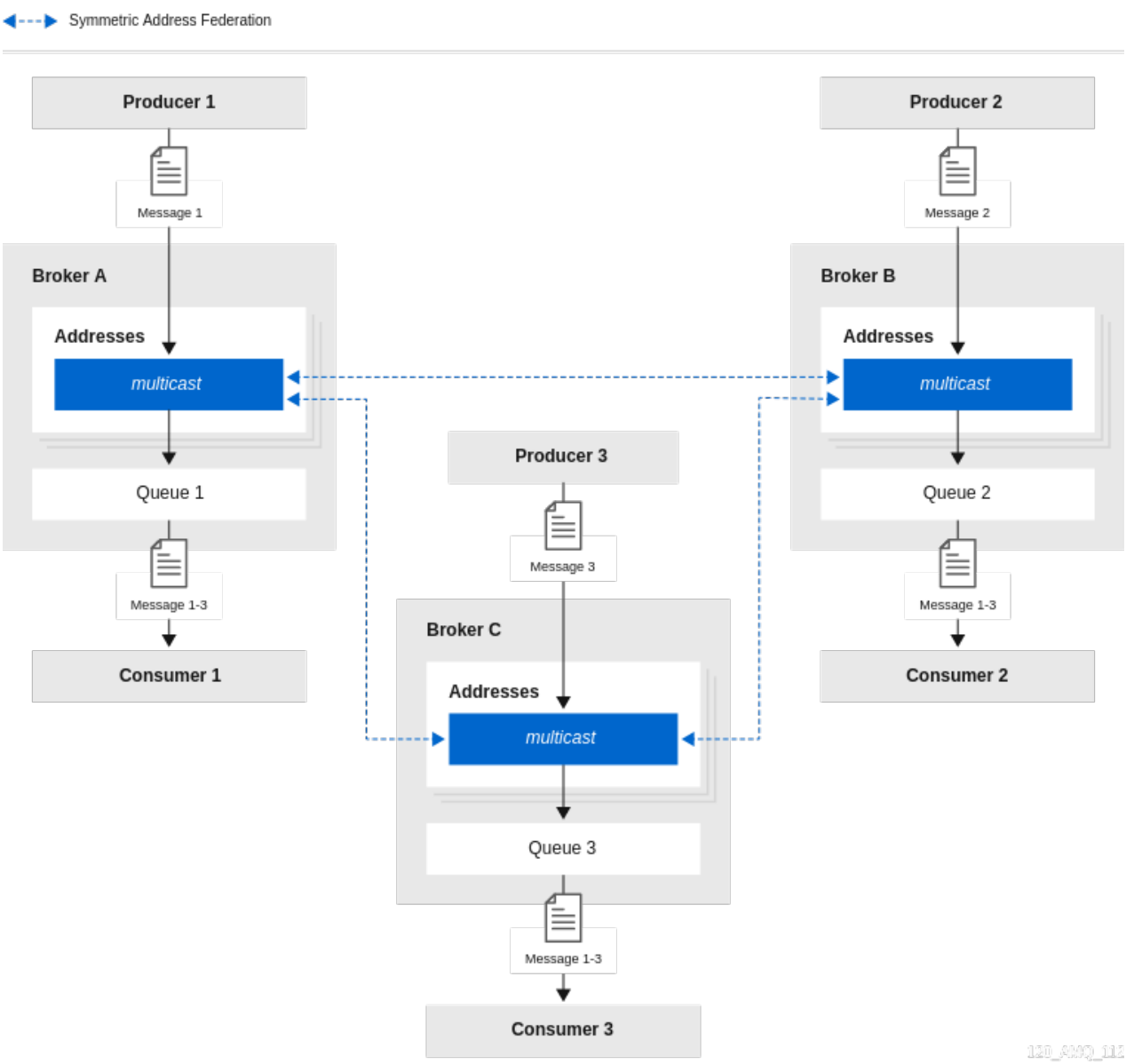


*대칭 토폴로지에 대한 주소 페더레이션을 구성할 때 주소 정책의 **max-hops** 속성 값을 **1** 로 설정하는 것이 중요합니다. 이렇게 하면 메시지가 한 번만 복사되므로 주기적인 복제를 방지할 수 있습니다. 이 속성이 더 큰 값으로 설정된 경우 소비자는 동일한 메시지의 여러 복사본을 수신합니다.*

전체 메시 토폴로지

전체 메시 토폴로지는 대칭 설정과 유사합니다. 세 개 이상의 브로커가 서로 대칭화되어 전체 메시를 생성합니다. 이 설정에서 생산자 및 소비자는 각 브로커에 연결됩니다. 큐와 소비자는 모든 생산자가 게시한 메시지를 수신할 수 있습니다. 이 토폴로지의 예는 다음과 같습니다.

*그림 **4.6.** 전체 메시 토폴로지의 주소 페더레이션*

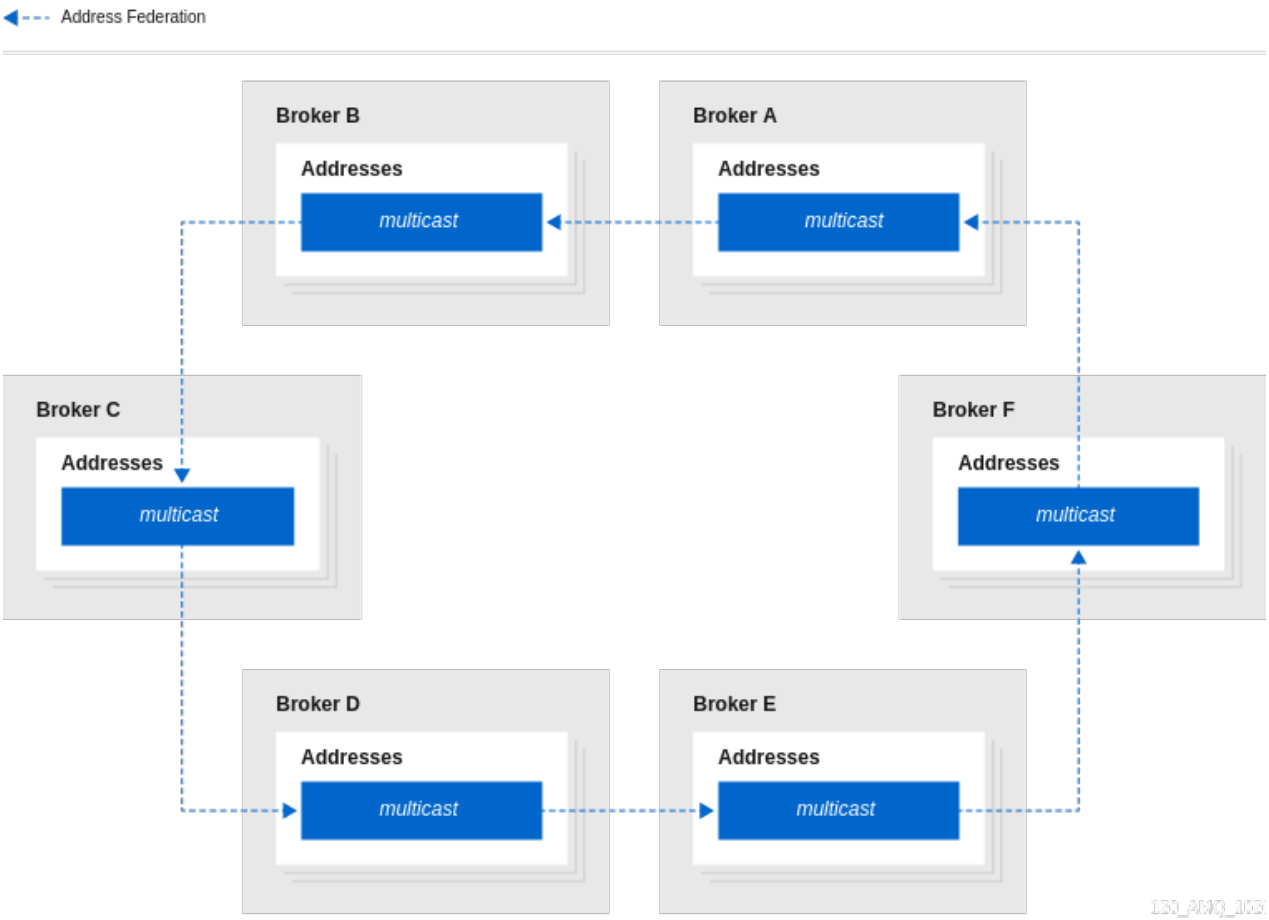


*대칭 설정과 마찬가지로 전체 메시 토폴로지에 대한 주소 페더레이션을 구성할 때 주소 정책의 **max-hops** 속성 값을 **1** 로 설정하는 것이 중요합니다. 이렇게 하면 메시지가 한 번만 복사되므로 주기적인 복제를 방지할 수 있습니다.*

링 토폴로지

브로커의 링에서 각 페더레이션 주소는 링에서 서로만 업스트림됩니다. 이 토폴로지의 예는 다음과 같습니다.

그림 4.7. 링 토폴로지의 주소 페더레이션



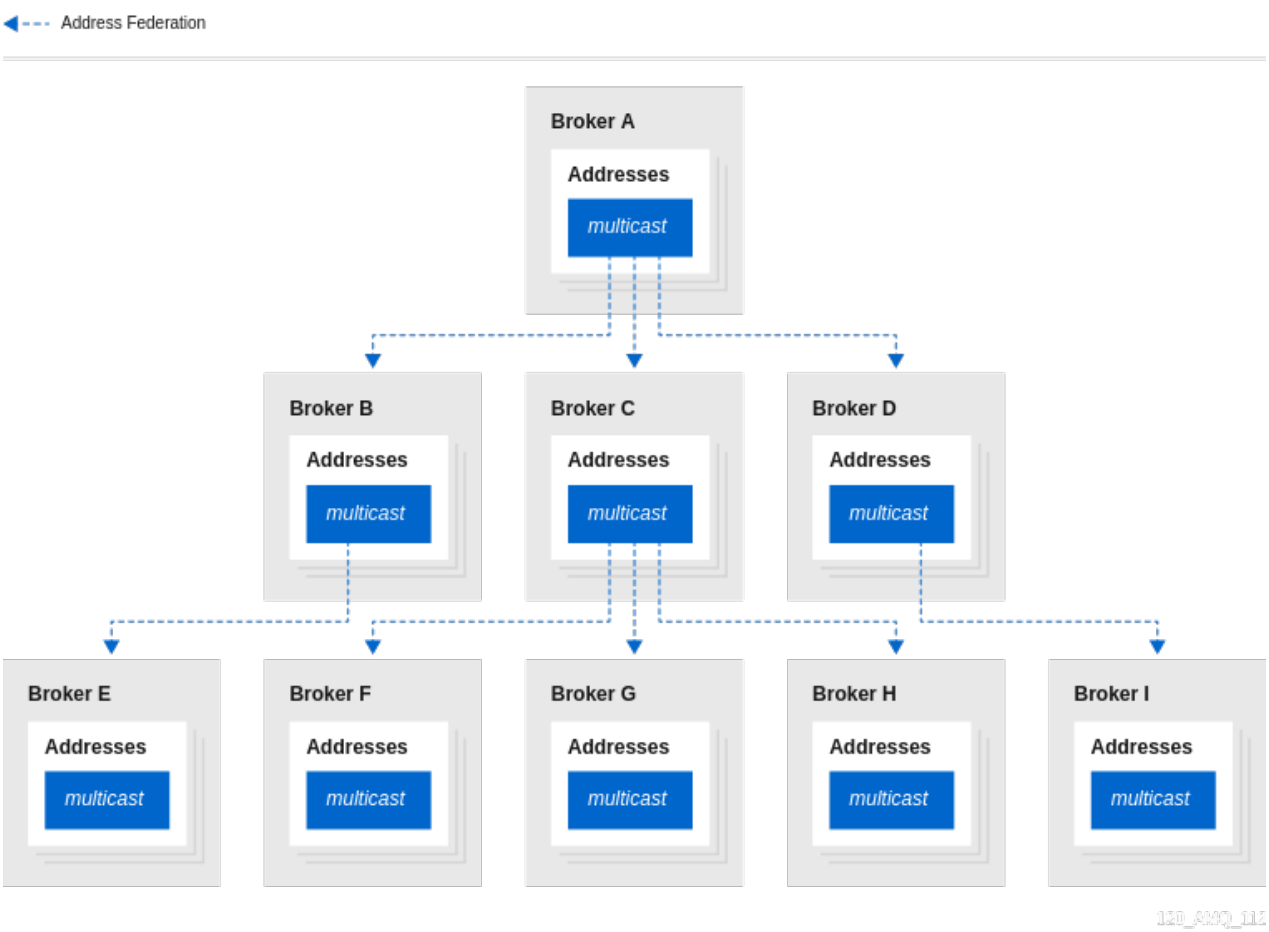
*순환 복제를 피하기 위해 링 토폴로지에 대한 페더레이션을 구성할 때 주소 정책의 **max-hops** 속성을 **n-1** 값으로 설정하는 것이 중요합니다. 여기서 **n** 은 링의 노드 수입니다. 예를 들어 위에 표시된 링 토폴로지에서 **max-hops** 값은 **5** 로 설정됩니다. 이렇게 하면 링의 모든 주소가 정확히 한 번 메시지를 볼 수 있습니다.*

링 토폴로지의 장점은 설정해야 하는 물리적 연결 수 측면에서 설정하는 것이 저렴하다는 점입니다. 그러나 이러한 유형의 토폴로지의 단점은 단일 브로커가 실패하면 전체 링이 실패한다는 것입니다.

팬 아웃 토폴로지

팬 아웃 토폴로지에서 단일 마스터 주소는 페더레이션 주소 트리에 의해 연결됩니다. 마스터 주소에 게시된 모든 메시지는 트리의 브로커에 연결된 모든 소비자가 수신할 수 있습니다. 트리는 임의의 깊이로 구성할 수 있습니다. 트리에서 기존 브로커를 재구성할 필요 없이 트리를 확장할 수도 있습니다. 이 토폴로지의 예는 다음과 같습니다.

그림 4.8. 팬 아웃 토폴로지의 주소 페더레이션



*팬 아웃 토폴로지에 대한 페더레이션을 구성할 때 주소 정책의 **max-hops** 속성을 **n-1** 값으로 설정해야 합니다. 여기서 **n** 은 트리의 수준 수입니다. 예를 들어 위에 표시된 팬 아웃 토폴로지에서 **max-hops** 값은 **2** 로 설정됩니다. 이렇게 하면 트리의 모든 주소가 정확히 한 번 메시지를 볼 수 있습니다.*

4.22.3. 주소 페더레이션 구성에서 다양한 바인딩 지원

주소 페더레이션을 구성할 때 주소 정책 구성에서 다양한 바인딩에 대한 지원을 추가할 수 있습니다. 이러한 지원을 추가하면 페더레이션이 다양한 바인딩에 응답하여 원격 브로커에서 지정된 주소에 대한 페더레이션 소비자를 만들 수 있습니다.

*예를 들어 **test.federation.source** 라는 주소가 주소 정책에 포함되어 있고 **test.federation.target** 이라는 다른 주소가 포함되어 있지 않다고 가정합니다. 일반적으로, **test.federation.target** 에서 큐가 생성되면 주소가 주소 정책의 일부가 아니기 때문에 페더레이션 소비자가 생성되지 않습니다. 그러나 **test.federation.source** 가 소스 주소이고 **test.federation.target** 이 전달 주소가 되도록 다양한 바인딩을 생성하는 경우 전달 주소에 지속성 소비자가 생성됩니다. 소스 주소는 여전히 멀티 캐스트 라우팅 유형을 사용해야 하지만 대상 주소는 멀티 캐스트 또는 **anycast** 를 사용할 수 있습니다.*

*예제 사용 사례는 **JMS** 주제(멀티 캐스트 주소)를 **JMS** 큐(**anycast** 주소)로 리디렉션하는 다양한 방법*

입니다. 이를 통해 레거시 소비자가 JMS 2.0 및 공유 서브스크립션을 지원하지 않는 항목에 대한 메시지의 부하 분산이 가능합니다.

4.22.4. 페더레이션 구성

Core 프로토콜 또는 7.12, AMQP를 사용하여 주소 및 큐 페더레이션을 구성할 수 있습니다. AMQP를 사용하여 페더레이션을 사용하면 다음과 같은 이점이 있습니다.

- 클라이언트가 AMQP를 메시징에 사용하는 경우 AMQP를 사용하여 AMQP와 Core 프로토콜 간에 메시지를 변환할 필요가 없으며 그 반대의 경우도 마찬가지입니다. 이는 페더레이션 프로토콜을 사용하는 경우 필요합니다.

- AMQP 페더레이션은 단일 발신 연결을 통해 양방향 페더레이션을 지원합니다. 이를 통해 원격 브로커가 로컬 브로커에 다시 연결할 필요가 없습니다.이 브로커는 페더레이션의 Core 프로토콜을 사용하고 네트워크 정책에 의해 방지 될 수있는 요구 사항입니다.

4.22.4.1. AMQP를 사용하여 페더레이션 구성

다음 정책을 사용하여 AMQP를 사용하여 주소 및 큐 페더레이션을 구성할 수 있습니다.

- 로컬 주소 정책은 로컬 브로커가 주소에서 수요를 감시하도록 구성하고, 해당 요청이 있는 경우 로컬 브로커에 메시지를 유도하기 위해 원격 브로커의 일치하는 주소에 페더레이션 소비자를 만듭니다.

- 원격 주소 정책은 원격 브로커가 주소에서 수요를 감시하도록 구성하고, 해당 요청이 있는 경우 로컬 브로커의 일치하는 주소에 페더레이션 소비자를 생성하여 원격 브로커에 메시지를 유도합니다.

- 로컬 큐 정책은 로컬 브로커가 대기열에 대한 수요를 감시하도록 구성하고, 해당 요청이 있는 경우 원격 브로커의 일치하는 큐에 페더레이션 소비자를 생성하여 로컬 브로커에 메시지를 전송합니다.

- 원격 대기열 정책은 원격 브로커가 대기열에서 수요를 감시하도록 구성하고, 해당 요청이 있는 경우 로컬 브로커의 일치하는 큐에 페더레이션 소비자를 생성하여 원격 브로커에 메시지를 전송합니다.

원격 주소 및 큐 정책은 원격 브로커로 전송되고 역방향 페더레이션 연결을 제공하기 위해 원격 브로커의 로컬 정책이 됩니다. 역방향 페더레이션 연결에 대한 정책을 적용할 때 정책을 수신한 브로커는 로컬

*브로커이고 정책을 보낸 브로커는 원격 브로커입니다. 로컬 브로커에 원격 주소 및 큐 정책을 구성하면 모든 페더레이션 구성을 단일 브로커에 유지할 수 있습니다. 예를 들어 **hub-and-spoke** 토폴로지에 유용한 접근 방식이 될 수 있습니다.*

4.22.4.1.1. AMQP를 사용하여 주소 페더레이션 구성

***< broker-connections>** 요소를 사용하여 **AMQP**를 사용하여 주소 페더레이션을 구성합니다.*

사전 요구 사항

***< amqp-connection >** 요소에 지정된 사용자에게는 원격 브로커의 주소와 큐와 일치하는 읽기 및 쓰기 권한이 있습니다.*

프로세스

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. ***< amqp-connection>** 요소를 포함하는 **<broker connections >** 요소를 추가합니다. **&lt ;amqp-connection >** 요소에서 원격 브로커의 연결 세부 정보를 지정하고 페더레이션 구성에 이름을 할당합니다. 예를 들면 다음과 같습니다.*

```
<broker-connections>
  <amqp-connection uri="tcp://<__HOST__>:<__PORT__>" user="federation_user"
password="federation_pwd" name="queue-federation-example">
  </amqp-connection>
</broker-connections>
```

3. ***&lt ;federation>** 요소를 추가하고 다음 중 하나 또는 둘 다를 포함합니다.*

   - *원격 브로커에서 로컬 브로커로 메시지를 통합하는 **<local-address-policy >** 요소입니다.*
   - *로컬 브로커에서 원격 브로커로 메시지를 통합하는 **<remote-address-policy >** 요소입니다.*

다음 예제에서는 로컬 및 원격 주소 정책이 모두 있는 페더레이션 요소를 보여줍니다.

```
<broker-connections>
  <amqp-connection uri="tcp://<__HOST__>:<__PORT__>" user="federation_user"
```

```
password="federation_pwd" name="queue-federation-example">
   <federation>
    <local-address-policy name="example-local-address-policy" auto-delete="true" auto-
delete-delay="1" auto-delete-message-count="2" max-hops="1" enable-divert-
bindings="true">
      <include address-match="queue.news.#" />
    <include address-match="queue.bbc.news" />
      <exclude address-match="queue.news.sport.#" />
     </local-address-policy>
     <remote-address-policy name="example-remote-address-policy">
      <include address-match="queue.usatoday" />
     </remote-address-policy>
   </federation>
  </amqp-connection>
</broker-connections>
```

동일한 매개변수는 로컬 및 원격 주소 정책 모두에서 구성할 수 있습니다. 유효한 매개변수는 다음과 같습니다.

name

*주소 정책의 이름입니다. 모든 주소 정책 이름은 **< broker-connections >** 요소의 **< federation >** 요소 내에서 고유해야 합니다.*

max-hops

페더레이션 중에 메시지가 만들 수 있는 최대 홉 수입니다. 기본값은 대부분의 간단한 페더레이션 배포에 적합합니다. 그러나 특정 토폴로지에서는 메시지가 루프되지 않도록 하는 데 더 큰 값이 필요할 수 있습니다.

자동 삭제

*주소 페더레이션의 경우 메시지가 페인딩되는 브로커에 생성됩니다. 시작 브로커의 연결이 끊어지고 지연 및 메시지 수 매개변수가 충족되면 자동 삭제 대기열을 표시하려면 이 매개변수를 **true**로 설정합니다. 동적으로 생성된 큐의 정리를 자동화하려는 경우 이 옵션이 유용합니다. 기본값은 **false** 이므로 큐가 자동으로 삭제되지 않습니다.*

auto-delete-delay

*브로커 시작 연결이 끊어진 후 생성된 큐를 자동으로 삭제할 수 있는 시간(밀리초)입니다. 기본값은 **0**입니다.*

auto-delete-message-count

*큐의 메시지 수가 큐를 자동으로 삭제하기 전에 큐보다 작거나 같아야 합니다. 기본값은 **0**입니다.*

enable-divert-bindings

***true** 로 설정하면 필요에 따라 다양한 바인딩을 청취할 수 있습니다. 주소가 있는 다양*

*한 바인딩이 주소 정책에 대한 포함된 주소와 일치하는 경우 다이버트의 전달 주소와 일치하는 모든 큐 바인딩이 수요를 생성합니다. 기본값은 **false**입니다.*

포함

정책에 포함할 주소와 일치하는 주소와 일치하는 패턴입니다. 여러 패턴을 지정할 수 있습니다. 패턴을 지정하지 않으면 모든 주소가 정책에 포함됩니다.

*정확한 주소를 지정할 수 있습니다(예: **queue.bbc.news**). 또는 숫자 기호(**#**) 와일드카드 문자를 사용하여 일치하는 주소 집합을 지정할 수 있습니다. 이전 예에서 로컬 주소 정책에는 **queue.news** 문자열로 시작하는 모든 주소도 포함됩니다.*

exclude

정책에서 제외할 주소와 일치하는 주소와 일치하는 패턴입니다. 여러 패턴을 지정할 수 있습니다. 패턴을 지정하지 않으면 주소가 정책에서 제외되지 않습니다.

*정확한 주소를 지정할 수 있습니다(예: **queue.bbc.news**). 또는 숫자 기호(**#**) 와일드카드 문자를 사용하여 일치하는 주소 집합을 지정할 수 있습니다. 이전 예에서 로컬 주소 정책은 **queue.news.sport** 문자열로 시작하는 모든 주소를 제외합니다.*

4.22.4.1.2. AMQP를 사용하여 대기열 페더레이션 구성

프로세스

***< broker connections>** 요소를 사용하여 **AMQP**의 대기열 페더레이션을 구성합니다.*

사전 요구 사항

***< amqp-connection >** 요소에 지정된 사용자에게는 원격 브로커의 주소와 큐와 일치하는 읽기 및 쓰기 권한이 있습니다.*

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. ***< amqp-connection>** 요소를 포함하는 **<broker connections >** 요소를 추가합니다. **&lt ;amqp-connection >** 요소에서 원격 브로커의 연결 세부 정보를 지정하고 페더레이션 구성에 이름을 할당합니다. 예를 들면 다음과 같습니다.*

```
<broker-connections>
  <amqp-connection uri="tcp://<__HOST__>:<__PORT__>" user="federation_user"
password="federation_pwd" name="queue-federation-example">
```

```
  </amqp-connection>
</broker-connections>
```

3. **&lt ;federation>** 요소를 추가하고 다음 중 하나 또는 둘 다를 포함합니다.

- 원격 브로커에서 로컬 브로커로 메시지를 통합하는 **<local-queue-policy >** 요소입니다.

- 로컬 브로커에서 원격 브로커로 메시지를 통합하는 **<remote-queue-policy >** 요소입니다.

다음 예제에서는 로컬 큐 정책을 포함하는 페더레이션 요소를 보여줍니다.

```
<broker-connections>
 <amqp-connection uri="tcp://HOST:PORT" name="federation-example">
  <federation>
   <local-queue-policy name="example-local-queue-policy">
    <include address-match="#" queue-match="#.remote" />
    <exclude address-match="#" queue-match="#.local" />
   </local-queue-policy>
  </federation>
 </amqp-connection>
</broker-connections>
```

name

큐 정책의 이름입니다. 모든 큐 정책 이름은 **< broker-connections >** 요소의 **< federation >** 요소 내에서 고유해야 합니다.

포함

정책에 포함되기 위해 해당 주소의 특정 큐와 일치하는 주소 일치 패턴 및 큐 일치 패턴입니다. **address-match** 매개변수와 마찬가지로 **queue-match** 매개변수에 대한 정확한 이름을 지정하거나 와일드카드 표현식을 사용하여 대기열 세트를 지정할 수 있습니다. 위 예제에서는 주소 일치 값 **#** 임을 나타내는 모든 주소의 **.remote** 문자열과 일치하는 큐가 포함됩니다.

exclude

정책을 제외하기 위해 해당 주소의 특정 큐와 일치하는 주소 일치 패턴 및 큐 일치 패턴입니다. **address-match** 매개변수와 마찬가지로 **queue-match** 매개변수에 대한 정확한 이름을 지정하거나 와일드카드 표현식을 사용하여 대기열 세트를 지정할 수 있습니다. 이전 예에서 주소 일치 값 **#** 로 표시되는 모든 주소의 **.local** 문자열과 일치하는 큐는 제외됩니다.

priority-adjustment

통합 소비자의 값을 조정하여 동일한 큐에 있는 다른 로컬 소비자보다 우선 순위가 낮은지 확인합니다. 기본값은 **-1** 로, 로컬 소비자가 페더레이션 소비자보다 우선 순위를 지정하도록 합니다.

include-federated

이 매개 변수의 값이 **false** 로 설정되면 구성이 이미 제공된 소비자(즉, 페더레이션 대기열의 소비자)를 다시 제공하지 않습니다. 이렇게 하면 대칭 또는 폐쇄 루프 토폴로지에서 제공되지 않는 소비자와 메시지가 시스템 전체에서 끝없이 흐름되는 상황을 방지할 수 있습니다.

closed-loop 토폴로지가 없는 경우 이 매개변수의 값을 **true** 로 설정할 수 있습니다. 예를 들어 **BrokerA, BrokerB** 및 **BrokerC**의 세 가지 브로커 체인과 **BrokerA**의 생산자와 **BrokerC**의 소비자가 있다고 가정합니다. 이 경우 **BrokerB**가 소비자를 **BrokerA**에 다시 공급하도록 합니다.

4.22.4.2. Core 프로토콜을 사용하여 페더레이션 구성

Core 프로토콜을 사용하도록 메시지 및 큐 페더레이션을 구성할 수 있습니다.

4.22.4.2.1. 브로커 클러스터에 대한 페더레이션 구성

다음 섹션의 예제에서는 독립 실행형 로컬 브로커와 원격 브로커 간에 주소 및 큐 페더레이션을 구성하는 방법을 보여줍니다. 독립 실행형 브로커 간 페더레이션의 경우 페더레이션 구성의 이름과 모든 주소 및 큐 정책의 이름은 로컬 브로커와 원격 브로커 간에 고유해야 합니다.

그러나 클러스터에서 브로커에 대한 페더레이션을 구성하는 경우 추가 요구 사항이 있습니다. 클러스터형 브로커의 경우 페더레이션 구성의 이름과 해당 구성 내의 모든 주소 및 큐 정책 이름은 해당 클러스터 의 모든 브로커에 대해 동일해야 합니다.

동일한 클러스터의 브로커가 동일한 페더레이션 구성 및 주소 및 큐 정책 이름을 사용하도록 하면 메시지 중복이 발생하지 않습니다. 예를 들어, 동일한 클러스터 내의 브로커가 서로 다른 페더레이션 구성 이름을 갖는 경우, 이로 인해 동일한 주소에 대해 여러 다른 이름 전달 대기열이 생성되어 다운스트림 소비자에 대한 메시지 복제가 발생할 수 있습니다. 반대로, 동일한 클러스터의 브로커가 동일한 페더레이션 구성 이름을 사용하는 경우 기본적으로 다운스트림 소비자에 부하 분산되는 복제, 클러스터형 전달 대기열을 생성합니다. 이렇게 하면 메시지 중복을 방지할 수 있습니다.

4.22.4.2.2. 업스트림 주소 페더레이션 구성

다음 예제에서는 독립 실행형 브로커 간에 업스트림 주소 페더레이션을 구성하는 방법을 보여줍니다. 이 예에서는 로컬(즉, 다운스트림) 브로커에서 일부 원격(즉, 업스트림) 브로커로 페더레이션을 구성합니다.

사전 요구 사항

- *다음 예제에서는 독립 실행형 브로커 간에 주소 페더레이션을 구성하는 방법을 보여줍니다. 그러나 브로커 클러스터에 대한 페더레이션 구성을 위한 요구 사항도 숙지해야 합니다. 자세한 내용은 4.22.4.2.1절. “브로커 클러스터에 대한 페더레이션 구성”의 내용을 참조하십시오.*

프로세스

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. ***< federation>** 요소를 포함하는 새로운 **< federations >** 요소를 추가합니다. 예를 들면 다음과 같습니다.*

```
<federations>
  <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
  </federation>
</federations>
```

name

페더레이션 구성의 이름입니다. 이 예에서 이름은 로컬 브로커의 이름에 해당합니다.

user

업스트림 브로커에 연결하기 위한 공유 사용자 이름입니다.

암호

업스트림 브로커에 연결하기 위한 공유 암호입니다.

참고

원격 브로커의 경우 사용자 및 암호 인증 정보가 다른 경우 구성에 추가할 때 해당 브로커의 인증 정보를 별도로 지정할 수 있습니다. 이 내용은 이 절차의 뒷부분에서 설명합니다.

3. *페더레이션 요소 내에서 **< address-policy>** 요소를 추가합니다. 예를 들면 다음과 같습니다.*

```
<federations>
  <federation name="eu-north-1" user="federation_username"
```

```
password="32a10275cf4ab4e9">

      <address-policy name="news-address-federation" auto-delete="true" auto-delete-
delay="300000" auto-delete-message-count="-1" enable-divert-bindings="false" max-
hops="1" transformer-ref="news-transformer">
      </address-policy>

   </federation>
</federations>
```

name

주소 정책의 이름입니다. 브로커에 구성된 모든 주소 정책에는 고유한 이름이 있어야 합니다.

자동 삭제

주소 페더레이션 중에 로컬 브로커는 원격 주소에서 자신 큐를 동적으로 생성합니다. **auto-delete** 속성 값은 로컬 브로커의 연결이 끊어지고 **auto-delete-delay** 및 **auto-delete-message-count** 속성의 값도 도달하면 원격 큐를 삭제해야 하는지 여부를 지정합니다. 동적으로 생성된 큐의 정리를 자동화하려는 경우 이 옵션이 유용합니다. 로컬 브로커가 오랜 시간 동안 연결이 끊어진 경우 원격 브로커에서 메시지를 빌드하지 않으려면 유용한 옵션도 사용할 수 있습니다. 그러나 연결이 끊긴 동안 메시지가 항상 대기 상태로 유지되도록 하려면 로컬 브로커에서 메시지가 손실되지 않도록 하려면 이 옵션을 **false** 로 설정할 수 있습니다.

auto-delete-delay

로컬 브로커의 연결이 끊어지면 동적으로 생성된 원격 대기열을 자동으로 삭제하기 전에 이 속성의 값을 밀리초 단위로 지정합니다.

auto-delete-message-count

로컬 브로커의 연결이 끊어진 후에도 이 속성의 값은 큐를 자동으로 삭제할 수 있기 전에 동적으로 생성된 원격 대기열에 남아 있을 수 있는 최대 메시지 수를 지정합니다.

enable-divert-bindings

이 속성을 **true** 로 설정하면 요구에 따라 다양한 바인딩을 수신할 수 있습니다. 주소 정책에 포함된 주소와 일치하는 주소가 있는 다양한 바인딩이 있는 경우 다이버트의 전달 주소와 일치하는 큐 바인딩이 수요를 생성합니다. 기본값은 **false**입니다.

max-hops

페더레이션 중에 메시지가 만들 수 있는 최대 홉 수입니다. 특정 토폴로지에는 이 속성에 대한 특정 값이 필요합니다. 이러한 요구 사항에 대한 자세한 내용은 4.22.2절. “주소 페더레이션을 위한 공통 토폴로지” 을 참조하십시오.

Transformer-ref

변환기 구성의 이름입니다. 페더 메시지 전송 중에 메시지를 변환하려면 변환기 구성을 추가할 수 있습니다. **Transformer** 구성은 이 절차의 뒷부분에서 설명합니다.

4.

&lt ;address-policy& gt; 요소 내에서 주소 일치 패턴을 추가하여 주소 정책에서 주소를 포함 및 제외합니다. 예를 들면 다음과 같습니다.

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <address-policy name="news-address-federation" auto-delete="true" auto-delete-
delay="300000" auto-delete-message-count="-1" enable-divert-bindings="false" max-
hops="1" transformer-ref="news-transformer">

         <include address-match="queue.bbc.new" />
         <include address-match="queue.usatoday" />
         <include address-match="queue.news.#" />

         <exclude address-match="queue.news.sport.#" />
      </address-policy>

   </federation>
</federations>
```

포함

*이 요소의 **address-match** 속성 값은 주소 정책에 포함할 주소를 지정합니다. 정확한 주소를 지정할 수 있습니다(예: **queue.bbc.new** 또는 **queue.usatoday**). 또는 와일드카드 표현식을 사용하여 일치하는 주소 집합 을 지정할 수 있습니다. 이전 예에서 **address** 정책에는 문자열 **queue.news** 로 시작하는 모든 주소 이름도 포함됩니다.*

exclude

*이 요소의 **address-match** 속성 값은 주소 정책에서 제외할 주소를 지정합니다. 정확한 주소 이름을 지정하거나 와일드카드 표현식을 사용하여 일치하는 주소 집합 을 지정할 수 있습니다. 이전 예에서 주소 정책은 문자열 **queue.news.sport** 로 시작하는 모든 주소 이름을 제외합니다.*

5.

(선택 사항) 페더 요소 내에서 사용자 지정 변환기 구현을 참조하도록 변환기 요소를 추가합니다. 예를 들면 다음과 같습니다.

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <address-policy name="news-address-federation" auto-delete="true" auto-delete-
delay="300000" auto-delete-message-count="-1" enable-divert-bindings="false" max-
hops="1" transformer-ref="news-transformer">

         <include address-match="queue.bbc.new" />
         <include address-match="queue.usatoday" />
```

```
            <include address-match="queue.news.#" />

            <exclude address-match="queue.news.sport.#" />
        </address-policy>

        <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
        </transformer>

    </federation>
</federations>
```

name

변환기 구성의 이름입니다. 이 이름은 로컬 브로커에서 고유해야 합니다. 이는 주소 정책의 **transformer-ref** 속성 값으로 지정하는 이름입니다.

class-name

org.apache.activemq.artemis.core.server.transformer.Transformer 인터페이스를 구현하는 사용자 정의 클래스의 이름입니다.

transformer의 **transform()** 메서드는 메시지가 전송되기 전에 메시지와 함께 호출됩니다. 이를 통해 메시지 헤더 또는 본문이 페더레이션되기 전에 변환할 수 있습니다.

속성

특정 변환기 구성에 대한 키-값 쌍을 보유하는 데 사용됩니다.

6. 페더레이션 요소 내에서 하나 이상의 업스트림 요소를 추가합니다. 각 업스트림 요소는 원격 브로커에 대한 연결과 해당 연결에 적용할 정책을 정의합니다. 예를 들면 다음과 같습니다.

```
<federations>
    <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

        <upstream name="eu-east-1">
            <static-connectors>
                <connector-ref>eu-east-connector1</connector-ref>
            </static-connectors>
            <policy ref="news-address-federation"/>
        </upstream>

        <upstream name="eu-west-1" >
            <static-connectors>
                <connector-ref>eu-west-connector1</connector-ref>
            </static-connectors>
```

```
            <policy ref="news-address-federation"/>
        </upstream>

        <address-policy name="news-address-federation" auto-delete="true" auto-delete-
delay="300000" auto-delete-message-count="-1" enable-divert-bindings="false" max-
hops="1" transformer-ref="news-transformer">

            <include address-match="queue.bbc.new" />
            <include address-match="queue.usatoday" />
            <include address-match="queue.news.#" />

            <exclude address-match="queue.news.sport.#" />
        </address-policy>

        <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
        </transformer>

    </federation>
</federations>
```

static-connectors

*로컬 브로커의 **broker.xml** 구성 파일의 다른 위치에 정의된 커넥터 요소를 참조하는 **connector -ref** 요소 목록이 포함되어 있습니다. 커넥터는 발신 연결에 사용할 전송(**TCP, SSL, HTTP** 등) 및 서버 연결 매개 변수(호스트, 포트 등)를 정의합니다. 이 절차의 다음 단계에서는 **static-connectors** 요소에서 참조되는 커넥터를 추가하는 방법을 보여줍니다.*

policy-ref

업스트림 브로커에 적용되는 다운스트림 브로커에 구성된 주소 정책의 이름입니다.

업스트림 요소에 지정할 수 있는 추가 옵션은 다음과 같습니다.

name

*업스트림 브로커 구성의 이름입니다. 이 예에서 이름은 **eu-east-1** 및 **eu-west-1** 이라는 업스트림 브로커에 해당합니다.*

user

업스트림 브로커에 대한 연결을 생성할 때 사용할 사용자 이름입니다. 지정하지 않으면 페더레이션 요소의 구성에 지정된 공유 사용자 이름이 사용됩니다.

암호

*업스트림 브로커에 대한 연결을 생성할 때 사용할 암호입니다. 지정하지 않으면 **federation** 요소의 구성에 지정된 공유 암호가 사용됩니다.*

call-failover-timeout

__call-timeout__ 과 유사하지만 장애 조치(__failover__) 시도 중에 호출이 수행될 때 사용됩니다. 기본값은 __-1__ 이며, 이는 시간 초과가 비활성화되어 있음을 의미합니다.

call-timeout

페더레이션 연결은 차단 호출인 패킷을 전송할 때 원격 브로커의 응답을 기다리는 시간(밀리초)입니다. 이 시간이 지나면 연결에서 예외가 발생합니다. 기본값은 __30000__ 입니다.

check-period

로컬 브로커가 페더레이션 연결 상태를 확인하기 위해 원격 브로커로 보내는 연속 "유지 관리" 메시지 사이의 기간(밀리초)입니다. 페더레이션 연결이 정상이면 원격 브로커는 각 __keep-alive__ 메시지에 응답합니다. 연결이 비정상인 경우 다운스트림 브로커가 업스트림 브로커로부터 응답을 수신하지 못하면 회로 차단기 라는 메커니즘이 페더레이션 소비자를 차단하는 데 사용됩니다. 자세한 내용은 __circuit-breaker-timeout__ 매개변수에 대한 설명을 참조하십시오. __check-period__ 매개변수의 기본값은 __30000__ 입니다.

circuit-breaker-timeout

다운스트림과 업스트림 브로커 간의 단일 연결은 여러 페더레이션 대기열과 주소 소비자에 의해 공유될 수 있습니다. 브로커 간의 연결이 끊어지면 각 페더레이션 소비자가 동시에 다시 연결하려고 시도할 수 있습니다. 이를 방지하기 위해 회로 차단기 라는 메커니즘이 소비자를 차단합니다. 지정된 타임아웃 값이 경과하면 회로 차단기가 연결을 다시 표시합니다. 성공하면 소비자는 차단 해제됩니다. 그렇지 않으면 회로 차단기가 다시 적용됩니다.

connection-ttl

시간, 즉 페더레이션 연결이 원격 브로커에서 메시지 수신을 중지하면 활성 상태로 유지됩니다. 기본값은 __60000__ 입니다.

discovery-group-ref

업스트림 브로커에 대한 연결에 대한 정적 커넥터를 정의하는 대신 이 요소를 사용하여 __broker.xml__ 구성 파일의 다른 위치에 이미 구성된 검색 그룹을 지정할 수 있습니다. 특히 기존 검색 그룹을 이 요소의 __discovery-group-name__ 속성에 대한 값으로 지정합니다. 검색 그룹에 대한 자세한 내용은 __14.1.6절. “브로커 검색 방법”__ 을 참조하십시오.

ha

업스트림 브로커 연결에 고가용성이 활성화되어 있는지 여부를 지정합니다. 이 매개변수의 값이 __true__ 로 설정된 경우 로컬 브로커는 업스트림 클러스터에서 사용 가능한 브로커에 연결할 수 있으며 라이브 업스트림 브로커가 종료되면 백업 브로커에 자동으로 실패할 수 있습니다. 기본값은 __false__입니다.

initial-connect-attempts

다운스트림 브로커가 업스트림 브로커에 연결하기 위해 생성하는 초기 시도 횟수입니

다. 연결을 설정하지 않고 이 값에 도달하면 업스트림 브로커가 영구적으로 오프라인으로 간주됩니다. 다운스트림 브로커는 더 이상 업스트림 브로커로 메시지를 라우팅하지 않습니다. 기본값은 **-1** 이며 이는 제한이 없음을 의미합니다.

max-retry-interval

원격 브로커에 연결할 때 후속 재연결 사이의 최대 시간(밀리초)입니다. 기본값은 **2000** 입니다.

재연결**-attempts**

연결에 실패하면 다운스트림 브로커가 업스트림 브로커에 다시 연결하려고 시도하는 횟수입니다. 연결을 다시 설정하지 않고 이 값에 도달하면 업스트림 브로커는 영구적으로 오프라인으로 간주됩니다. 다운스트림 브로커는 더 이상 업스트림 브로커로 메시지를 라우팅하지 않습니다. 기본값은 **-1** 이며 이는 제한이 없음을 의미합니다.

retry-interval

원격 브로커에 대한 연결이 실패한 경우 후속 재연결 시도 사이의 기간(밀리초)입니다. 기본값은 **500** 입니다.

retry-interval-multiplier

retry-interval 매개변수 값에 적용되는 인수를 곱합니다. 기본값은 **1** 입니다.

share-connection

다운스트림 및 업스트림 연결이 모두 동일한 브로커에 대해 구성된 경우 다운스트림 및 업스트림 구성이 모두 **true** 로 설정된 한 동일한 연결이 공유됩니다. 기본값은 **false**입니다.

7.

로컬 브로커에서 원격 브로커에 커넥터를 추가합니다. 이러한 커넥터는 페더레이션 주소 구성의 정적 연결 요소에서 참조되는 커넥터 입니다. 예를 들면 다음과 같습니다.

```
<connectors>
  <connector name="eu-west-1-connector">tcp://localhost:61616</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617</connector>
</connectors>
```

4.22.4.2.3. 다운스트림 주소 페더레이션 구성

다음 예제에서는 독립 실행형 브로커에 대한 다운스트림 주소 페더레이션을 구성하는 방법을 보여줍니다.

다운스트림 주소 페더레이션을 사용하면 하나 이상의 원격 브로커가 로컬 브로커에 다시 연결하는 데 사용하는 로컬 브로커에 구성을 추가할 수 있습니다. 이 접근 방식의 장점은 모든 페더레이션 구성을 단일

브로커에 유지할 수 있다는 것입니다. 예를 들어 **hub-and-spoke** 토폴로지에 유용한 접근 방식이 될 수 있습니다.

참고

다운스트림 주소 페더레이션은 페더레이션 연결 방향과 업스트림 주소 구성을 대체합니다. 따라서 구성에 원격 브로커를 추가하면 다운스트림 브로커로 간주됩니다. 다운스트림 브로커는 구성의 연결 정보를 사용하여 이제 업스트림으로 간주되는 로컬 브로커에 다시 연결합니다. 이 문제는 원격 브로커에 대한 구성을 추가할 때 이 예제의 뒷부분에서 설명합니다.

사전 요구 사항

- 업스트림 주소 페더레이션 구성에 대해 잘 알고 있어야 합니다. 4.22.4.2.2절. "업스트림 주소 페더레이션 구성"을 참조하십시오.
- 다음 예제에서는 독립 실행형 브로커 간에 주소 페더레이션을 구성하는 방법을 보여줍니다. 그러나 브로커 클러스터에 대한 페더레이션 구성을 위한 요구 사항도 숙지해야 합니다. 자세한 내용은 4.22.4.2.1절. "브로커 클러스터에 대한 페더레이션 구성"의 내용을 참조하십시오.

프로세스

1. 로컬 브로커에서 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.
2. **< federation>** 요소를 포함하는 **< federations >** 요소를 추가합니다. 예를 들면 다음과 같습니다.

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
   </federation>
</federations>
```

3. 주소 정책 구성을 추가합니다. 예를 들면 다음과 같습니다.

```
<federations>
   ...
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <address-policy name="news-address-federation" max-hops="1" auto-delete="true"
auto-delete-delay="300000" auto-delete-message-count="-1" transformer-ref="news-
```

```
transformer">

            <include address-match="queue.bbc.new" />
            <include address-match="queue.usatoday" />
            <include address-match="queue.news.#" />

            <exclude address-match="queue.news.sport.#" />
        </address-policy>

    </federation>
  ...
</federations>
```

4. *전송 전에 메시지를 변환하려면 변환기 구성을 추가합니다. 예를 들면 다음과 같습니다.*

```
<federations>
    ...
    <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

        <address-policy name="news-address-federation" max-hops="1" auto-delete="true"
auto-delete-delay="300000" auto-delete-message-count="-1" transformer-ref="news-
transformer">

            <include address-match="queue.bbc.new" />
            <include address-match="queue.usatoday" />
            <include address-match="queue.news.#" />

            <exclude address-match="queue.news.sport.#" />
        </address-policy>

        <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
        </transformer>

    </federation>
  ...
</federations>
```

5. *각 원격 브로커에 대한 다운스트림 요소를 추가합니다. 예를 들면 다음과 같습니다.*

```
<federations>
    ...
    <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

        <downstream name="eu-east-1">
            <static-connectors>
                <connector-ref>eu-east-connector1</connector-ref>
```

```
            </static-connectors>
            <upstream-connector-ref>netty-connector</upstream-connector-ref>
            <policy ref="news-address-federation"/>
        </downstream>

        <downstream name="eu-west-1" >
            <static-connectors>
               <connector-ref>eu-west-connector1</connector-ref>
            </static-connectors>
            <upstream-connector-ref>netty-connector</upstream-connector-ref>
            <policy ref="news-address-federation"/>
        </downstream>

        <address-policy name="news-address-federation" max-hops="1" auto-delete="true"
auto-delete-delay="300000" auto-delete-message-count="-1" transformer-ref="news-
transformer">
            <include address-match="queue.bbc.new" />
            <include address-match="queue.usatoday" />
            <include address-match="queue.news.#" />

            <exclude address-match="queue.news.sport.#" />
        </address-policy>

        <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
        </transformer>

    </federation>
 ...
</federations>
```

이전 구성에 표시된 대로 원격 브로커는 이제 로컬 브로커의 다운스트림으로 간주됩니다. 다운스트림 브로커는 구성의 연결 정보를 사용하여 로컬(즉, 업스트림) 브로커에 다시 연결합니다.

6.

로컬 브로커에서 로컬 및 원격 브로커가 페더레이션 연결을 설정하는 데 사용하는 커넥터와 수락을 추가합니다. 예를 들면 다음과 같습니다.

```
<connectors>
  <connector name="netty-connector">tcp://localhost:61616</connector>
  <connector name="eu-west-1-connector">tcp://localhost:61616</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617</connector>
</connectors>

<acceptors>
  <acceptor name="netty-acceptor">tcp://localhost:61616</acceptor>
</acceptors>
```

connector name="netty-connector"

로컬 브로커가 원격 브로커에 전송하는 커넥터 구성입니다. 원격 브로커는 이 구성을 사용하여 로컬 브로커에 다시 연결합니다.

connector name="eu-west-1-connector", connector name="eu-east-1-connector"

원격 브로커에 연결합니다. 로컬 브로커는 이러한 커넥터를 사용하여 원격 브로커에 연결하고 원격 브로커가 로컬 브로커에 다시 연결하는 데 필요한 구성을 공유합니다.

acceptor name="netty-acceptor"

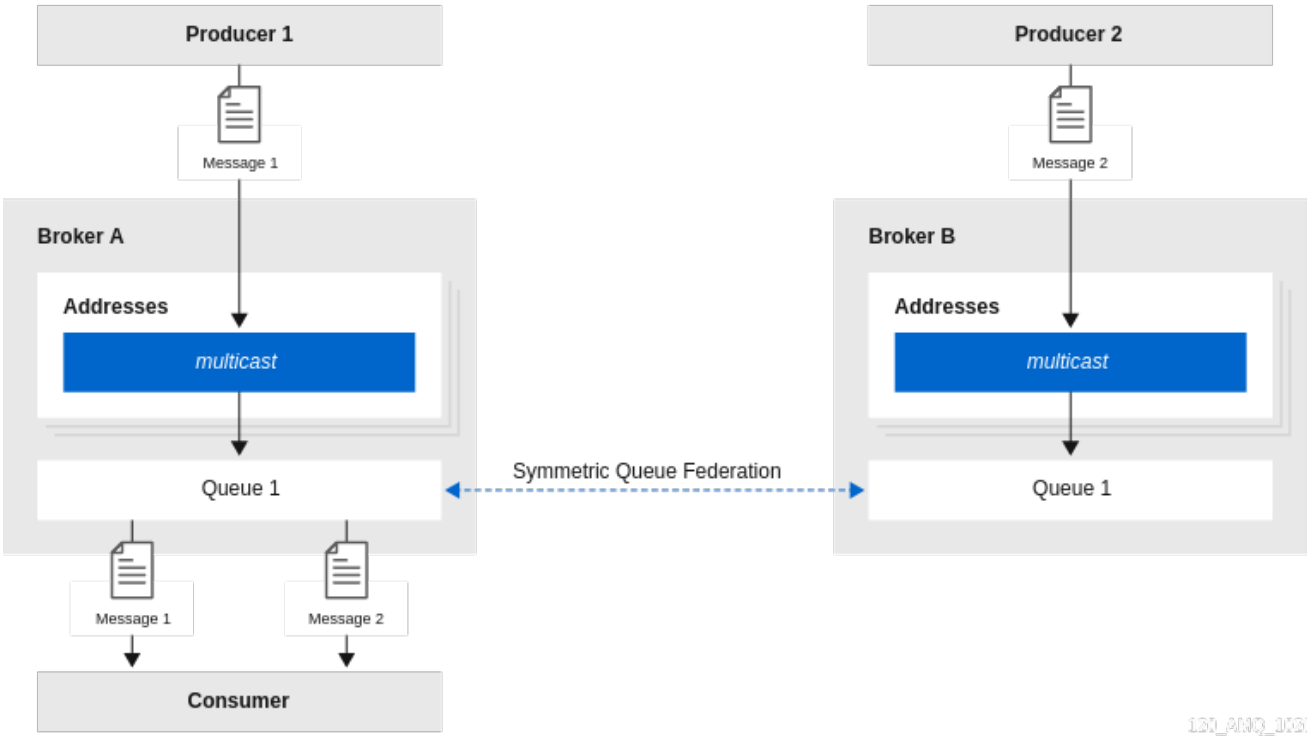
원격 브로커가 로컬 브로커에 다시 연결하는 데 사용하는 커넥터에 해당하는 로컬 브로커의 수락자.

4.22.4.2.4. 큐 페더레이션 정보

큐 페더레이션은 다른 원격 브로커의 로컬 브로커에서 단일 큐의 부하를 분산하는 방법을 제공합니다.

부하 분산을 달성하기 위해 로컬 브로커는 로컬 소비자의 메시지에 대한 요구를 충족하기 위해 원격 대기열에서 메시지를 검색합니다. 예를 들면 다음과 같습니다.

그림 4.9. 대칭 큐 페더레이션



원격 큐를 재구성할 필요가 없으며 동일한 브로커 또는 동일한 클러스터에 있을 필요가 없습니다. 원격 링크를 설정하는 데 필요한 모든 구성이 로컬 브로커에 있습니다.

4.22.4.2.4.1. 큐 페더레이션의 이점

아래에 설명 된 몇 가지 이유는 큐 페더레이션을 구성하도록 선택할 수 있습니다.

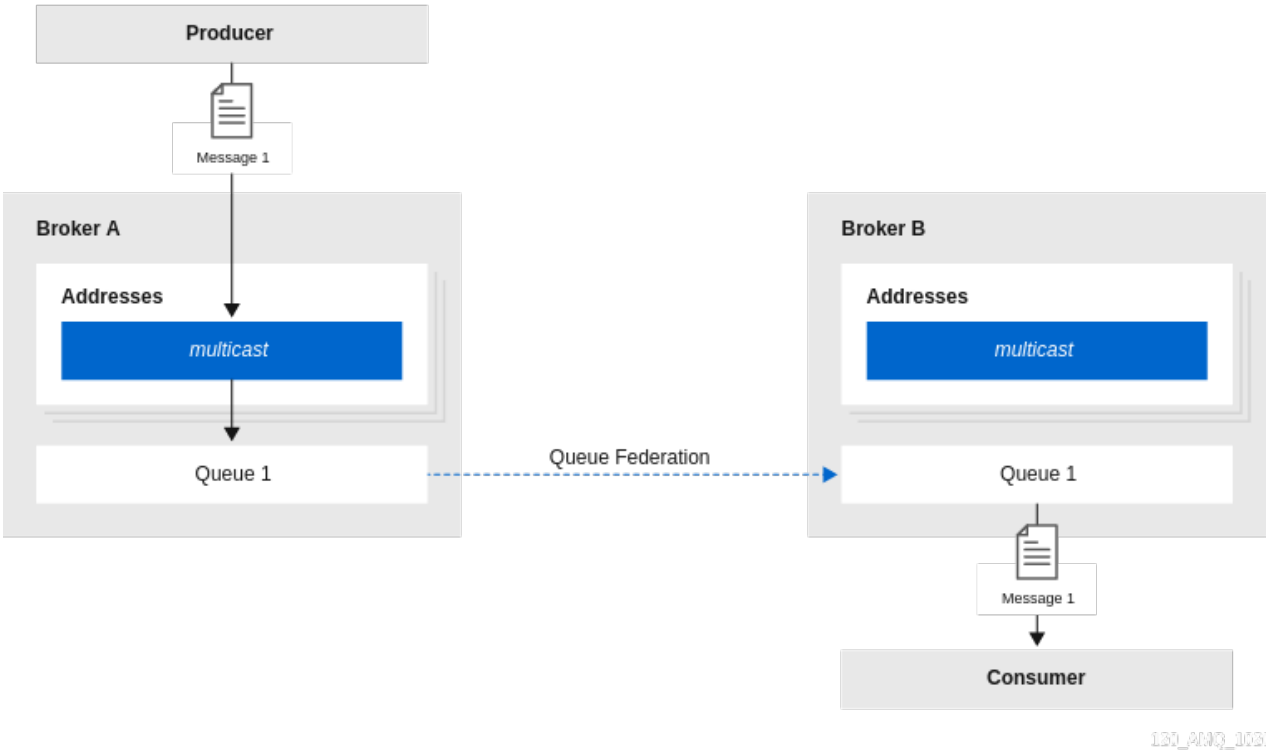
용량 증가

큐 페더레이션은 많은 브로커에 분산되는 "논리적" 큐를 만들 수 있습니다. 이 논리 분산 큐는 단일 브로커의 단일 큐보다 용량이 훨씬 높습니다. 이 설정에서 가능한 많은 메시지가 원래 게시 된 브로커에서 소비됩니다. 시스템은 부하 분산이 필요한 경우에만 페더레이션에서 메시지를 이동합니다.

다중 리전 설정 배포

다중 지역 설정에서는 한 지역 또는 장소에 메시지 생산자가 있을 수 있으며 다른 지역에는 소비자가 있을 수 있습니다. 그러나 지정된 지역의 생산자 및 소비자 연결을 로컬로 유지하는 것이 이상적이어야 합니다. 이 경우 생산자와 소비자인 각 리전에 브로커를 배포하고 큐 페더레이션을 사용하여 지역 간에 메시지를 이동할 수 있습니다. 예를 들면 다음과 같습니다.

그림 4.10. 다중 지역 대기열 페더레이션



*안전한 엔터프라이즈 **LAN**과 **DMZ** 간 통신*

*네트워킹 보안에서 **DMZ**(분밀 화된 영역)는 엔터프라이즈의 외부 연결 서비스를 포함하는 물리적 또는 논리 서브네트워크이며 일반적으로 인터넷과 같이 더 큰 네트워크를 신뢰할 수 없는 네트워크에 노출합니다. 엔터프라이즈의 나머지 **LAN**(**Local Area Network**)은 방화벽 뒤에서 이 외부 네트워크와 격리되어 있습니다.*

다수의 메시지 생산자가 DMZ에 있고 보안 엔터프라이즈 LAN의 많은 소비자가 있는 경우 생산자가 보안 엔터프라이즈 LAN의 브로커에 연결할 수 있도록 하는 것이 적절하지 않을 수 있습니다. 이 경우 생산자가 메시지를 게시할 수 있는 DMZ에 브로커를 배포할 수 있습니다. 그런 다음 엔터프라이즈 LAN의 브로커는 DMZ의 브로커에 연결하고 페더레이션 대기열을 사용하여 DMZ 브로커에서 메시지를 수신할 수 있습니다.

4.22.4.2.5. 업스트림 큐 페더레이션 구성

다음 예제에서는 독립 실행형 브로커에 대해 업스트림 큐 페더레이션을 구성하는 방법을 보여줍니다. 이 예에서는 로컬(즉, 다운스트림) 브로커에서 일부 원격(즉, 업스트림) 브로커로 페더레이션을 구성합니다.

사전 요구 사항

- 다음 예제에서는 독립 실행형 브로커 간에 큐 페더레이션을 구성하는 방법을 보여줍니다. 그러나 브로커 클러스터에 대한 페더레이션 구성을 위한 요구 사항도 숙지해야 합니다. 자세한 내용은 4.22.4.2.1절. “브로커 클러스터에 대한 페더레이션 구성”의 내용을 참조하십시오.

프로세스

1. <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. 새로운 < federations > 요소 내에서 < federation> 요소를 추가합니다. 예를 들면 다음과 같습니다.

```
<federations>
  <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
  </federation>
</federations>
```

name

페더레이션 구성의 이름입니다. 이 예에서 이름은 다운스트림 브로커의 이름에 해당합니다.

user

업스트림 브로커에 연결하기 위한 공유 사용자 이름입니다.

암호

업스트림 브로커에 연결하기 위한 공유 암호입니다.

참고

- 업스트림 브로커의 경우 사용자 및 암호 인증 정보가 다른 경우 구성에 추가할 때 해당 브로커의 인증 정보를 별도로 지정할 수 있습니다. 이 내용은 이 절차의 뒷부분에서 설명합니다.

3. **federation** 요소 내에서 **< queue-policy>** 요소를 추가합니다. **< queue-policy>** 요소의 속성 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<federations>
  <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

    <queue-policy name="news-queue-federation" include-federated="true" priority-
adjustment="-5" transformer-ref="news-transformer">
    </queue-policy>

  </federation>
</federations>
```

name

큐 정책의 이름입니다. 브로커에 구성된 모든 대기열 정책에는 고유한 이름이 있어야 합니다.

include-federated

이 속성의 값이 **false** 로 설정되면 구성이 이미 제공된 소비자(즉, 페더레이션 대기열의 소비자)를 다시 제공하지 않습니다. 이렇게 하면 대칭 또는 **closed-loop** 토폴로지에서 공급되지 않은 소비자가 없고 메시지가 시스템 전체에서 지속적으로 전달되는 상황을 방지할 수 있습니다.

closed-loop 토폴로지가 없는 경우 이 속성의 값을 **true** 로 설정할 수 있습니다. 예를 들어 **BrokerA,BrokerB** 및 **BrokerC** 의 세 가지 브로커 체인과 **BrokerA** 의 생산자와 **BrokerC** 의 소비자가 있다고 가정합니다. 이 경우 **BrokerB** 가 소비자를 **BrokerA** 에 다시 공급하기를 원할 것입니다.

priority-adjustment

소비자가 큐에 연결하면 업스트림(즉, 페더레이션) 소비자가 생성될 때 우선 순위가 사용됩니다. 페더레이션 소비자의 우선순위는 우선순위 조정 속성의 값에 의해 조정 됩니다. 이 속성의 기본값은 **-1** 이며, 이는 로컬 소비자가 로드 밸런싱 중에 페더레이션 소비자보다 우선 순위를 갖도록 합니다. 그러나 필요에 따라 우선순위 조정의 값을 변경할 수 있습니다.

우선 순위 조정이 너무 많은 메시지가 페더레이션 소비자로 이동하는 것을 방지하기에 불충분하여 브로커 간에 메시지를 이동할 수 있는 경우 페더레이션 소비자로 이동하는 메시지 배치 크기를 제한할 수 있습니다. 배치 크기를 제한하려면 페더레이션 소비자의 연결 **URI**에서 **consumerWindowSize** 값을 **0** 으로 설정합니다.

```
tcp://<host>:<port>?consumerWindowSize=0
```

consumerWindowSize 값을 **0** 으로 설정하면 **AMQ Broker**는 일치하는 주소에 대한 주소 설정에 **defaultConsumerWindowSize** 매개변수 값을 사용하여 브로커 간에 이동할 수 있는 메시지의 배치 크기를 결정합니다. **defaultConsumerWindowSize** 속성의 기본값은 **1048576** 바이트입니다.

Transformer-ref

변환기 구성의 이름입니다. 페더 메시지 전송 중에 메시지를 변환하려면 변환기 구성을 추가할 수 있습니다. **Transformer** 구성은 이 절차의 뒷부분에서 설명합니다.

4.

&lt ;queue-policy > 요소 내에서 큐 정책에서 주소를 포함 및 제외하기 위해 주소 일치 패턴을 추가합니다. 예를 들면 다음과 같습니다.

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <queue-policy name="news-queue-federation" include-federated="true" priority-
adjustment="-5" transformer-ref="news-transformer">

         <include queue-match="#" address-match="queue.bbc.new" />
         <include queue-match="#" address-match="queue.usatoday" />
         <include queue-match="#" address-match="queue.news.#" />

         <exclude queue-match="#.local" address-match="#" />

      </queue-policy>

   </federation>
</federations>
```

포함

이 요소의 **address-match** 속성 값은 큐 정책에 포함할 주소를 지정합니다. 정확한 주소를 지정할 수 있습니다(예: **queue.bbc.new** 또는 **queue.usatoday**). 또는 와일드카드 표현식을 사용하여 일치하는 주소 집합 을 지정할 수 있습니다. 이전 예에서 큐 정책에는 문자열 **queue.news** 로 시작하는 모든 주소 이름도 포함됩니다.

address-match 속성과 함께 **queue-match** 속성을 사용하여 큐 정책의 해당 주소에

*특정 큐를 포함할 수 있습니다. **address-match** 속성과 마찬가지로 정확한 큐 이름을 지정하거나 와일드카드 표현식을 사용하여 큐 집합 을 지정할 수 있습니다. 이전 예에서 숫자 기호(**#**) 와일드카드 문자는 각 주소의 모든 큐 또는 주소 집합의 모든 큐가 큐 정책에 포함되어 있음을 나타냅니다.*

exclude

*이 요소의 **address-match** 속성 값은 큐 정책에서 제외할 주소를 지정합니다. 정확한 주소를 지정하거나 와일드카드 표현식을 사용하여 일치하는 주소 집합 을 지정할 수 있습니다. 이전 예에서 숫자 기호(**#**) 와일드카드 문자는 모든 주소의 **queue-match** 속성과 일치하는 모든 큐가 제외됨을 의미합니다. 이 경우 **.local** 문자열로 끝나는 모든 큐가 제외됩니다. 이는 특정 대기열이 페더레이션이 아닌 로컬 큐로 유지됨을 나타냅니다.*

5. *페더 요소 내에서 사용자 지정 변환기 구현을 참조하는 변환기 요소를 추가합니다. 예를 들면 다음과 같습니다.*

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <queue-policy name="news-queue-federation" include-federated="true" priority-
adjustment="-5" transformer-ref="news-transformer">

         <include queue-match="#" address-match="queue.bbc.new" />
         <include queue-match="#" address-match="queue.usatoday" />
         <include queue-match="#" address-match="queue.news.#" />

         <exclude queue-match="#.local" address-match="#" />

      </queue-policy>

      <transformer name="news-transformer">
         <class-name>org.myorg.NewsTransformer</class-name>
         <property key="key1" value="value1"/>
         <property key="key2" value="value2"/>
      </transformer>

   </federation>
</federations>
```

name

*변환기 구성의 이름입니다. 이 이름은 해당 브로커에서 고유해야 합니다. 이 이름을 주소 정책의 **transformer-ref** 속성 값으로 지정합니다.*

class-name

***org.apache.activemq.artemis.core.server.transformer.Transformer** 인터페이스를 구현하는 사용자 정의 클래스의 이름입니다.*

transformer의 transform() 메서드는 메시지가 전송되기 전에 메시지와 함께 호출됩니다. 이를 통해 메시지 헤더 또는 본문이 페더레이션되기 전에 변환할 수 있습니다.

속성

특정 변환기 구성에 대한 키-값 쌍을 보유하는 데 사용됩니다.

6. *페더레이션 요소 내에서 하나 이상의 업스트림 요소를 추가합니다. 각 업스트림 요소는 업스트림 브로커 연결과 해당 연결에 적용할 정책을 정의합니다. 예를 들면 다음과 같습니다.*

```
<federations>
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <upstream name="eu-east-1">
         <static-connectors>
            <connector-ref>eu-east-connector1</connector-ref>
         </static-connectors>
         <policy ref="news-queue-federation"/>
      </upstream>

      <upstream name="eu-west-1" >
         <static-connectors>
            <connector-ref>eu-west-connector1</connector-ref>
         </static-connectors>
         <policy ref="news-queue-federation"/>
      </upstream>

      <queue-policy name="news-queue-federation" include-federated="true" priority-
adjustment="-5" transformer-ref="news-transformer">

         <include queue-match="#" address-match="queue.bbc.new" />
         <include queue-match="#" address-match="queue.usatoday" />
         <include queue-match="#" address-match="queue.news.#" />

         <exclude queue-match="#.local" address-match="#" />

      </queue-policy>

      <transformer name="news-transformer">
         <class-name>org.myorg.NewsTransformer</class-name>
         <property key="key1" value="value1"/>
         <property key="key2" value="value2"/>
      </transformer>

   </federation>
</federations>
```

static-connectors

로컬 브로커의 **broker.xml** 구성 파일의 다른 위치에 정의된 커넥터 요소를 참조하는 **connector -ref** 요소 목록이 포함되어 있습니다. 커넥터는 발신 연결에 사용할 전송(**TCP, SSL, HTTP** 등) 및 서버 연결 매개 변수(호스트, 포트 등)를 정의합니다. 이 절차의 다음 단계에서는 페더레이션 대기열 구성의 정적 연결 요소에서 참조하는 커넥터 를 추가하는 방법을 보여줍니다.

policy-ref

업스트림 브로커에 적용되는 다운스트림 브로커에 구성된 큐 정책의 이름입니다.

업스트림 요소에 지정할 수 있는 추가 옵션은 다음과 같습니다.

name

업스트림 브로커 구성의 이름입니다. 이 예에서 이름은 **eu-east-1** 및 **eu-west-1** 이라는 업스트림 브로커에 해당합니다.

user

업스트림 브로커에 대한 연결을 생성할 때 사용할 사용자 이름입니다. 지정하지 않으면 페더레이션 요소의 구성에 지정된 공유 사용자 이름이 사용됩니다.

암호

업스트림 브로커에 대한 연결을 생성할 때 사용할 암호입니다. 지정하지 않으면 **federation** 요소의 구성에 지정된 공유 암호가 사용됩니다.

call-failover-timeout

call-timeout 과 유사하지만 장애 조치(**failover**) 시도 중에 호출이 수행될 때 사용됩니다. 기본값은 **-1** 이며, 이는 시간 초과가 비활성화되어 있음을 의미합니다.

call-timeout

페더레이션 연결은 차단 호출인 패킷을 전송할 때 원격 브로커의 응답을 기다리는 시간(밀리초)입니다. 이 시간이 지나면 연결에서 예외가 발생합니다. 기본값은 **30000** 입니다.

check-period

로컬 브로커가 페더레이션 연결 상태를 확인하기 위해 원격 브로커로 보내는 연속 "유지 관리" 메시지 사이의 기간(밀리초)입니다. 페더레이션 연결이 정상이면 원격 브로커는 각 **keep-alive** 메시지에 응답합니다. 연결이 비정상인 경우 다운스트림 브로커가 업스트림 브로커로부터 응답을 수신하지 못하면 회로 차단기 라는 메커니즘이 페더레이션 소비자를 차단하는 데 사용됩니다. 자세한 내용은 **circuit-breaker-timeout** 매개변수에 대한 설명을 참조하십시오. **check-period** 매개변수의 기본값은 **30000** 입니다.

circuit-breaker-timeout

다운스트림과 업스트림 브로커 간의 단일 연결은 여러 페더레이션 대기열과 주소 소비자에 의해 공유될 수 있습니다. 브로커 간의 연결이 끊어지면 각 페더레이션 소비자가 동시에 다시 연결하려고 시도할 수 있습니다. 이를 방지하기 위해 회로 차단기 라는 메커니즘이 소비자를 차단합니다. 지정된 타임아웃 값이 경과하면 회로 차단기가 연결을 다시 표시합니다. 성공하면 소비자는 차단 해제됩니다. 그렇지 않으면 회로 차단기가 다시 적용됩니다.

connection-ttl

*시간, 즉 페더레이션 연결이 원격 브로커에서 메시지 수신을 중지하면 활성 상태로 유지됩니다. 기본값은 **60000** 입니다.*

discovery-group-ref

*업스트림 브로커에 대한 연결에 대한 정적 커넥터를 정의하는 대신 이 요소를 사용하여 **broker.xml** 구성 파일의 다른 위치에 이미 구성된 검색 그룹을 지정할 수 있습니다. 특히 기존 검색 그룹을 이 요소의 **discovery-group-name** 속성에 대한 값으로 지정합니다. 검색 그룹에 대한 자세한 내용은 **14.1.6절. "브로커 검색 방법"** 을 참조하십시오.*

ha

*업스트림 브로커 연결에 고가용성이 활성화되어 있는지 여부를 지정합니다. 이 매개변수의 값이 **true** 로 설정된 경우 로컬 브로커는 업스트림 클러스터에서 사용 가능한 브로커에 연결할 수 있으며 라이브 업스트림 브로커가 종료되면 백업 브로커에 자동으로 실패할 수 있습니다. 기본값은 **false**입니다.*

initial-connect-attempts

*다운스트림 브로커가 업스트림 브로커에 연결하기 위해 생성하는 초기 시도 횟수입니다. 연결을 설정하지 않고 이 값에 도달하면 업스트림 브로커가 영구적으로 오프라인으로 간주됩니다. 다운스트림 브로커는 더 이상 업스트림 브로커로 메시지를 라우팅하지 않습니다. 기본값은 **-1** 이며 이는 제한이 없음을 의미합니다.*

max-retry-interval

*원격 브로커에 연결할 때 후속 재연결 사이의 최대 시간(밀리초)입니다. 기본값은 **2000** 입니다.*

재연결-attempts

*연결에 실패하면 다운스트림 브로커가 업스트림 브로커에 다시 연결하려고 시도하는 횟수입니다. 연결을 다시 설정하지 않고 이 값에 도달하면 업스트림 브로커는 영구적으로 오프라인으로 간주됩니다. 다운스트림 브로커는 더 이상 업스트림 브로커로 메시지를 라우팅하지 않습니다. 기본값은 **-1** 이며 이는 제한이 없음을 의미합니다.*

retry-interval

*원격 브로커에 대한 연결이 실패한 경우 후속 재연결 시도 사이의 기간(밀리초)입니다. 기본값은 **500** 입니다.*

retry-interval-multiplier

retry-interval 매개변수 값에 적용되는 인수를 곱합니다. 기본값은 **1** 입니다.

share-connection

다운스트림 및 업스트림 연결이 모두 동일한 브로커에 대해 구성된 경우 다운스트림 및 업스트림 구성이 모두 **true** 로 설정된 한 동일한 연결이 공유됩니다. 기본값은 **false**입니다.

7.

로컬 브로커에서 원격 브로커에 커넥터를 추가합니다. 이러한 커넥터는 페더레이션 주소 구성의 정적 연결 요소에서 참조되는 커넥터 입니다. 예를 들면 다음과 같습니다.

```
<connectors>
  <connector name="eu-west-1-connector">tcp://localhost:61616</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617</connector>
</connectors>
```

참고

대용량 메시지가 페더레이션 연결을 통과하도록 하려면 페더레이션 연결에 사용되는 커넥터에서 **consumerWindowSize** 매개변수를 **-1**로 설정합니다. **consumerWindowSize** 매개변수를 **-1**로 설정하면 이 매개변수에 대한 제한이 설정되어 있지 않으므로 대규모 메시지가 연결을 통과할 수 있습니다. 예를 들면 다음과 같습니다.

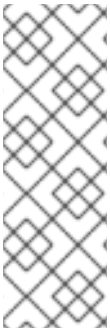
```
<connectors>
  <connector name="eu-west-1-connector">tcp://localhost:61616?
consumerWindowSize=-1</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617?consumerWindowSize=-
1</connector>
</connectors>
```

대규모 메시지에 대한 자세한 내용은 *8장. 대용량 메시지 처리* 을 참조하십시오.

4.22.4.2.6. 다운스트림 대기열 페더레이션 구성

다음 예제에서는 다운스트림 큐 페더레이션을 구성하는 방법을 보여줍니다.

다운스트림 큐 페더레이션을 사용하면 하나 이상의 원격 브로커가 로컬 브로커에 다시 연결하는 데 사용하는 로컬 브로커에 구성을 추가할 수 있습니다. 이 접근 방식의 장점은 모든 페더레이션 구성을 단일 브로커에 유지할 수 있다는 것입니다. 예를 들어 **hub-and-spoke** 토폴로지에 유용한 접근 방식이 될 수 있습니다.



참고

다운스트림 큐 페더레이션은 페더레이션 연결 방향과 업스트림 대기열 구성의 방향을 되돌립니다. 따라서 구성에 원격 브로커를 추가하면 다운스트림 브로커로 간주됩니다. 다운스트림 브로커는 구성의 연결 정보를 사용하여 이제 업스트림으로 간주되는 로컬 브로커에 다시 연결합니다. 이 문제는 원격 브로커에 대한 구성을 추가할 때 이 예제의 뒷부분에서 설명합니다.

사전 요구 사항

- 업스트림 큐 페더레이션 구성에 대해 잘 알고 있어야 합니다. 4.22.4.2.5절. “업스트림 큐 페더레이션 구성”을 참조하십시오.

- 다음 예제에서는 독립 실행형 브로커 간에 큐 페더레이션을 구성하는 방법을 보여줍니다. 그러나 브로커 클러스터에 대한 페더레이션 구성을 위한 요구 사항도 숙지해야 합니다. 자세한 내용은 4.22.4.2.1절. “브로커 클러스터에 대한 페더레이션 구성”의 내용을 참조하십시오.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **< federation>** 요소를 포함하는 **< federations >** 요소를 추가합니다. 예를 들면 다음과 같습니다.

```
<federations>
  <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">
  </federation>
</federations>
```

3. 큐 정책 구성을 추가합니다. 예를 들면 다음과 같습니다.

```
<federations>
   ...
   <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

      <queue-policy name="news-queue-federation" priority-adjustment="-5" include-
federated="true" transformer-ref="new-transformer">

         <include queue-match="#" address-match="queue.bbc.new" />
         <include queue-match="#" address-match="queue.usatoday" />
         <include queue-match="#" address-match="queue.news.#" />
```

```
            <exclude queue-match="#.local" address-match="#" />

        </queue-policy>

    </federation>
  ...
</federations>
```

4. *전송 전에 메시지를 변환하려면 변환기 구성을 추가합니다. 예를 들면 다음과 같습니다.*

```
<federations>
    ...
    <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

        <queue-policy name="news-queue-federation" priority-adjustment="-5" include-
federated="true" transformer-ref="news-transformer">

            <include queue-match="#" address-match="queue.bbc.new" />
            <include queue-match="#" address-match="queue.usatoday" />
            <include queue-match="#" address-match="queue.news.#" />

            <exclude queue-match="#.local" address-match="#" />

        </queue-policy>

        <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
        </transformer>

    </federation>
  ...
</federations>
```

5. *각 원격 브로커에 대한 다운스트림 요소를 추가합니다. 예를 들면 다음과 같습니다.*

```
<federations>
    ...
    <federation name="eu-north-1" user="federation_username"
password="32a10275cf4ab4e9">

        <downstream name="eu-east-1">
            <static-connectors>
                <connector-ref>eu-east-connector1</connector-ref>
            </static-connectors>
            <upstream-connector-ref>netty-connector</upstream-connector-ref>
            <policy ref="news-address-federation"/>
        </downstream>
```

```
        <downstream name="eu-west-1" >
            <static-connectors>
                <connector-ref>eu-west-connector1</connector-ref>
            </static-connectors>
            <upstream-connector-ref>netty-connector</upstream-connector-ref>
            <policy ref="news-address-federation"/>
        </downstream>

        <queue-policy name="news-queue-federation" priority-adjustment="-5" include-
federated="true" transformer-ref="new-transformer">

            <include queue-match="#" address-match="queue.bbc.new" />
            <include queue-match="#" address-match="queue.usatoday" />
            <include queue-match="#" address-match="queue.news.#" />

            <exclude queue-match="#.local" address-match="#" />

        </queue-policy>

        <transformer name="news-transformer">
            <class-name>org.myorg.NewsTransformer</class-name>
            <property key="key1" value="value1"/>
            <property key="key2" value="value2"/>
        </transformer>

    </federation>
  ...
</federations>
```

이전 구성에 표시된 대로 원격 브로커는 이제 로컬 브로커의 다운스트림으로 간주됩니다. 다운스트림 브로커는 구성의 연결 정보를 사용하여 로컬(즉, 업스트림) 브로커에 다시 연결합니다.

6. *로컬 브로커에서 로컬 및 원격 브로커가 페더레이션 연결을 설정하는 데 사용하는 커넥터와 수락을 추가합니다. 예를 들면 다음과 같습니다.*

```
<connectors>
  <connector name="netty-connector">tcp://localhost:61616</connector>
  <connector name="eu-west-1-connector">tcp://localhost:61616</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617</connector>
</connectors>

<acceptors>
  <acceptor name="netty-acceptor">tcp://localhost:61616</acceptor>
</acceptors>
```

connector name="netty-connector"

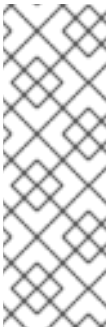
로컬 브로커가 원격 브로커에 전송하는 커넥터 구성입니다. 원격 브로커는 이 구성을 사용하여 로컬 브로커에 다시 연결합니다.

connector name="eu-west-1-connector" , connector name="eu-east-1-connector"

원격 브로커에 연결합니다. 로컬 브로커는 이러한 커넥터를 사용하여 원격 브로커에 연결하고 원격 브로커가 로컬 브로커에 다시 연결하는 데 필요한 구성을 공유합니다.

acceptor name="netty-acceptor"

원격 브로커가 로컬 브로커에 다시 연결하는 데 사용하는 커넥터에 해당하는 로컬 브로커의 수락자.



참고

페더레이션 연결을 통해 많은 메시지가 전송되도록 하려면 페더레이션 연결에 사용되는 커넥터에서 **consumerWindowSize** 매개변수를 **-1**로 설정합니다. **consumerWindowSize** 매개변수를 **-1**로 설정하면 이 매개변수에 대한 제한이 설정되어 있지 않으므로 대규모 메시지가 연결을 통과할 수 있습니다. 예를 들면 다음과 같습니다.

```
<connectors>
  <connector name="eu-west-1-connector">tcp://localhost:61616?
consumerWindowSize=-1</connector>
  <connector name="eu-east-1-connector">tcp://localhost:61617?
consumerWindowSize=-1</connector>
</connectors>
```

대규모 메시지에 대한 자세한 내용은 **8장. 대용량 메시지 처리** 을 참조하십시오.

5장. 브로커 보안

5.1. 연결 보안

브로커가 메시징 클라이언트에 연결되거나 브로커가 다른 브로커에 연결된 경우 **TLS(Transport Layer Security)**를 사용하여 이러한 연결을 보호할 수 있습니다.

다음 두 가지 **TLS** 구성을 사용할 수 있습니다.

- 단방향 **TLS:** 브로커만 인증서를 제공합니다. 가장 일반적인 구성입니다.
- 두 방향(또는 상호) **TLS:** 브로커와 클라이언트(또는 기타 브로커) 모두 인증서를 제공합니다.

5.1.1. 단방향 TLS 구성

다음 절차에서는 단방향 **TLS**에 대해 지정된 수락자를 구성하는 방법을 보여줍니다.

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.
2. 지정된 허용자의 경우 **sslEnabled** 키를 추가하고 값을 **true** 로 설정합니다. 또한 **keyStorePath** 및 **keyStorePassword** 키를 추가합니다. 브로커 키 저장소에 해당하는 값을 설정합니다. 예를 들면 다음과 같습니다.

```
<acceptor name="artemis">tcp://0.0.0.0:61616?
sslEnabled=true;keyStorePath=../etc/broker.keystore;keyStorePassword=1234!</accept
or>
```

5.1.2. 양방향 TLS 구성

다음 절차에서는 양방향 **TLS**를 구성하는 방법을 보여줍니다.

사전 요구 사항

- 단방향 **TLS**에 대해 지정된 어셉터를 이미 구성해야 합니다. 자세한 내용은 **5.1.1절. "단방향 TLS 구성"**의 내용을 참조하십시오.

프로세스

1. *<broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.*

2. *이전에 단방향 TLS용으로 구성한 승인자의 경우 needClientAuth 키를 추가합니다. 값을 true 로 설정합니다. 예를 들면 다음과 같습니다.*

```
<acceptor name="artemis">tcp://0.0.0.0:61616?
sslEnabled=true;keyStorePath=../etc/broker.keystore;keyStorePassword=1234!;needCl
ientAuth=true</acceptor>
```

3. *이전 단계의 구성은 클라이언트의 인증서가 신뢰할 수 있는 공급자가 서명한다고 가정합니다. 클라이언트의 인증서가 신뢰할 수 있는 공급자(예: 자체 서명)에 서명 되지 않은 경우 브로커는 클라이언트의 인증서를 신뢰 저장소로 가져와야 합니다. 이 경우 trustStorePath 및 trustStorePassword 키를 추가합니다. 브로커 신뢰 저장소에 해당하는 값을 설정합니다. 예를 들면 다음과 같습니다.*

```
<acceptor name="artemis">tcp://0.0.0.0:61616?
sslEnabled=true;keyStorePath=../etc/broker.keystore;keyStorePassword=1234!;needCl
ientAuth=true;trustStorePath=../etc/client.truststore;trustStorePassword=5678!</accept
or>
```

참고

AMQ Broker는 여러 프로토콜을 지원하며 각 프로토콜과 플랫폼은 TLS 매개변수를 지정하는 방법이 다릅니다. 그러나 Core Protocol(a bridge)을 사용하는 클라이언트의 경우 TLS 매개변수는 브로커의 수락자처럼 커넥터 URL에 구성됩니다.

자체 서명된 인증서가 JVM(Java Virtual Machine) 신뢰 저장소에 신뢰할 수 있는 인증서로 나열된 경우 JVM은 인증서의 만료 날짜를 확인하지 않습니다. 프로덕션 환경에서는 인증 기관에서 서명한 인증서를 사용하는 것이 좋습니다.

5.1.3. TLS 구성 옵션

다음 표에는 사용 가능한 모든 TLS 구성 옵션이 나와 있습니다.

표 5.1. TLS 구성 옵션

옵션	참고

옵션	참고
sslEnabled	연결에 SSL이 활성화되었는지 여부를 지정합니다. TLS를 활성화하려면 **true** 로 설정해야 합니다. 기본값은 **false**입니다.
keyStorePath	**수락자에서 사용되는 경우:** 브로커 인증서를 보유한 브로커의 TLS 키 저장소 경로(자체 서명 또는 기관에서 서명)합니다. **커넥터에 사용되는 경우:** 클라이언트 인증서가 있는 클라이언트의 TLS 키 저장소로 이동합니다. 이는 양방향 TLS를 사용하는 경우에만 커넥터와 관련이 있습니다. 브로커에서 이 값을 구성할 수 있지만 클라이언트가 다운로드하여 사용합니다. 클라이언트가 브로커에 설정된 것과 다른 경로를 사용해야 하는 경우 표준 **javax.net.ssl.keyStore** 시스템 속성 또는 AMQ 특정 **org.apache.activemq.ssl.keyStore** 시스템 속성을 사용하여 브로커 설정을 덮어쓸 수 있습니다. AMQ별 시스템 속성은 클라이언트의 다른 구성 요소가 이미 표준 Java 시스템 속성을 사용하고 있는 경우 유용합니다.
keyStorePassword	**수락자에 사용되는 경우:** 브로커의 키 저장소에 대한 암호입니다. **커넥터에 사용되는 경우:** 클라이언트의 키 저장소에 대한 암호입니다. 이는 양방향 TLS를 사용하는 경우에만 커넥터와 관련이 있습니다. 브로커에서 이 값을 구성할 수 있지만 클라이언트가 다운로드하여 사용합니다. 클라이언트가 브로커에 설정된 것과 다른 암호를 사용해야 하는 경우 표준 **javax.net.ssl.keyStorePassword** 시스템 속성 또는 AMQ 특정 **org.apache.activemq.ssl.keyStorePassword** 시스템 속성을 사용하여 브로커 설정을 덮어쓸 수 있습니다. AMQ별 시스템 속성은 클라이언트의 다른 구성 요소가 이미 표준 Java 시스템 속성을 사용하고 있는 경우 유용합니다.

옵션	참고
trustStorePath	**수락자에서 사용되는 경우:** 브로커가 신뢰하는 모든 클라이언트의 키를 보유한 브로커의 TLS 신뢰 저장소 경로입니다. 이는 양방향 TLS를 사용하는 경우에만 어셉터와 관련이 있습니다. **커넥터에 사용되는 경우:** 클라이언트가 신뢰하는 모든 브로커의 공개 키를 보유한 클라이언트의 TLS 신뢰 저장소 경로입니다. 브로커에서 이 값을 구성할 수 있지만 클라이언트가 다운로드하여 사용합니다. 클라이언트가 서버에 설정된 것과 다른 경로를 사용해야 하는 경우 표준 **javax.net.ssl.trustStore** 시스템 속성 또는 AMQ 특정 **org.apache.activemq.ssl.trustStore** 시스템 속성을 사용하여 서버 측 설정을 재정의할 수 있습니다. AMQ별 시스템 속성은 클라이언트의 다른 구성 요소가 이미 표준 Java 시스템 속성을 사용하고 있는 경우 유용합니다.
trustStorePassword	**수락자에서 사용되는 경우:** 브로커의 신뢰 저장소에 대한 암호입니다. 이는 양방향 TLS를 사용하는 경우에만 어셉터와 관련이 있습니다. **커넥터에 사용되는 경우:** 클라이언트의 신뢰 저장소에 대한 암호입니다. 브로커에서 이 값을 구성할 수 있지만 클라이언트가 다운로드하여 사용합니다. 클라이언트가 브로커에 설정된 것과 다른 암호를 사용해야 하는 경우 표준 **javax.net.ssl.trustStorePassword** 시스템 속성 또는 AMQ 특정 **org.apache.activemq.ssl.trustStorePassword** 시스템 속성을 사용하여 브로커 설정을 재정의할 수 있습니다. AMQ별 시스템 속성은 클라이언트의 다른 구성 요소가 이미 표준 Java 시스템 속성을 사용하고 있는 경우 유용합니다.

옵션	참고
enabledCipherSuites	허용자 또는 커넥터 모두에 TLS 통신에 사용되는 쉼표로 구분된 암호화 제품군 목록입니다. 클라이언트 애플리케이션에서 지원하는 가장 안전한 암호화 제품군을 지정합니다. 브로커와 클라이언트에 공통되는 암호화 제품군의 쉼표로 구분된 목록을 지정하거나 암호화 모음을 지정하지 않는 경우 브로커와 클라이언트는 사용할 암호화 제품군을 서로 협상합니다. 지정할 암호화 모음을 모르는 경우 먼저 디버그 모드에서 실행 중인 클라이언트와 broker-client 연결을 설정하여 브로커와 클라이언트 모두에 공통된 암호화 제품군을 확인할 수 있습니다. 그런 다음 브로커에서 **enabledCipherSuites** 를 구성합니다. 사용 가능한 암호화 제품군은 브로커 및 클라이언트에서 사용하는 TLS 프로토콜 버전에 따라 다릅니다. 브로커를 업그레이드한 후 기본 TLS 프로토콜 버전이 변경되면 브로커와 클라이언트가 공통 암호화 제품군을 사용할 수 있도록 이전 TLS 프로토콜 버전을 선택해야 할 수 있습니다. 자세한 내용은 **enabledProtocols** 를 참조하십시오.
enabledProtocols	어셉터 또는 커넥터에 사용하든, 이는 TLS 통신에 사용되는 쉼표로 구분된 프로토콜 목록입니다. TLS 프로토콜 버전을 지정하지 않으면 브로커는 JVM의 기본 버전을 사용합니다. 브로커가 JVM에 기본 TLS 프로토콜 버전을 사용하고 브로커를 업그레이드한 후 해당 버전이 변경되면 브로커 및 클라이언트에서 사용하는 TLS 프로토콜 버전이 호환되지 않을 수 있습니다. 이후 TLS 프로토콜 버전을 사용하는 것이 좋지만, **enabledProtocols** 에서 이전 버전을 지정하여 최신 TLS 프로토콜 버전을 지원하지 않는 클라이언트와 상호 운용할 수 있습니다.
needClientAuth	이 속성은 승인자만 사용할 수 있습니다. 두 방향 TLS가 필요하다는 어셉터에 연결하는 클라이언트에 지시합니다. 유효한 값은 **true** 또는 **false**입니다. 기본값은 **false**입니다.

5.2. 클라이언트 인증

5.2.1. 클라이언트 인증 방법

브로커에서 클라이언트 인증을 구성하려면 다음 방법을 사용할 수 있습니다.

사용자 이름 및 암호 기반 인증

다음 옵션 중 하나를 사용하여 사용자 인증 정보를 직접 검증합니다.

- 브로커에 로컬로 저장된 속성 파일 세트에 대해 인증 정보를 확인합니다. 브로커에 대한 제한된 액세스를 허용하고 로그인 모듈을 결합하여 보다 복잡한 사용 사례를 지원하는 게스트 계정을 구성할 수도 있습니다.

- 중앙 **X.500** 디렉터리 서버에 저장된 사용자 데이터에 대해 클라이언트 자격 증명을 확인하도록 **LDAP(Lightweight Directory Access Protocol)** 로그인 모듈을 구성합니다.

인증서 기반 인증

브로커와 클라이언트가 상호 인증을 위해 인증서를 제공하도록 양방향 **TLS(Transport Layer Security)**를 구성합니다. 관리자는 승인된 클라이언트 사용자 및 역할을 정의하는 속성 파일도 구성해야 합니다. 이러한 속성 파일은 브로커에 저장됩니다.

Kerberos 기반 인증

SASL(Simple Authentication and Security Layer) 프레임워크의 **GSSAPI** 메커니즘을 사용하여 클라이언트의 **Kerberos** 보안 자격 증명을 인증하도록 브로커를 구성합니다.

다음 섹션에서는 사용자 및 암호 기반 인증과 인증서 기반 인증을 모두 구성하는 방법을 설명합니다.

추가 리소스

- **LDAP** 및 **Kerberos**의 완전한 인증 및 권한 부여 워크플로에 대한 자세한 내용은 다음을 참조하십시오.
  - 5.4절. “인증 및 권한 부여에 LDAP 사용”
  - 5.5절. “인증 및 권한 부여에 Kerberos 사용”

5.2.2. 속성 파일을 기반으로 사용자 및 암호 인증 구성

AMQ Broker는 주소를 기반으로 큐에 보안을 적용하기 위한 유연한 역할 기반 보안 모델을 지원합니다. 대기열은 일대일(**point-to-point** 메시징의 경우) 또는 **many-to-one(publish-subscribe** 메시징의 경우)에 바인딩됩니다. 메시지가 주소로 전송되면 브로커는 해당 주소에 바인딩된 대기열 세트를 조회하고 메시지를 해당 대기열 세트로 라우팅합니다.

기본 사용자 및 암호 인증이 필요한 경우 **PropertiesLoginModule** 을 사용하여 정의합니다. 이 로그인 모듈은 브로커에 로컬로 저장된 다음 구성 파일에 대해 사용자 인증 정보를 확인합니다.

artemis-users.properties

사용자 및 해당 암호를 정의하는 데 사용됩니다.

artemis-roles.properties

역할을 정의하고 해당 역할에 사용자를 할당하는 데 사용됩니다.

login.config

사용자 및 암호 인증 및 게스트 액세스에 대한 로그인 모듈을 구성하는 데 사용됩니다.

artemis-users.properties 파일은 보안을 위해 해시된 암호를 포함할 수 있습니다.

다음 섹션에서는 구성 방법을 보여줍니다.

- 기본 사용자 및 암호 인증
- 게스트 액세스를 포함하는 사용자 및 암호 인증

5.2.2.1. 기본 사용자 및 암호 인증 구성

다음 절차에서는 기본 사용자 및 암호 인증을 구성하는 방법을 보여줍니다.

프로세스

1. &lt ;broker_instance_dir> /etc/login.config 구성 파일을 엽니다. 기본적으로 새로운 **AMQ Broker 7.12** 인스턴스의 이 파일에는 다음 행이 포함되어 있습니다.

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient
    debug=false
    reload=true
    org.apache.activemq.jaas.properties.user="artemis-users.properties"
    org.apache.activemq.jaas.properties.role="artemis-roles.properties";
     };
```

ActiveMQ

구성의 별칭입니다.

org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule

구현 클래스입니다.

충분합니다.

***PropertiesLoginModule** 에 필요한 성공 수준을 지정하는 플래그입니다. 설정할 수 있는 값은 다음과 같습니다.*

- *필수: 로그인 모듈이 성공하려면 필요합니다. 인증은 성공 또는 실패와 관계없이 지정된 별칭 아래에 구성된 로그인 모듈 목록을 계속 진행합니다.*
- *필수 조건: 로그인 모듈이 성공하려면 필수입니다. 오류가 발생하면 애플리케이션에 제어를 즉시 반환합니다. 인증은 지정된 별칭 아래에 구성된 로그인 모듈 목록을 진행하지 않습니다.*
- ***sufficient:** 로그인 모듈이 성공할 필요가 없습니다. 성공하면 제어가 애플리케이션으로 돌아가고 인증이 더 진행되지 않습니다. 실패하면 인증 시도에서 지정된 별칭 아래에 구성된 로그인 모듈 목록을 진행합니다.*
- *선택 사항: 로그인 모듈이 성공할 필요가 없습니다. 인증은 성공 또는 실패 여부에 관계없이 지정된 별칭 아래에 구성된 로그인 모듈 목록을 계속 종료합니다.*

org.apache.activemq.jaas.properties.user

로그인 모듈 구현에 대한 사용자 및 암호 집합을 정의하는 속성 파일을 지정합니다.

org.apache.activemq.jaas.properties.role

사용자를 로그인 모듈 구현을 위해 정의된 역할에 매핑하는 속성 파일을 지정합니다.

2. ***&lt ;broker_instance_dir> /etc/artemis-users.properties** 구성 파일을 엽니다.*

3. *사용자를 추가하고 사용자에게 암호를 할당합니다. 예를 들면 다음과 같습니다.*

```
user1=secret
user2=access
user3=myPassword
```

4. &lt ;broker_instance_dir> /etc/artemis-roles.properties 구성 파일을 엽니다.

5. 이전에 Artemis -users.properties 파일에 추가한 사용자에게 역할 이름을 할당합니다. 예를 들면 다음과 같습니다.

```
admin=user1,user2
developer=user3
```

6. &lt ;broker_instance_dir> /etc/bootstrap.xml 구성 파일을 엽니다.

7. 필요한 경우 다음과 같이 보안 도메인 별칭(이 인스턴스에서 activemq)을 파일에 추가합니다.

```
<jaas-security domain="activemq"/>
```

5.2.2.2. 게스트 액세스 구성

로그인 인증 정보가 없거나 인증 정보가 실패한 사용자의 경우 게스트 계정을 사용하여 브로커에 대한 제한된 액세스 권한을 부여할 수 있습니다.

명령줄 스위치 --allow-anonymous (- require-login)를 사용하여 게스트 액세스가 활성화된 브로커 인스턴스를 생성할 수 있습니다.

다음 절차에서는 게스트 액세스를 구성하는 방법을 보여줍니다.

사전 요구 사항

- 이 절차에서는 이미 기본 사용자 및 암호 인증을 구성한 것으로 가정합니다. 자세한 내용은 5.2.2.1절. “기본 사용자 및 암호 인증 구성” 에서 참조하십시오.

프로세스

1. 이전에 기본 사용자 및 암호 인증을 위해 구성한 < broker_instance_dir> /etc/login.config

구성 파일을 엽니다.

2. 이전에 추가한 속성 로그인 모듈 구성 후 게스트 로그인 모듈 구성을 추가합니다. 예를 들면 다음과 같습니다.

```
activemq {
 org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient
    debug=true
    org.apache.activemq.jaas.properties.user="artemis-users.properties"
    org.apache.activemq.jaas.properties.role="artemis-roles.properties";

 org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule sufficient
    debug=true
    org.apache.activemq.jaas.guest.user="guest"
    org.apache.activemq.jaas.guest.role="restricted";
};
```

org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule

구현 클래스입니다.

org.apache.activemq.jaas.guest.user

익명 사용자에게 할당된 사용자 이름입니다.

org.apache.activemq.jaas.guest.role

익명 사용자에게 할당된 역할입니다.

사용자가 인증 정보를 제공하는 경우 이전 구성에 따라 사용자 및 암호 인증 모듈이 활성화됩니다. 사용자가 인증 정보를 제공하지 않거나 제공된 인증 정보가 올바르지 않은 경우 게스트 인증이 활성화됩니다.

5.2.2.2.1. 게스트 액세스 예

다음 예제에서는 인증 정보가 없는 사용자만 게스트로 로그인되는 사용 사례에 대한 게스트 액세스 구성을 보여줍니다. 이 예제에서 로그인 모듈의 순서가 이전 구성 프로시저와 비교되어 있는지 확인합니다. 또한 **properties** 로그인 모듈에 연결된 플래그가 필수 로 변경됩니다.

```
activemq {
   org.apache.activemq.artemis.spi.core.security.jaas.GuestLoginModule sufficient
      debug=true
     credentialsInvalidate=true
     org.apache.activemq.jaas.guest.user="guest"
     org.apache.activemq.jaas.guest.role="guests";
```

```
   org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule requisite
      debug=true
      org.apache.activemq.jaas.properties.user="artemis-users.properties"
      org.apache.activemq.jaas.properties.role="artemis-roles.properties";
};
```

이전 구성에 따라 로그인 인증 정보가 제공되지 않으면 게스트 인증 모듈이 활성화됩니다.

이 사용 사례의 경우 게스트 로그인 모듈 구성에서 **credentialsInvalidate** 옵션을 **true** 로 설정해야 합니다.

인증 정보가 제공되면 **properties** 로그인 모듈이 활성화됩니다. 자격 증명은 유효해야 합니다.

추가 리소스

- **JAAS(Java Authentication and Authorization Service)**에 대한 자세한 내용은 **Java** 벤더의 설명서를 참조하십시오. 예를 들어 **login.config** 구성에 대한 **Oracle** 튜토리얼은 **Oracle Java** 문서의 **JAAS** 로그인 구성 파일을 참조하십시오.
- 클라이언트 인증 정보를 검증하기 위해 **LDAP** 로그인 모듈을 구성하는 방법을 알아보려면 **5.4.1**절. “클라이언트 인증을 위해 **LDAP** 구성” 를 참조하십시오.
- 구성 파일에서 암호를 암호화하는 방법에 대한 자세한 내용은 **5.9.2**절. “구성 파일에서 암호 암호화” 을 참조하십시오.

5.2.3. 인증서 기반 인증 구성

JAAS(Java Authentication and Authorization Service) 인증서 로그인 모듈은 **TLS(Transport Layer Security)**를 사용하는 클라이언트에 대한 인증 및 권한 부여를 처리합니다. 이 모듈을 사용하려면 양방향 **TLS(Transport Layer Security)**를 사용하고 클라이언트가 자체 인증서로 구성해야 합니다. 인증은 **JAAS** 인증서 로그인 모듈이 아닌 **TLS** 핸드셰이크 중에 수행됩니다.

인증서 로그인 모듈의 역할은 다음과 같습니다.

- 허용 가능한 사용자 집합을 제한합니다. 관련 속성 파일에 명시적으로 나열된 **DN(**사용자 고유 이름 **)**만 인증할 수 있습니다.

- 수신된 사용자 **ID**와 그룹 목록을 연결합니다. 이는 승인을 용이하게 합니다.

- 들어오는 클라이언트 인증서가 있어야 합니다(기본적으로 클라이언트 인증서의 존재를 선택 사항으로 처리하도록 **TLS** 계층이 구성됨).

인증서 로그인 모듈은 인증서 **DN** 컬렉션을 플랫 텍스트 파일 쌍에 저장합니다. 파일은 사용자 이름과 그룹 **ID** 목록을 각 **DN**과 연결합니다.

인증서 로그인 모듈은 **org.apache.activemq.artemis.spi.core.security.jaas.CryostatFileCertificateLoginModule** 클래스에 의해 구현됩니다.

5.2.3.1. 인증서 기반 인증을 사용하도록 브로커 구성

다음 절차에서는 인증서 기반 인증을 사용하도록 브로커를 구성하는 방법을 보여줍니다.

사전 요구 사항

- 양방향 **TLS(Transport Layer Security)**를 사용하도록 브로커를 구성해야 합니다. 자세한 내용은 **5.1.2**절. “양방향 **TLS** 구성”의 내용을 참조하십시오.

프로세스

1. 이전에 브로커 키 저장소로 가져온 사용자 인증서에서 **DN(Subject Distinguished Names)**을 가져옵니다.

   a. 키 저장소 파일의 인증서를 임시 파일로 내보냅니다. 예를 들면 다음과 같습니다.

   ```
   keytool -export -file <file_name> -alias broker-localhost -keystore broker.ks -storepass <password>
   ```

   b. 내보낸 인증서의 내용을 출력합니다.

   ```
   keytool -printcert -file <file_name>
   ```

   출력은 다음과 유사합니다.

```
Owner: CN=localhost, OU=broker, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Issuer: CN=localhost, OU=broker, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Serial number: 4537c82e
Valid from: Thu Oct 19 19:47:10 BST 2006 until: Wed Jan 17 18:47:10 GMT 2007
Certificate fingerprints:
        MD5:  3F:6C:0C:89:A8:80:29:CC:F5:2D:DA:5C:D7:3F:AB:37
        SHA1: F0:79:0D:04:38:5A:46:CE:86:E1:8A:20:1F:7B:AB:3A:46:E4:34:5C
```

Owner 항목은 주체 **DN**입니다. 주체 **DN**을 입력하는 데 사용되는 형식은 플랫폼에 따라 다릅니다. 위의 문자열은 다음과 같이 표시될 수도 있습니다.

```
Owner: `CN=localhost,\ OU=broker,\ O=Unknown,\ L=Unknown,\ ST=Unknown,\
C=Unknown`
```

2. 인증서 기반 인증을 구성합니다.

a. **&lt ;broker_instance_dir> /etc/login.config** 구성 파일을 엽니다. 인증서 로그인 모듈을 추가하고 사용자 및 역할 속성 파일을 참조합니다. 예를 들면 다음과 같습니다.

```
activemq {

org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule
        debug=true
        org.apache.activemq.jaas.textfiledn.user="artemis-users.properties"
        org.apache.activemq.jaas.textfiledn.role="artemis-roles.properties";
};
```

org.apache.activemq.artemis.spi.core.security.jaas.TextFileCertificateLoginModule

구현 클래스입니다.

org.apache.activemq.jaas.textfiledn.user

로그인 모듈 구현에 대한 사용자 및 암호 집합을 정의하는 속성 파일을 지정합니다.

org.apache.activemq.jaas.textfiledn.role

사용자를 로그인 모듈 구현을 위해 정의된 역할에 매핑하는 속성 파일을 지정합니다.

b. **&lt ;broker_instance_dir> /etc/artemis-users.properties** 구성 파일을 엽니다. 사용자와 해당 **DN**은 이 파일에 정의되어 있습니다. 예를 들면 다음과 같습니다.

```
system=CN=system,O=Progress,C=US
user=CN=humble user,O=Progress,C=US
guest=CN=anon,O=Progress,C=DE
```

이전 구성을 기반으로 **system** *이라는 사용자는* **CN=system,O=Progress,C=US Subject DN**에 *매핑됩니다.*

c.

&lt ;broker_instance_dir> /etc/artemis-roles.properties *구성 파일을 엽니다. 사용 가능한 역할 및 해당 역할을 보유한 사용자는 이 파일에 정의됩니다. 예를 들면 다음과 같습니다.*

```
admins=system
users=system,user
guests=guest
```

이전 구성에서 **users** *역할의 경우 여러 사용자를 쉼표로 구분된 목록으로 나열합니다.*

d.

다음과 같이 보안 도메인 별칭(이 인스턴스에서 **activemq)**이 **bootstrap.xml** *에서 참조되었는지 확인합니다.*

```
<jaas-security domain="activemq"/>
```

5.2.3.2. AMQP 클라이언트에 대한 인증서 기반 인증 구성

***SASL(Simple Authentication and Security Layer) EXTERNAL** 메커니즘 구성 매개 변수를 사용하여 브로커에 연결할 때 인증서 기반 인증을 위해* **AQMP** *클라이언트를 구성합니다.*

브로커는 모든 인증서를 인증하는 것과 동일한 방식으로 **AMQP** *클라이언트의* **TLS(Transport Layer Security)/SSL(Secure Sockets Layer)** *인증서를 인증합니다.*

1.

브로커는 클라이언트의 **TLS/SSL** *인증서를 읽고 인증서 제목에서* **ID**를 *가져옵니다.*

2.

인증서 주체는 인증서 로그인 모듈을 통해 브로커 **ID**에 *매핑됩니다. 그런 다음 브로커는 해당 역할을 기반으로 사용자에게 권한을 부여합니다.*

다음 절차에서는 **AMQP** *클라이언트에 대한 인증서 기반 인증을 구성하는 방법을 보여줍니다.* **AMQP** *클라이언트가 인증서 기반 인증을 사용할 수 있도록 하려면 클라이언트가 브로커에 연결하는 데 사용하는*

URI에 구성 매개변수를 추가해야 합니다.

사전 요구 사항

- 다음을 구성해야 합니다.
  - **Two-way TLS.** 자세한 내용은 **5.1.2절. "양방향 TLS 구성"**의 내용을 참조하십시오.
  - 인증서 기반 인증을 사용하는 브로커입니다. 자세한 내용은 **5.2.3.1절. "인증서 기반 인증을 사용하도록 브로커 구성"**의 내용을 참조하십시오.

프로세스

1. 편집할 **URI**가 포함된 리소스를 엽니다.

```
amqps://localhost:5500
```

2. 연결에 **TSL/SSL**을 활성화하려면 **sslEnabled=true** 매개변수를 추가합니다.

```
amqps://localhost:5500?sslEnabled=true
```

3. 클라이언트 신뢰 저장소 및 키 저장소와 관련된 매개변수를 추가하여 브로커와 **TSL/SSL** 인증서를 교환할 수 있습니다.

```
amqps://localhost:5500?
sslEnabled=true&trustStorePath=<trust_store_path>&trustStorePassword=<trust_store_
password>&keyStorePath=<key_store_path>&keyStorePassword=<key_store_password>
```

4. **saslMechanisms=EXTERNAL** 매개변수를 추가하여 브로커가 **TSL/SSL** 인증서에 있는 **ID**를 사용하여 클라이언트를 인증하도록 요청합니다.

```
amqps://localhost:5500?
sslEnabled=true&trustStorePath=<trust_store_path>&trustStorePassword=<trust_store_
password>&keyStorePath=<key_store_path>&keyStorePassword=<key_store_password>
&saslMechanisms=EXTERNAL
```

추가 리소스

-

AMQ Broker의 인증서 기반 인증에 대한 자세한 내용은 5.2.3.1절. “인증서 기반 인증을 사용하도록 브로커 구성” 을 참조하십시오.

- AMQP 클라이언트 구성에 대한 자세한 내용은 Red Hat Customer Portal 로 이동하여 클라이언트와 관련된 제품 설명서를 참조하십시오.

5.3. 클라이언트 승인

5.3.1. 클라이언트 권한 부여 방법

주소 및 큐 생성 및 삭제, 메시지 전송 및 소비와 같은 브로커에서 작업을 수행하도록 클라이언트에 권한을 부여하려면 다음 방법을 사용할 수 있습니다.

사용자 및 역할 기반 권한 부여

인증된 사용자 및 역할에 대해 브로커 보안 설정을 구성합니다.

클라이언트 권한을 부여하도록 LDAP 구성

인증 및 권한 부여를 모두 처리하도록 LDAP(Lightweight Directory Access Protocol) 로그인 모듈을 구성합니다. LDAP 로그인 모듈은 중앙 X.500 디렉터리 서버에 저장된 사용자 데이터에 대해 들어오는 인증 정보를 확인하고 사용자 역할을 기반으로 권한을 설정합니다.

클라이언트 권한을 부여하도록 Kerberos 구성

Kerberos 인증 및 권한 부여 서비스 (JAAS) Krb5LoginModule 로그인 모듈을 구성하여 인증 정보를 AMQ Broker 역할에 매핑하는 PropertiesLoginModule 또는 LDAPLoginModule 로그인 모듈에 전달합니다.

5.3.2. 사용자 및 역할 기반 권한 부여 구성

5.3.2.1. 권한 설정

권한은 broker.xml 구성 파일의 < security-setting > 요소를 통해 큐(주소 기반)에 대해 정의됩니다. 구성 파일의 < security- settings> 요소에서 < security-settings >의 여러 인스턴스를 정의할 수 있습니다. 정확한 주소 일치를 지정하거나 숫자 기호(#) 및 별표(*) 와일드카드 문자를 사용하여 일치하는 와일드카드를 정의할 수 있습니다.

주소와 일치하는 대기열 세트에 다른 권한을 부여할 수 있습니다. 이러한 권한은 다음 표에 표시되어 있습니다.

표 5.2. 대기열 권한

사용자가...할 수 있도록 허용	이 매개변수 사용...
주소 생성	**createAddress**
주소 삭제	**deleteAddress**
일치하는 주소 아래에 Cryostat 큐 생성	**createDurableQueue**
일치하는 주소 아래의 Cryostat 큐 삭제	**deleteDurableQueue**
일치하는 주소에 정렬할 수 없는 큐를 생성	**createNonDurableQueue**
일치하는 주소에서 확인할 수 없는 큐 삭제	**deleteNonDurableQueue**
일치하는 주소로 메시지 보내기	**전송**
일치하는 주소에 바인딩된 큐의 메시지 사용	**consume**
관리 주소로 관리 메시지를 전송하여 관리 작업 호출	**관리**
일치하는 주소에 바인딩된 큐 검색	**검색**
관리 작업의 하위 집합에 대한 읽기 전용 액세스 권한	**view**
보기 권한으로 액세스 권한이 부여되지 않은 모든 작업인 변경 관리 작업에 액세스합니다.	**edit**

각 권한에 대해 권한이 부여된 역할 목록을 지정합니다. 지정된 사용자에게 역할 중 하나가 있는 경우 해당 주소 집합에 대한 권한이 부여됩니다.

다음 섹션에서는 권한에 대한 몇 가지 구성 예제를 보여줍니다.

5.3.2.1.1. 단일 주소에 대한 메시지 프로덕션 구성

다음 절차에서는 단일 주소에 대한 메시지 프로덕션 권한을 구성하는 방법을 보여줍니다.

프로세스

1. *&lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.*
2. *< security- settings> 요소에 단일 <security-setting > 요소를 추가합니다. 일치 키의 경우*

주소를 지정합니다. 예를 들면 다음과 같습니다.

```
<security-settings>
   <security-setting match="my.destination">
      <permission type="send" roles="producer"/>
   </security-setting>
</security-settings>
```

이전 구성을 기반으로 생산자 역할의 멤버는 **my.destination** 주소에 대한 전송 권한이 있습니다.

5.3.2.1.2. 단일 주소에 대한 메시지 사용 구성

다음 절차에서는 단일 주소에 대한 메시지 사용 권한을 구성하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **< security- settings>** 요소에 단일 **<security-setting >** 요소를 추가합니다. 일치 키의 경우 주소를 지정합니다. 예를 들면 다음과 같습니다.

```
<security-settings>
   <security-setting match="my.destination">
      <permission type="consume" roles="consumer"/>
   </security-setting>
</security-settings>
```

이전 구성을 기반으로 소비자 역할의 멤버는 **my.destination** 주소에 대한 사용 권한을 갖습니다.

5.3.2.1.3. 모든 주소에 대한 전체 액세스 구성

다음 절차에서는 모든 주소 및 연결된 큐에 대한 전체 액세스를 구성하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2.

< security- settings> 요소에 단일 <security-setting > 요소를 추가합니다. 일치 키의 경우 모든 주소에 대한 액세스를 구성하려면 숫자 기호(#) 와일드카드 문자를 지정합니다. 예를 들면 다음과 같습니다.

```
<security-settings>
   <security-setting match="#">
      <permission type="createDurableQueue" roles="guest"/>
      <permission type="deleteDurableQueue" roles="guest"/>
      <permission type="createNonDurableQueue" roles="guest"/>
      <permission type="deleteNonDurableQueue" roles="guest"/>
      <permission type="createAddress" roles="guest"/>
      <permission type="deleteAddress" roles="guest"/>
      <permission type="send" roles="guest"/>
      <permission type="browse" roles="guest"/>
      <permission type="consume" roles="guest"/>
      <permission type="manage" roles="guest"/>
   </security-setting>
</security-settings>
```

이전 구성을 기반으로 모든 큐에 대한 guest 역할의 멤버에게 모든 권한이 부여됩니다. 이 기능은 모든 사용자에게 게스트 역할을 할당하도록 익명 인증이 구성된 개발 시나리오에서 유용할 수 있습니다.

추가 리소스

- *더 복잡한 사용 사례 구성에 대한 자세한 내용은 5.3.2.1.4절. "여러 보안 설정 구성" 을 참조하십시오.*

5.3.2.1.4. 여러 보안 설정 구성

다음 예제 절차에서는 일치하는 주소 집합에 대해 여러 보안 설정을 개별적으로 구성하는 방법을 보여줍니다. 이는 모든 주소에 대한 전체 액세스 권한을 부여하는 방법을 보여주는 이 섹션의 이전 예제와 대조됩니다.

1. *&lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.*

2. *< security- settings> 요소에 단일 <security-setting > 요소를 추가합니다. 일치 키의 경우 일치하는 주소 집합에 설정을 적용하려면 숫자 기호(#) 와일드카드 문자를 포함합니다. 예를 들면 다음과 같습니다.*

```
<security-setting match="globalqueues.europe.#">
  <permission type="createDurableQueue" roles="admin"/>
  <permission type="deleteDurableQueue" roles="admin"/>
  <permission type="createNonDurableQueue" roles="admin, guest, europe-users"/>
```

```
    <permission type="deleteNonDurableQueue" roles="admin, guest, europe-users"/>
    <permission type="send" roles="admin, europe-users"/>
    <permission type="consume" roles="admin, europe-users"/>
  </security-setting>
```

match=globalqueues.europe.#

숫자 기호(#) 와일드카드 문자는 브로커에서 "모든 순서의 단어"로 해석됩니다. 마침표(.)로 구분됩니다. 이 예제에서 보안 설정은 **globalqueues.europe** 문자열로 시작하는 모든 주소에 적용됩니다.

permission type="createDurableQueue"

admin 역할이 있는 사용자만 **globalqueues.europe** 문자열로 시작하는 주소에 바인딩된 **Cryostat** 큐를 생성하거나 삭제할 수 있습니다.

permission type="createNonDurableQueue"

역할 **admin,guest** 또는 **europe-users** 가 있는 모든 사용자는 **globalqueues.europe** 문자열로 시작하는 주소에 바인딩된 임시 큐를 만들거나 삭제할 수 있습니다.

권한 유형="send"

역할 **admin** 또는 **europe-users** 를 사용하는 모든 사용자는 문자열 **globalqueues.europe**로 시작하는 주소에 바인딩된 큐로 메시지를 보낼 수 있습니다.

permission type="consume"

역할 **admin** 또는 **europe-users** 를 사용하는 모든 사용자는 문자열 **globalqueues.europe**로 시작하는 주소에 바인딩된 대기열의 메시지를 사용할 수 있습니다.

3. (선택 사항) 보다 좁은 주소 세트에 다른 보안 설정을 적용하려면 다른 **< security-setting>** 요소를 추가합니다. 일치 키의 경우 더 구체적인 텍스트 문자열을 지정합니다. 예를 들면 다음과 같습니다.

```
<security-setting match="globalqueues.europe.orders.#">
  <permission type="send" roles="europe-users"/>
  <permission type="consume" roles="europe-users"/>
</security-setting>
```

두 번째 **security-setting** 요소에서 **globalqueues.europe.orders.# match**는 첫 번째 **security-setting** 요소에 지정된 **globalqueues.europe.#** 보다 더 구체적입니다. **globalqueues.europe.orders.#** 과 일치하는 모든 주소의 경우 **createDurableQueue,deleteDurableQueue,createNonDurableQueue,deleteNonDurableQueue** 에서는 파일의 첫 번째 **security-setting** 요소에서 상속 되지 않습니다. 예를 들어 **globalqueues.europe.orders.plastics** 주소의 경우 존재하는 유일한 권한은 **europe-users** 역할에 대해 전송 및 소비 됩니다.

따라서 한 **security-setting** 블록에 지정된 권한이 다른 블록에 상속되지 않으므로 해당 권한을 지정하지 않고 보다 구체적인 **security-setting** 블록에서 권한을 효과적으로 거부할 수 있습니다.

5.3.2.1.5. 사용자로 대기열 구성

큐가 자동으로 생성되면 연결 클라이언트의 사용자 이름이 큐에 할당됩니다. 이 사용자 이름은 큐의 메타데이터로 포함됩니다. 이름은 **Cryostat** 및 **AMQ Broker** 관리 콘솔에 의해 노출됩니다.

다음 절차에서는 브로커 구성에 수동으로 정의한 큐에 사용자 이름을 추가하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 지정된 큐의 경우 사용자 키를 추가합니다. 값을 할당합니다. 예를 들면 다음과 같습니다.

```
<address name="ExampleQueue">
  <anycast>
    <queue name="ExampleQueue">
      <user>admin</user>
    </queue>
  </anycast>
</address>
```

이전 구성을 기반으로 **admin** 사용자는 **ExampleQueue** 큐에 할당됩니다.

참고

- 큐에서 사용자를 구성해도 해당 큐에 대한 보안 의미 체계는 변경되지 않으며 해당 큐의 메타데이터에만 사용됩니다.

- 사용자 간 매핑과 보안 관리자 라는 구성 요소에서 처리하는 역할 간의 매핑입니다. 보안 관리자는 브로커에 저장된 속성 파일에서 사용자 자격 증명을 읽습니다. 기본적으로 **AMQ Broker**는 **org.apache.activemq.artemis.spi.core.security.ActiveMQJAASSecurityManager** 보안 관리자를 사용합니다. 이 기본 보안 관리자는 **JAAS** 및 **Red Hat JBoss EAP(JBoss Enterprise Application Platform)** 보안과의 통합을 제공합니다.

  사용자 정의 보안 관리자를 사용하는 방법에 대한 자세한 내용은 5.6.2절. “사용자 정의 보안 관리자 지정” 을 참조하십시오.

5.3.2.2. 역할 기반 액세스 제어 구성

RBAC(역할 기반 액세스 제어)는 속성 및 메서드의 액세스를 제한하는 데 사용됩니다. **Cryostat**는 **AMQ Broker**에서 관리 작업을 지원하기 위해 관리 **API**를 노출하는 방법입니다.

다음 방법 중 하나를 사용하여 **Cryostat**에 대한 액세스를 제한할 수 있습니다.

- 기본 방법인 **management.xml** 파일에서 권한 부여 요소를 구성합니다.

- **broker.xml** 파일에서 보안 설정을 구성합니다.

management.xml 파일을 업데이트할 때와 달리 **broker.xml** 파일의 보안 설정을 변경한 후에는 브로커를 다시 시작할 필요가 없습니다.

5.3.2.2.1. management.xml 파일에서 역할 기반 액세스 제어 구성.

관리 작업에 대한 역할 기반 액세스 제어를 구성하는 기본 방법은 **management.xml** 파일에서 권한 부여 요소를 구성하는 것입니다.

다음 예제 절차에서는 역할을 특정 **Cryostat** 및 해당 속성 및 메서드에 매핑하는 방법을 보여줍니다.

사전 요구 사항

- 사용자 및 역할을 정의했습니다. 자세한 내용은 5.2.2.1절. “기본 사용자 및 암호 인증 구성”의 내용을 참조하십시오.

프로세스

1. &lt ;broker_instance_dir> /etc/management.xml 구성 파일을 엽니다.

2. role-access 요소를 검색하고 구성을 편집합니다. 예를 들면 다음과 같습니다.

```
<role-access>
  <match domain="org.apache.activemq.artemis">
    <access method="list*" roles="view,update,amq"/>
    <access method="get*" roles="view,update,amq"/>
    <access method="is*" roles="view,update,amq"/>
    <access method="set*" roles="update,amq"/>
    <access method="*" roles="amq"/>
  </match>
</role-access>
```

- 이 경우 도메인 이름이 org.apache.activemq.apache 인 모든 Cryostat 속성에 일치가 적용됩니다.

- 일치하는 Cryostat 속성에 대한 보기,업데이트 또는 amq 역할의 액세스는 목록*, get* , set*, is* 및 * 액세스 방법 중 에 대한 액세스 권한을 제어합니다. method="*"(wildcard) 구문은 구성에 나열되지 않은 다른 모든 방법을 지정하는 catch-all 방법으로 사용됩니다. 구성의 각 액세스 메서드는 Cryostat 메서드 호출로 변환됩니다.

- 호출된 Cryostat 메서드는 구성에 나열된 메서드와 일치합니다. 예를 들어 org.apache.activemq.artemis 도메인을 사용하여 Cryostat에서 listMessages 라는 메서드를 호출하면 브로커는 목록 메서드 구성에 정의된 역할에 다시 액세스합니다.

- 전체 메서드 이름을 사용하여 액세스를 구성할 수도 있습니다. 예를 들면 다음과 같습니다.

```
<access method="listMessages" roles="view,update,amq"/>
```

3. 브로커를 시작하거나 다시 시작합니다.

- Linux에서: & lt;broker_instance_dir> /bin/artemis 실행

- *Windows에서: & lt;broker_instance_dir> \bin\artemis-service.exe start*

Cryostat 속성과 일치하는 키 특성을 추가하여 도메인 내의 특정 Cryostat를 일치시킬 수도 있습니다.

5.3.2.2.1.1. 역할 기반 액세스 예

이 섹션에서는 역할 기반 액세스 제어를 적용하는 다음 예를 보여줍니다.

- *도메인의 모든 큐에 역할을 매핑.*
- *도메인의 특정 큐에 역할을 매핑.*
- *지정된 접두사를 포함하는 모든 큐 이름에 역할을 매핑 합니다.*
- *다양한 역할을 다른 대기열 세트에 매핑합니다.*

다음 예제에서는 키 특성을 사용하여 지정된 도메인의 모든 큐에 역할을 매핑하는 방법을 보여줍니다.

```
<match domain="org.apache.activemq.artemis" key="subcomponent=queues">
  <access method="list*" roles="view,update,amq"/>
  <access method="get*" roles="view,update,amq"/>
  <access method="is*" roles="view,update,amq"/>
  <access method="set*" roles="update,amq"/>
  <access method="*" roles="amq"/>
</match>
```

다음 예제에서는 키 특성을 사용하여 이름이 지정된 특정 큐에 역할을 매핑하는 방법을 보여줍니다. 이 예제에서 명명된 큐는 예제Queue 입니다.

```
<match domain="org.apache.activemq.artemis" key="queue=exampleQueue">
  <access method="list*" roles="view,update,amq"/>
  <access method="get*" roles="view,update,amq"/>
  <access method="is*" roles="view,update,amq"/>
```

```
    <access method="set*" roles="update,amq"/>
    <access method="*" roles="amq"/>
  </match>
```

다음 예제에서는 지정된 접두사가 포함된 모든 큐에 역할을 매핑하는 방법을 보여줍니다. 이 예에서는 별표(*) 와일드카드 연산자가 접두사로 시작하는 모든 큐 이름과 일치하도록 사용됩니다.

```
<match domain="org.apache.activemq.artemis" key="queue=example*">
  <access method="list*" roles="view,update,amq"/>
  <access method="get*" roles="view,update,amq"/>
  <access method="is*" roles="view,update,amq"/>
  <access method="set*" roles="update,amq"/>
  <access method="*" roles="amq"/>
</match>
```

동일한 특성 집합(예: 다양한 대기열 세트)에 대해 역할을 다르게 매핑해야 할 수 있습니다. 이 경우 구성 파일에 일치하는 요소를 여러 개 포함할 수 있습니다. 그러나 동일한 도메인에 여러 일치 항목을 가질 수 있습니다.

예를 들어 다음과 같이 구성된 두 개의 **<match >** 요소를 고려해 보십시오.

```
<match domain="org.apache.activemq.artemis" key="queue=example*">
```

및

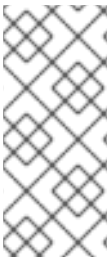
```
<match domain="org.apache.activemq.artemis" key="queue=example.sub*">
```

이 구성을 기반으로 **org.apache.activemq.artemis** 도메인에 있는 **example.sub.queue** 라는 큐는 두 와일드카드 키 표현식과 일치합니다. 따라서 브로커는 큐에 매핑할 역할 집합, 첫 번째 일치 요소에 지정된 역할 또는 두 번째 일치 요소에 지정된 역할을 결정하는 우선순위 지정 체계가 필요합니다.

동일한 도메인에 일치하는 항목이 여러 개인 경우 브로커는 역할을 매핑할 때 다음 우선순위 지정 체계를 사용합니다.

- 와일드카드 일치보다 정확한 일치 항목의 우선순위가 지정됨
- 더 긴 와일드카드 일치는 더 짧은 와일드카드 일치보다 우선 순위가 지정됩니다.

*이 예에서 긴 와일드카드 표현식은 **example.sub.queue** 의 큐 이름과 가장 근접하게 일치하므로 브로커는 두 번째 **< match >** 요소에 구성된 **role-mapping**을 적용합니다.*



참고

***default-access** 요소는 **role-access** 또는 허용 목록 구성을 사용하여 처리되지 않는 모든 메서드 호출의 **catch-all** 요소입니다. **default-access** 및 **role-access** 요소에는 동일한 **match** 요소 의미 체계가 있습니다.*

5.3.2.2.1.2. allowlist 요소 구성

*허용 목록은 사용자 인증이 필요하지 않은 사전 승인 도메인 또는 **Cryostats** 집합입니다. 인증을 바이패스해야 하는 도메인 목록 또는 도메인 목록 또는 둘 다 제공할 수 있습니다. 예를 들어 **allowlist** 를 사용하여 **AMQ Broker** 관리 콘솔에 필요한 모든 **Cryostat**를 지정할 수 있습니다.*

*다음 예제 절차에서는 **allowlist** 요소를 구성하는 방법을 보여줍니다.*

프로세스

1. *&lt ;broker_instance_dir> /etc/management.xml 구성 파일을 엽니다.*

2. *allowlist 요소를 검색하고 구성을 편집합니다.*

```
<allowlist>
  <entry domain="hawtio"/>
</allowlist>
```

*이 예에서 도메인 **hawtio** 가 있는 모든 **Cryostat**는 인증 없이 액세스할 수 있습니다. 또한 **< entry domain="hawtio" key="type=*"/ >** 형식의 와일드카드 항목을 사용할 수도 있습니다.*

3. *브로커를 시작하거나 다시 시작합니다.*

   - *Linux에서: & lt;broker_instance_dir> /bin/artemis 실행*
   - *Windows에서: & lt;broker_instance_dir> \bin\artemis-service.exe start*

5.3.2.2.2. broker.xml 파일에서 역할 기반 액세스 제어 구성

management.xml 파일 대신 broker.xml 파일에서 관리 작업에 대한 역할 기반 액세스 제어를 구성할 수 있습니다. broker.xml 파일에서 권한을 업데이트하려면 브로커를 다시 시작할 필요가 없습니다.

broker.xml 파일에서 관리 작업에 대한 보기 또는 편집 권한을 부여할 수 있습니다. 보기 또는 편집 권한이 있는 역할에 사용할 수 있는 특정 관리 작업은 사전 정의된 정규식으로 제어됩니다. 정규식과 일치하는 모든 작업에는 뷰 권한과 기타 모든 작업에 대한 편집 권한이 있는 역할로 액세스할 수 있습니다.

사전 요구 사항

- 사용자 및 역할을 정의했습니다. 자세한 내용은 5.2.2.1절. “기본 사용자 및 암호 인증 구성”의 내용을 참조하십시오.

프로세스

1. 브로커가 이 파일에서 기본 RBAC 구성을 사용하지 못하도록 management.xml 파일에서 권한 부여 요소 구성을 삭제합니다.
   a. < broker_instance_dir>/etc/management.xml 파일을 편집합니다.
   b. 파일에서 권한 부여 요소 구성을 삭제합니다.
   c. < broker_instance_dir>/etc/management.xml 파일을 저장합니다.
2. 브로커 JVM에 환경 변수를 추가하여 broker.xml 파일에서 RBAC 구성을 사용하도록 브로커를 구성합니다.
   a. &lt ;broker_instance_dir>/etc/artemis.profile 파일을 엽니다.
   b. Java 시스템 인수의 JAVA_ARGS 목록에 다음 인수를 추가합니다.

```
-Djavax.management.builder.initial=org.apache.activemq.artemis.core.server.management.ArtemisRbacMBeanServerBuilder
```

c.

Artemis .profile 파일을 저장합니다.

3.

&lt ;broker_instance_dir> /etc/broker.xml 파일을 열어 관리 작업에 대한 RBAC를 구성합니다.

4.

security-settings 요소를 검색하고 관리 작업에 대한 security-setting 요소를 추가합니다.

관리 작업의 일치 주소 형식은 다음과 같습니다.

<_management-rbac-prefix_>.<_resource type_>.<_resource name_>.<_operation_>

management-rbac-prefix 매개변수의 기본값은 mops 입니다.

다음 예제 RBAC 구성에서 일치 주소의 숫자 기호(#)는 admin 역할 보기를 부여하고 모든 Cryostat에 대한 권한을 편집합니다.

```
<security-settings>
 ..
 <security-setting match="mops.#">
  <permission type="view" roles="admin"/>
  <permission type="edit" roles="admin"/>
 </security-setting>
 ..
</security-setting>
```

5.

&lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 저장합니다.

관리 작업을 위한 역할 기반 액세스 제어의 기타 예

다음 예제에서는 manager 역할 보기 와 activemq.management 주소에 대한 편집 권한을 부여합니다. 작업 위치의 별표(*)는 모든 작업에 대한 액세스 권한을 부여합니다.

```
<security-setting match="mops.address.activemq.management.*">
  <permission type="view" roles="manager"/>
</security-setting>
```

다음 예제에는 브로커를 사용하여 지정된 작업을 수행할 수 있는 모든 사용자 권한을 거부하는 빈 roles 목록이 있습니다.

```
<security-setting match="mops.broker.forceFailover">
  <permission type="edit" roles=""/>
</security-setting>
```

5.3.2.3. 리소스 제한 설정

경우에 따라 특정 사용자가 권한 부여 및 인증과 관련된 일반 보안 설정을 초과하여 수행할 수 있는 특정 제한을 설정하는 것이 유용합니다.

5.3.2.3.1. 연결 및 큐 제한 구성

다음 예제 절차에서는 사용자가 생성할 수 있는 연결 및 대기열 수를 제한하는 방법을 보여줍니다.

1. **<broker_instance_dir>/etc/broker.xml** 구성 파일을 엽니다.

2. **resource-limit-settings** 요소를 추가합니다. **max-connections** 및 **max-queues** 의 값을 지정합니다. 예를 들면 다음과 같습니다.

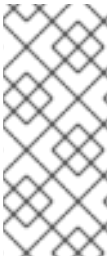
```
<resource-limit-settings>
  <resource-limit-setting match="myUser">
    <max-connections>5</max-connections>
    <max-queues>3</max-queues>
  </resource-limit-setting>
</resource-limit-settings>
```

max-connections

일치하는 사용자가 브로커에서 생성할 수 있는 세션 수를 정의합니다. 기본값은 **-1** 이며 이는 제한이 없음을 의미합니다. 세션 수를 제한하려면 **AMQ Core Protocol JMS** 클라이언트에서 브로커에 대한 각 연결이 두 개의 세션을 생성하는 경우를 고려하십시오.

max-queues

일치하는 사용자가 생성할 수 있는 대기열 수를 정의합니다. 기본값은 **-1** 이며 이는 제한이 없음을 의미합니다.



참고

브로커 구성의 **address-setting** 요소에 지정할 수 있는 일치 문자열과 달리 **resource-limit-settings** 에서 지정하는 일치 문자열은 와일드카드 구문을 사용할 수 없습니다. 대신 **match** 문자열은 리소스 제한 설정을 적용할 특정 사용자를 정의합니다.

5.4. 인증 및 권한 부여에 LDAP 사용

LDAP 로그인 모듈은 중앙 **X.500** 디렉터리 서버에 저장된 사용자 데이터에 대해 들어오는 자격 증명을 확인하여 인증 및 권한 부여를 활성화합니다. **org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule** 에 의해 구현됩니다.

5.4.1. 클라이언트 인증을 위해 LDAP 구성

다음 예제 절차에서는 **LDAP**를 사용하여 클라이언트를 인증하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **security-settings** 요소 내에서 **security-setting** 요소를 추가하여 권한을 구성합니다. 예를 들면 다음과 같습니다.

```
<security-settings>
   <security-setting match="#">
      <permission type="createDurableQueue" roles="user"/>
      <permission type="deleteDurableQueue" roles="user"/>
      <permission type="createNonDurableQueue" roles="user"/>
      <permission type="deleteNonDurableQueue" roles="user"/>
      <permission type="send" roles="user"/>
      <permission type="consume" roles="user"/>
   </security-setting>
</security-settings>
```

이전 구성에서는 모든 큐에 대한 특정 권한을 사용자 역할의 멤버에게 할당합니다.

3. **< broker_instance_dir>/etc/login.config** 파일을 엽니다.

4. 사용 중인 디렉터리 서비스를 기반으로 **LDAP** 로그인 모듈을 구성합니다.

a.

Microsoft Active Directory 디렉터리 서비스를 사용하는 경우 이 예제와 유사한 구성을 추가합니다.

```
activemq {
 org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule required
    debug=true
    initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
    connectionURL="LDAP://localhost:389"

connectionUsername="CN=Administrator,CN=Users,OU=System,DC=example,DC=
com"
    connectionPassword=redhat.123
    connectionProtocol=s
    connectionTimeout="5000"
    authentication=simple
    userBase="dc=example,dc=com"
    userSearchMatching="(CN={0})"
    userSearchSubtree=true
    readTimeout="5000"
    roleBase="dc=example,dc=com"
    roleName=cn
    roleSearchMatching="(member={0})"
    roleSearchSubtree=true
    ;
};
```

참고

Microsoft Active Directory를 사용하고 connectionUsername 속성에 지정해야 하는 값에 공백(예: OU=System Accounts)이 포함된 경우 값을 큰 따옴표("") 쌍으로 묶어야 하며 쌍의 이중 인용을 이스케이프하려면 백슬래시(\)를 사용해야 합니다. 예를 들어 connectionUsername="CN=Administrator,CN=Users,OU=\"System Accounts\",DC=example,DC=com" 입니다.

b.

ApacheDS 디렉터리 서비스를 사용하는 경우 다음 예제와 유사한 구성을 추가합니다.

```
activemq {
 org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule required
    debug=true
    initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
    connectionURL="ldap://localhost:10389"
    connectionUsername="uid=admin,ou=system"
    connectionPassword=secret
    connectionProtocol=s
    connectionTimeout=5000
    authentication=simple
    userBase="dc=example,dc=com"
    userSearchMatching="(uid={0})"
```

```
    userSearchSubtree=true
    userRoleName=
    readTimeout=5000
    roleBase="dc=example,dc=com"
    roleName=cn
    roleSearchMatching="(member={0})"
    roleSearchSubtree=true
    ;
};
```

debug

디버깅을 켜십시오 (true) 또는 off (false). 기본값은 false입니다.

initialContextFactory

항상 com.sun.jndi.ldap.LdapCtxFactory로 설정해야 합니다.

connectionURL

LDAP URL을 사용하는 디렉터리 서버의 위치 __<ldap://Host:Port>. 선택 옵션으로 슬래시(/)를 추가한 다음 디렉터리 트리의 특정 노드의 DN을 추가하여 이 URL을 지정할 수 있습니다. Apache DS의 기본 포트는 10389 이고 Microsoft AD의 기본값은 389 입니다.

connectionUsername

디렉터리 서버에 대한 연결을 여는 사용자의 고유 이름(DN)입니다. 예: uid=admin,ou=system. 일반적으로 디렉터리 서버는 연결을 열기 위해 클라이언트가 사용자 이름/암호 자격 증명을 제공해야 합니다.

connectionPassword

connectionUsername 에서 DN과 일치하는 암호입니다. 디렉터리 서버에서 디렉터리 정보 트리 (DIT)에서 암호는 일반적으로 해당 디렉터리 항목에 userPassword 속성으로 저장됩니다.

connectionProtocol

모든 값이 지원되지만 효과적으로 사용되지 않습니다. 기본값이 없으므로 이 옵션을 명시적으로 설정해야 합니다.

connectionTimeout

브로커가 디렉터리 서버에 연결하는 데 사용할 수 있는 최대 시간(밀리초)을 지정합니다. 브로커가 이 시간 내에 디렉터리에 연결할 수 없는 경우 연결 시도가 중단됩니다. 이 속성에 0 이하 값을 지정하면 기본 TCP 프로토콜의 시간 초과 값이 대신 사용됩니다. 값을 지정하지 않으면 브로커는 연결을 설정하거나 기본 네트워크가 시간 초과될 때까지 무기한 대기합니다.

연결에 대해 연결 풀링이 요청되면 이 속성은 브로커가 최대 풀 크기에 이미 도달하여 풀의 모든 연결이 사용 중인 경우 브로커가 연결을 기다리는 최대 시간을 지정합니다. 값이 **0** 이하인 경우 브로커는 연결을 사용할 수 있을 때까지 무기한 대기합니다. 그렇지 않으면 브로커가 최대 대기 시간에 도달했을 때 연결 시도를 중단합니다.

인증

LDAP 서버에 바인딩할 때 사용되는 인증 방법을 지정합니다. 이 매개변수는 단순(사용자 이름 및 암호가 필요한) 또는 **none** (명명 액세스 허용)으로 설정할 수 있습니다.

userBase

DIT의 특정 하위 트리를 선택하여 사용자 항목을 검색합니다. 하위 트리는 **DN**에 의해 지정되며, 하위 트리의 기본 노드를 지정합니다. 예를 들어 이 옵션을 **ou=User,ou=ActiveMQ,ou=system** 로 설정하면 사용자 항목 검색이 **ou=User,ou=ActiveMQ,ou=system** 노드 아래에 있는 하위 트리로 제한됩니다.

userSearchMatching

userBase 에서 선택한 하위 트리에 적용되는 **LDAP** 검색 필터를 지정합니다. 자세한 내용은 아래 **5.4.1.1절. "일치하는 매개변수 검색"** 섹션을 참조하십시오.

userSearchSubtree

userBase 에서 지정한 노드를 기준으로 사용자 항목의 검색 깊이를 지정합니다. 이 옵션은 부울입니다. **false** 값을 지정하면 검색에서 **userBase** 노드의 하위 항목 중 하나와 일치하려고 합니다(**Java x.naming.directory.SearchControls.ONELEVEL_SCOPE**). **true** 값을 지정하면 검색이 **userBase** 노드의 하위 트리에 속하는 모든 항목과 일치하려고 합니다(**Java x.naming.directory.SearchControls.SUBTREE_SCOPE**).

userRoleName

사용자의 역할 이름 목록이 포함된 사용자 항목의 속성 이름입니다. 역할 이름은 브로커의 권한 부여 플러그인에서 그룹 이름으로 해석됩니다. 이 옵션을 생략하면 사용자 항목에서 역할 이름을 추출하지 않습니다.

readTimeout

브로커가 디렉터리 서버에서 **LDAP** 요청으로 응답을 수신하기 위해 기다릴 수 있는 최대 시간(밀리초)을 지정합니다. 이 시점에서 브로커가 디렉터리 서버에서 응답을 수신하지 못하면 브로커가 요청을 중단합니다. 값을 **0** 이하로 지정하거나 값을 지정하지 않으면 브로커는 디렉터리 서버의 응답이 **LDAP** 요청까지 무기한 대기합니다.

roleBase

역할 데이터가 디렉터리 서버에 직접 저장되는 경우 **userRoleName** 옵션을 지정하는 대신 역할 옵션(**roleBase,roleSearchMatching,roleSearchSubtree, roleName**)의 조합을 사용할 수 있습니다. 이 옵션은 역할/그룹 항목을 검색할 **DIT**의 특정 하위 트리를 선택합니다. 하위 트리는 **DN**에 의해 지정되며, 하위 트리의 기본 노드를 지정합니다. 예를

*들어 이 옵션을 **ou=Group,ou=ActiveMQ,ou=system** 로 설정하면 역할/그룹 항목 검색은 **ou=Group,ou=ActiveMQ,ou=system** 노드 아래에 있는 하위 트리로 제한됩니다.*

roleName

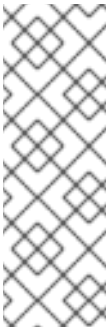
*역할/그룹의 이름이 포함된 역할 항목의 특성 유형(예: **C, O, OU** 등). 이 옵션을 생략하면 역할 검색 기능이 효과적으로 비활성화됩니다.*

roleSearchMatching

***roleBase** 에서 선택한 하위 트리에 적용되는 **LDAP** 검색 필터를 지정합니다. 자세한 내용은 아래 **5.4.1.1절. “일치하는 매개변수 검색”** 섹션을 참조하십시오.*

roleSearchSubtree

***roleBase** 에서 지정한 노드를 기준으로 역할 항목의 검색 깊이를 지정합니다. **false** (기본값)로 설정하면 검색이 **roleBase** 노드의 하위 항목 중 하나와 일치하려고 하는 경우(**Java x.naming.directory.SearchControls.ONELEVEL_SCOPE**). **true** 인 경우 **roleBase** 노드의 하위 트리에 속하는 모든 항목을 일치시킵니다(**javax.naming.directory.SearchControls.SUBTREE_SCOPE**에 매핑).*



참고

***Apache DS**는 **DN** 경로의 **OID** 부분을 사용합니다. **Microsoft Active Directory**는 **CN** 부분을 사용합니다. 예를 들어 **Apache DS**의 **oid=testuser,dc=example,dc=com** 과 같은 **DN** 경로를 사용할 수 있지만 **Microsoft Active Directory**에서 **cn=testuser,dc=example,dc=com** 과 같은 **DN** 경로를 사용할 수 있습니다.*

5.

브로커(서비스 또는 프로세스)를 시작하거나 다시 시작합니다.

5.4.1.1. 일치하는 매개변수 검색

userSearchMatching

***LDAP** 검색 작업에 전달하기 전에 **java.text.MessageFormat** 클래스에서 구현한 대로 이 구성 매개변수에 제공된 문자열 대체가 적용됩니다.*

*즉, 특수 문자열 **{0}** 은 들어오는 클라이언트 인증 정보에서 추출된 대로 사용자 이름으로 대체됩니다. 대체 후 문자열은 **LDAP** 검색 필터로 해석됩니다. 구문은 **IETF** 표준 **RFC 2254**에 의해 정의됩니다.*

*예를 들어 이 옵션이 **(uid={0})** 로 설정되고 수신된 사용자 이름이 **jdoe** 인 경우 문자열 대체 후 검색 필터가 **(uid=jdoe)** 됩니다.*

*결과 검색 필터가 사용자 기반인 **ou=User,ou=ActiveMQ,ou=system** 에서 선택한 하위 트리에 적용되는 경우 **uid=jdoe,ou=User,ou=ActiveMQ,ou=system.***

roleSearchMatching

*이 명령은 두 개의 대체 문자열을 지원하는 경우를 제외하고 **userSearchMatching** 옵션과 유사한 방식으로 작동합니다.*

*대체 문자열 **{0}** 은 일치하는 사용자 항목의 전체 **DN**(즉, 사용자 검색 결과)을 대체합니다. 예를 들어 사용자 **jdoe** 의 경우 대체된 문자열은 **uid=jdoe,ou=User,ou=ActiveMQ,ou=system** 일 수 있습니다.*

*대체 문자열 **{1}** 은/는 수신된 사용자 이름을 대체합니다. 예를 들면 **jdoe** 입니다.*

*이 옵션이 **(member=uid={1})** 로 설정되고 수신된 사용자 이름이 **jdoe** 인 경우 검색 필터는 문자열 대체(**ApacheDS** 검색 필터 구문 사용) 후 **(member=uid=jdoe)** 가 됩니다.*

*결과 검색 필터가 역할 기반 (**ou=Group,ou=ActiveMQ,ou=ActiveMQ,ou=system**)에서 선택한 하위 트리에 적용되는 경우 **member** 속성이 **uid=jdoe** (멤버 특성의 값이 **DN**)인 모든 역할 항목과 일치합니다.*

*이 옵션은 기본값이 없으므로 역할 검색을 비활성화하더라도 항상 설정해야 합니다. **OpenLDAP** 를 사용하는 경우 검색 필터의 구문은 **(member:=uid=jdoe)** 입니다.*

추가 리소스

- *검색 필터 구문에 대한 간략한 소개는 **Oracle JNDI** 튜토리얼을 참조하십시오.*

5.4.2. LDAP 권한 부여 구성

LegacyLDAPSecuritySettingPlugin 보안 설정 플러그인은 **LDAPAuthorizationMap** 및 **cached LDAPAuthorizationMap** 을 통해 **AMQ 6**에서 이전에 처리하는 보안 정보를 읽고 이 정보를 가능한 경우 해당 **AMQ 7** 보안 설정으로 변환합니다.

AMQ 6 및 **AMQ 7** 브로커의 보안 구현은 정확히 일치하지 않습니다. 따라서 플러그인은 두 버전 간에 일부 변환을 수행하여 거의 동일한 기능을 수행합니다.

다음 예제에서는 플러그인을 구성하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **security-settings** 요소 내에서 **security-setting-plugin** 요소를 추가합니다. 예를 들면 다음과 같습니다.

```
<security-settings>
   <security-setting-plugin class-
name="org.apache.activemq.artemis.core.server.impl.LegacyLDAPSecuritySettingPlugin">
      <setting name="initialContextFactory" value="com.sun.jndi.ldap.LdapCtxFactory"/>
      <setting name="connectionURL"
value="ldap://localhost:1024"/>`ou=destinations,o=ActiveMQ,ou=system`
      <setting name="connectionUsername" value="uid=admin,ou=system"/>
      <setting name="connectionPassword" value="secret"/>
      <setting name="connectionProtocol" value="s"/>
      <setting name="authentication" value="simple"/>
   </security-setting-plugin>
</security-settings>
```

class-name

구현은 **org.apache.activemq.artemis.core.server.impl.LegacyLDAPSecuritySettingPlugin** 입니다.

initialContextFactory

LDAP 연결에 사용되는 초기 컨텍스트 팩토리입니다. 항상 **com.sun.jndi.ldap.LdapCtxFactory** (즉, 기본값)로 설정해야 합니다.

connectionURL

LDAP URL <ldap://Host:Port>을 사용하여 디렉터리 서버의 위치를 지정합니다. 선택적으로 디렉토리 트리에 있는 특정 노드의 DN(고유 이름) 뒤에 슬래시(/)를 추가하여 이 URL을 지정할 수 있습니다. 예: ldap://ldapserver:10389/ou=system. 기본값은 ldap://localhost:1024 입니다.

connectionUsername

디렉터리 서버에 대한 연결을 여는 사용자의 DN입니다. 예: uid=admin,ou=system. 일반적으로 디렉터리 서버는 연결을 열기 위해 클라이언트가 사용자 이름/암호 자격 증명을 제공해야 합니다.

connectionPassword

connectionUsername 에서 DN과 일치하는 암호입니다. 디렉터리 서버에서 디렉터리 정보 트리 (DIT)에서 암호는 일반적으로 해당 디렉터리 항목에 userPassword 속성으로 저장됩니다.

connectionProtocol

현재 사용되지 않습니다. 나중에 이 옵션을 사용하면 디렉터리 서버에 연결하기 위해 SSL(Secure Socket Layer)을 선택할 수 있습니다. 기본값이 없으므로 이 옵션을 명시적으로 설정해야 합니다.

인증

LDAP 서버에 바인딩할 때 사용되는 인증 방법을 지정합니다. 이 매개변수의 유효한 값은 simple (사용자 이름 및 암호) 또는 none (익명)입니다. 기본값은 simple 입니다.

참고

SASL(Simple Authentication and Security Layer) 인증은 지원되지 않습니다.

이전 구성 예에 표시되지 않는 기타 설정은 다음과 같습니다.

destinationBase

자식이 모든 대상에 대한 권한을 제공하는 노드의 DN을 지정합니다. 이 경우 DN은 리터럴 값입니다(즉, 속성 값에서 문자열 대체가 수행되지 않음). 예를 들어 이 속성의 일반적인 값은 ou=destinations,o=ActiveMQ,ou=system 이며, 기본값은 ou=destinations,o=ActiveMQ,ou=system 입니다.

filter

모든 종류의 대상에 대한 권한을 검색할 때 사용되는 LDAP 검색 필터를 지정합니다. 검색 필터는

큐 또는 주제 노드의 하위 항목 또는 하위 항목 중 하나와 일치하려고 합니다. 기본값은 (cn=) 입니다.*

roleAttribute

값이 역할의 DN인 필터 와 일치하는 노드의 특성을 지정합니다. 기본값은 uniqueMember 입니다.

adminPermissionValue

관리자 권한과 일치하는 값을 지정합니다. 기본값은 admin 입니다.

readPermissionValue

읽기 권한과 일치하는 값을 지정합니다. 기본값은 read 입니다.

writePermissionValue

쓰기 권한과 일치하는 값을 지정합니다. 기본값은 write 입니다.

enableListener

LDAP 서버에서 만든 업데이트를 자동으로 수신하고 브로커의 권한 부여 구성을 실시간으로 업데이트하는 리스너를 활성화할지 여부를 지정합니다. 기본값은 true입니다.

mapAdminToManage

레거시(즉, AMQ 6) 관리자 권한을 AMQ 7 관리 권한에 매핑할지 여부를 지정합니다. 아래 표에서 매핑 의미 체계에 대한 세부 정보를 참조하십시오. 기본값은 false입니다.

LDAP에 정의된 큐 또는 주제의 이름은 보안 설정의 "match" 역할을 하며, 권한 값은 AMQ 6 유형에서 AMQ 7 유형으로 매핑되며 역할은 그대로 매핑됩니다. LDAP에 정의된 큐 또는 주제의 이름은 보안 설정의 일치 역할을 하므로 보안 설정이 JMS 대상에 예상대로 적용되지 않을 수 있습니다. AMQ 7은 필요에 따라 JMS 대상 앞에 "jms.queue" 또는 "jms.topic"을 접두사로 지정하기 때문입니다.

AMQ 6에는 읽기,쓰기, 관리자 등 세 가지 권한 유형이 있습니다. 이러한 권한 유형은 ActiveMQ 웹 사이트; 보안에 설명되어 있습니다.

AMQ 7에는 다음과 같은 권한 유형이 있습니다.

- ***createAddress***
- ***deleteAddress***

- *createDurableQueue*
- *deleteDurableQueue*
- *createNonDurableQueue*
- *deleteNonDurableQueue*
- *전송*
- *consume*
- *관리*
- *검색*

*이 표는 보안 설정 플러그인이 **AMQ 6** 권한 유형을 **AMQ 7** 권한 유형에 매핑하는 방법을 보여줍니다.*

*표 **5.3. AMQ 6** 권한 유형을 **AMQ 7**에 매핑*

AMQ 6 권한 유형	AMQ 7 권한 유형
읽기	사용, 검색
쓰기	전송
admin	createAddress, deleteAddress, createDurableQueue, deleteDurableQueue, createNonDurableQueue, deleteNonDurableQueue, manage (**mapAdminToManage** is set to **true**)

*아래 설명된 대로 플러그인이 **AMQ 6** 및 **AMQ 7** 권한 유형 간에 일부 변환을 수행하여 동일성을 달성하는 몇 가지 경우가 있습니다.*

- *AMQ 6에는 유사한 권한 유형이 없기 때문에 매핑에는 기본적으로 AMQ 7 관리 권한 유형이 포함되지 않습니다. 그러나 mapAdminToManage 이 true 로 설정된 경우 플러그인은 AMQ 6 관리자 권한을 AMQ 7 관리 권한에 매핑합니다.*

- *AMQ 6의 관리자 권한 유형은 대상이 없는 경우 브로커가 자동으로 대상을 생성하고 사용자에게 메시지를 보내는지 결정합니다. AMQ 7은 사용자가 대상에 메시지를 보낼 수 있는 권한이 있는 경우 대상의 자동 생성을 허용합니다. 따라서 플러그인은 기본적으로 위에 표시된 AMQ 7 권한에 레거시 관리자 권한을 매핑합니다. mapAdminToManage 이 true 로 설정된 경우 플러그인은 AMQ 6 관리자 권한을 AMQ 7 관리 권한에 매핑합니다.*

allowQueueAdminOnRead

기존 읽기 권한을 createDurableQueue, createNonDurableQueue, deleteDurableQueue 권한에 매핑할지 여부에 관계없이 JMS 클라이언트가 관리자 권한이 없어도 사용 가능한 서브스크립션을 생성할 수 있도록 합니다. AMQ 6에서 허용되었습니다. 기본값은 false입니다.

이 표는 allowQueueAdminOnRead 가 true 인 경우 AMQ 6 권한 유형을 AMQ 7 권한 유형에 매핑하는 방법을 보여줍니다.

표 5.4. allowQueueAdminOnRead 가 true인 경우 AMQ 6 권한 유형을 AMQ 7에 매핑

AMQ 6 권한 유형	AMQ 7 권한 유형
읽기	consume, browse, createDurableQueue, createNonDurableQueue, deleteDurableQueue
쓰기	전송
admin	createAddress, deleteAddress, deleteNonDurableQueue, manage (**mapAdminToManage** 가 **true**로 설정된 경우)

5.4.3. login.config 파일에서 암호 암호화

조직에서 LDAP로 데이터를 자주 안전하게 저장하므로 login.config 파일에 브로커가 조직의 LDAP 서버와 통신하는 데 필요한 구성이 포함될 수 있습니다. 이 구성 파일에는 일반적으로 LDAP 서버에 로그인하는 암호가 포함되어 있으므로 이 암호를 암호화해야 합니다.

사전 요구 사항

- *5.4.2절. "LDAP 권한 부여 구성" 에 설명된 대로 필요한 속성을 추가하도록 login.config 파일을 수정했는지 확인합니다.*

프로세스

다음 절차에서는 **< broker_instance_dir> /etc/login.config** 파일에 있는 **connectionPassword** 매개변수 값을 마스킹하는 방법을 보여줍니다.

1. 명령 프롬프트에서 **mask** 유틸리티를 사용하여 암호를 암호화합니다.

```
$ <broker_instance_dir>/bin/artemis mask <password>
```

```
result: 3a34fd21b82bf2a822fa49a8d8fa115d
```

2. **< broker_instance_dir>/etc/login.config** 파일을 엽니다. **connectionPassword** 매개변수를 찾습니다.

```
connectionPassword = <password>
```

3. 일반 텍스트 암호를 암호화된 값으로 바꿉니다.

```
connectionPassword = 3a34fd21b82bf2a822fa49a8d8fa115d
```

4. 암호화된 값을 **"ENC()"** 식별자로 래핑합니다.

```
connectionPassword = "ENC(3a34fd21b82bf2a822fa49a8d8fa115d)"
```

이제 **login.config** 파일에 마스크된 암호가 포함됩니다. 암호가 **"ENC()"** 식별자로 래핑되므로 **AMQ Broker**는 사용하기 전에 암호를 해독합니다.

추가 리소스

- **AMQ Broker**에 포함된 구성 파일에 대한 자세한 내용은 **AMQ Broker** 구성 파일 및 위치를 참조하십시오.

5.4.4. 외부 역할 매핑

LDAP와 같은 외부 인증 공급자의 역할을 브로커가 내부적으로 사용하는 역할에 매핑할 수 있습니다.

외부 역할을 매핑하려면 **broker.xml** 구성 파일의 **security-settings** 요소에 **role-mapping** 항목을 생

성합니다. 예를 들면 다음과 같습니다.

```
<security-settings>
...
   <role-mapping from="cn=admins,ou=Group,ou=ActiveMQ,ou=system" to="my-admin-role"/>
   <role-mapping from="cn=users,ou=Group,ou=ActiveMQ,ou=system" to="my-user-role"/>
</security-settings>
```

참고

- 역할 매핑은 추가 기능입니다. 즉, 사용자는 원래 역할과 새로 할당된 역할을 유지합니다.
- 역할 매핑은 대기열 액세스를 승인하는 역할에만 영향을 미치며 웹 콘솔 액세스를 활성화하는 방법을 제공하지 않습니다.

5.5. 인증 및 권한 부여에 KERBEROS 사용

AMQP 프로토콜을 사용하여 메시지를 보내고 받을 때 클라이언트는 **SASL(Simple Authentication and Security Layer)** 프레임워크에서 **GSSAPI** 메커니즘을 사용하여 **AMQ Broker**가 인증하는 **Kerberos** 보안 자격 증명을 보낼 수 있습니다. **Kerberos** 자격 증명은 인증된 사용자를 **LDAP** 디렉터리 또는 텍스트 기반 속성 파일에 구성된 할당된 역할에 매핑하여 권한 부여에 사용할 수도 있습니다.

TLS(Transport Layer Security)와 함께 **SASL**을 사용하여 메시징 애플리케이션을 보호할 수 있습니다. **SASL**은 사용자 인증을 제공하며 **TLS**는 데이터 무결성을 제공합니다.

중요

- AMQ Broker가 Kerberos 인증 정보를 인증하고 인증하려면 Kerberos 인프라를 배포하고 구성해야 합니다. Kerberos 배포에 대한 자세한 내용은 운영 체제 설명서를 참조하십시오.
  - RHEL 7의 경우 Kerberos 사용을 참조하십시오.
  - Windows의 경우 Kerberos 인증 개요 를 참조하십시오.
- Oracle 또는 IBM JDK 사용자는 JCE(Java Cryptography Extension)를 설치해야 합니다. 자세한 내용은 Oracle 버전의 JCE 또는 IBM 버전의 JCE 문서를 참조하십시오.

다음 절차에서는 인증 및 권한 부여를 위해 Kerberos를 구성하는 방법을 보여줍니다.

5.5.1. Kerberos를 사용하도록 네트워크 연결 구성

AMQ Broker는 SASL(Simple Authentication and Security Layer) 프레임워크의 GSSAPI 메커니즘을 사용하여 Kerberos 보안 자격 증명과 통합됩니다. AMQ Broker에서 Kerberos를 사용하려면 각 수락자가 Kerberos 인증 정보를 사용하는 클라이언트를 인증하거나 승인하려면 GSSAPI 메커니즘을 사용하도록 구성해야 합니다.

다음 절차에서는 Kerberos를 사용하도록 어셉터를 구성하는 방법을 보여줍니다.

사전 요구 사항

- AMQ Broker가 Kerberos 인증 정보를 인증하고 인증하려면 Kerberos 인프라를 배포하고 구성해야 합니다.

프로세스

1. 브로커를 중지합니다.
   a. Linux의 경우:
      -

```
<broker_instance_dir>/bin/artemis stop
```

b. ***Windows*에서:**

```
<broker_instance_dir>\bin\artemis-service.exe stop
```

2. ***&lt ;broker_instance_dir> /etc/broker.xml* 구성 파일을 엽니다.**

3. *허용되는 **URL**의 쿼리 문자열에 이름-값 쌍 **saslMechanisms=GSSAPI** 를 추가합니다.*

```
<acceptor name="amqp">
  tcp://0.0.0.0:5672?protocols=AMQP;saslMechanisms=GSSAPI
</acceptor>
```

*이전 구성은 수락자가 **Kerberos** 자격 증명을 인증할 때 **GSSAPI** 메커니즘을 사용한다는 것을 의미합니다.*

4. *(선택 사항) **PLAIN** 및 **ANONYMOUS SASL** 메커니즘도 지원됩니다. 여러 메커니즘을 지정하려면 쉼표로 구분된 목록을 사용합니다. 예를 들면 다음과 같습니다.*

```
<acceptor name="amqp">
  tcp://0.0.0.0:5672?protocols=AMQP;saslMechanisms=GSSAPI,PLAIN
</acceptor>
```

*그 결과는 **GSSAPI** 및 **PLAIN SASL** 메커니즘을 모두 사용하는 어셉터입니다.*

5. *브로커를 시작합니다.*

a. ***Linux*의 경우:**

```
<broker_instance_dir>/bin/artemis run
```

b. ***Windows*에서:**

```
<broker_instance_dir>\bin\artemis-service.exe start
```

추가 리소스

- 허용자에 대한 자세한 내용은 2.1절. “승인 정보” 을 참조하십시오.

5.5.2. Kerberos 인증 정보를 사용하여 클라이언트 인증

AMQ Broker는 SASL(Simple Authentication and Security Layer) 프레임워크의 GSSAPI 메커니즘을 사용하는 AMQP 연결의 Kerberos 인증을 지원합니다.

브로커는 JAAS(Java Authentication and Authorization Service)를 사용하여 Kerberos 어셉터 인증 정보를 가져옵니다. Java 설치에 포함된 JAAS 라이브러리는 Kerberos 인증 정보를 인증하는 로그인 모듈인 Krb5LoginModule 과 함께 패키지됩니다. Krb5LoginModule 에 대한 자세한 내용은 Java 벤더의 설명서를 참조하십시오. 예를 들어 Oracle은 Java 8 설명서 의 일부로 Krb5LoginModule 로그인 모듈에 대한 정보를 제공합니다.

사전 요구 사항

- Kerberos 보안 자격 증명을 사용하여 AMQP 연결을 인증하려면 승인자의 GSSAPI 메커니즘을 활성화해야 합니다. 자세한 내용은 5.5.1절. “Kerberos를 사용하도록 네트워크 연결 구성”의 내용을 참조하십시오.

프로세스

1. 브로커를 중지합니다.

   a. Linux의 경우:

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. Windows에서:

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. &lt ;broker_instance_dir> /etc/login.config 구성 파일을 엽니다.

3. amqp-sasl-gssapi 라는 구성 범위를 추가합니다. 다음 예제에서는 Oracle 및 OpenJDK 버전의 JDK에서 발견된 Krb5LoginModule 에 대한 구성을 보여줍니다.

```
amqp-sasl-gssapi {
   com.sun.security.auth.module.Krb5LoginModule required
   isInitiator=false
   storeKey=true
   useKeyTab=true
   principal="amqp/my_broker_host@example.com"
   debug=true;
};
```

amqp-sasl-gssapi

기본적으로 브로커의 **GSSAPI** 메커니즘 구현에서는 **amqp-sasl-gssapi** 라는 **JAAS** 구성 범위를 사용하여 **Kerberos** 어셉터 인증 정보를 가져옵니다.

Krb5LoginModule

이 버전의 **Krb5LoginModule** 은 **JDK**의 **Oracle** 및 **OpenJDK** 버전에서 제공됩니다. **Java** 벤더의 설명서를 참조하여 **Krb5LoginModule** 의 정규화된 클래스 이름과 사용 가능한 옵션을 확인합니다.

useKeyTab

Krb5LoginModule 은 보안 주체를 인증할 때 **Kerberos keytab**을 사용하도록 구성되어 있습니다. 키탭은 **Kerberos** 환경의 툴링을 사용하여 생성됩니다. **Kerberos** 키탭 생성에 대한 자세한 내용은 벤더의 설명서를 참조하십시오.

주체

보안 주체는 **amqp/my_broker_host@example.com** 로 설정됩니다. 이 값은 **Kerberos** 환경에서 생성된 서비스 주체에 해당해야 합니다. 서비스 주체 생성에 대한 자세한 내용은 벤더의 설명서를 참조하십시오.

4. 브로커를 시작합니다.

   a. **Linux**의 경우:

   ```
   <broker_instance_dir>/bin/artemis run
   ```

   b. **Windows**에서:

   ```
   <broker_instance_dir>\bin\artemis-service.exe start
   ```

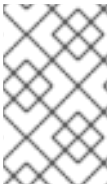
5.5.2.1. 대체 구성 범위 사용

AMQP 수락자의 URL에 saslLoginConfigScope 매개변수를 추가하여 대체 구성 범위를 지정할 수 있습니다. 다음 구성 예제에서 매개 변수 saslLoginConfigScope 에는 alternative-sasl-gssapi 값이 제공됩니다. 그 결과는 < broker_instance_dir> /etc/login.config 구성 파일에 선언된 alternative-sasl-gssapi 라는 대체 범위를 사용하는 어셉터입니다.

```
<acceptor name="amqp">
tcp://0.0.0.0:5672?
protocols=AMQP;saslMechanisms=GSSAPI,PLAIN;saslLoginConfigScope=alternative-sasl-gssapi`
</acceptor>
```

5.5.3. Kerberos 인증 정보를 사용하여 클라이언트 인증

AMQ Broker에는 역할을 매핑할 때 다른 보안 모듈에서 사용할 JAAS Krb5LoginModule 로그인 모듈이 포함됩니다. 이 모듈은 Kerberos 인증 주체를 AMQ Broker UserPrincipal로 설정된 주체의 주체에 추가합니다. 그러면 인증 정보를 Kerberos 인증 Peer Principal를 AMQ Broker 역할에 매핑하는 PropertiesLoginModule 또는 LDAPLoginModule 모듈에 전달할 수 있습니다.



참고

Kerberos Peer Principal는 역할 멤버로만 브로커 사용자로 존재하지 않습니다.

사전 요구 사항

- Kerberos 보안 자격 증명을 사용하여 AMQP 연결을 인증하려면 승인자의 GSSAPI 메커니즘을 활성화해야 합니다.

프로세스

1. 브로커를 중지합니다.

   a. Linux의 경우:

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. Windows에서:

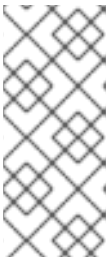
   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. *<broker_instance_dir>/etc/login.config 구성 파일을 엽니다.*

3. *AMQ Broker Krb5LoginModule 및 LDAPLoginModule 에 대한 구성을 추가합니다. LDAP 공급자의 설명서를 참조하여 구성 옵션을 확인합니다.*

   구성 예는 다음과 같습니다.

```
org.apache.activemq.artemis.spi.core.security.jaas.Krb5LoginModule required
  ;
org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule optional
  initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
  connectionURL="ldap://localhost:1024"
  authentication=GSSAPI
  saslLoginConfigScope=broker-sasl-gssapi
  connectionProtocol=s
  userBase="ou=users,dc=example,dc=com"
  userSearchMatching="(krb5PrincipalName={0})"
  userSearchSubtree=true
  authenticateUser=false
  roleBase="ou=system"
  roleName=cn
  roleSearchMatching="(member={0})"
  roleSearchSubtree=false
  ;
```



참고

이전 예에 표시된 Krb5LoginModule 버전은 AMQ Broker와 함께 배포되며, 역할 매핑을 위해 다른 AMQ 모듈에서 사용할 수 있는 브로커 ID로 Kerberos ID를 변환합니다.

4. *브로커를 시작합니다.*

   a. ***Linux의 경우:***

```
<broker_instance_dir>/bin/artemis run
```

   b. ***Windows에서:***

```
<broker_instance_dir>\bin\artemis-service.exe start
```

추가 리소스

- AMQ Broker에서 GSSAPI 메커니즘 활성화에 대한 자세한 내용은 5.5.1절. “Kerberos를 사용하도록 네트워크 연결 구성” 을 참조하십시오.
- PropertiesLoginModule 에 대한 자세한 내용은 5.2.2.1절. “기본 사용자 및 암호 인증 구성” 을 참조하십시오.
- LDAPLoginModule 에 대한 자세한 내용은 5.4.1절. “클라이언트 인증을 위해 LDAP 구성” 을 참조하십시오.

5.6. 보안 관리자 지정

브로커는 보안 관리자 라는 구성 요소를 사용하여 인증 및 권한 부여를 처리합니다.

AMQ Broker에는 두 가지 보안 관리자가 포함되어 있습니다.

- ActiveMQJAASSecurityManager 보안 관리자입니다. 이 보안 관리자는 JAAS 및 Red Hat JBoss EAP(JBoss Enterprise Application Platform) 보안과의 통합을 제공합니다. AMQ Broker에서 사용하는 기본 보안 관리자입니다.
- ActiveMQBasicSecurityManager 보안 관리자입니다. 이 기본 보안 관리자는 JAAS를 지원하지 않습니다. 대신 사용자 이름과 암호 자격 증명을 통해 인증 및 권한 부여를 지원합니다. 이 보안 관리자는 관리 API를 사용하여 사용자 추가, 제거 및 업데이트를 지원합니다. 모든 사용자 및 역할 데이터는 브로커 바인딩 저널에 저장됩니다. 즉, 라이브 브로커에 대한 변경 사항도 백업 브로커에서 사용할 수 있습니다.

포함된 보안 관리자 대신 시스템 관리자가 브로커 보안 구현을 보다 효과적으로 제어하고자 할 수 있습니다. 이 경우 브로커 구성에서 사용자 지정 보안 관리자를 지정할 수도 있습니다. 사용자 정의 보안 관리자는 org.apache.activemq.artemis.spi.core.security.ActiveMQSecurityManager5 인터페이스를 구현하는 사용자 정의 클래스입니다.

다음 하위 섹션의 예제에서는 사용할 브로커를 구성하는 방법을 보여줍니다.

- 기본 JAAS 보안 관리자 대신 기본 보안 관리자

- 사용자 정의 보안 관리자

5.6.1. 기본 보안 관리자 사용

AMQ Broker에는 기본 **ActiveMQJAASSecurityManager** 보안 관리자 외에도 **ActiveMQBasicSecurityManager** 보안 관리자도 포함되어 있습니다.

기본 보안 관리자를 사용하면 모든 사용자 및 역할 데이터가 바인딩 저널(또는 **JDBC** 지속성을 사용하는 경우 바인딩 테이블)에 저장됩니다. 따라서 라이브 백업 브로커 그룹을 구성한 경우 라이브 브로커에 포맷한 모든 사용자 관리는 장애 조치 시 백업 브로커에 자동으로 반영됩니다. 이렇게 하면 이러한 동작을 수행하는 대체 방법인 **LDAP** 서버를 별도로 관리할 필요가 없습니다.

기본 보안 관리자를 구성하고 사용하기 전에 다음 사항에 유의하십시오.

- 기본 보안 관리자는 기본 **JAAS** 보안 관리자처럼 연결할 수 없습니다.
- 기본 보안 관리자는 **JAAS**를 지원하지 않습니다. 대신 사용자 이름과 암호 자격 증명을 통한 인증 및 권한 부여만 지원합니다.
- **AMQ** 관리 콘솔에는 **JAAS**가 필요합니다. 따라서 기본 보안 관리자를 사용하고 콘솔을 사용하려면 사용자 및 암호 인증에 대한 **login.config** 구성 파일도 구성해야 합니다. 사용자 및 암호 인증 구성에 대한 자세한 내용은 **5.2.2.1**절. “기본 사용자 및 암호 인증 구성” 을 참조하십시오.
- **AMQ Broker**에서 사용자 관리는 브로커 관리 **API**에서 제공합니다. 이 관리에는 사용자와 역할을 추가, 나열, 업데이트 및 제거하는 기능이 포함됩니다. **Cryostat**, 관리 메시지, **HTTP(Jokia** 또는 **AMQ Management Console** 사용) 및 **AMQ Broker** 명령줄 인터페이스를 사용하여 이러한 기능을 수행할 수 있습니다. 브로커가 이 데이터를 직접 저장하므로 사용자를 관리하려면 브로커가 실행 중이어야 합니다. 바인딩 데이터를 수동으로 수정할 수 없습니다.
- **HTTP**를 통한 관리 액세스(예: **Jolokia** 또는 **AMQ Management Console** 사용)는 **JAAS** 로그인 모듈에서 처리합니다. **JConsole** 또는 기타 원격 **Cryostat** 툴을 통한 **Cryostat** 액세스는 기본 보안 관리자가 처리합니다. 관리 메시지는 기본 보안 관리자가 처리합니다.

5.6.1.1. 기본 보안 관리자 구성

다음 절차에서는 기본 보안 관리자를 사용하도록 브로커를 구성하는 방법을 보여줍니다.

프로세스

1. &lt ;broker-instance-dir> /etc/boostrap.xml 구성 파일을 엽니다.

2. security-manager 요소에서 class-name 속성의 전체 ActiveMQBasicSecurityManager 클래스 이름을 지정합니다.

```
<broker xmlns="http://activemq.org/schema">
  ...
  <security-manager class-
name="org.apache.activemq.artemis.spi.core.security.ActiveMQBasicSecurityManager">
  </security-manager>
  ...
</broker>
```

3. 사용자 및 역할 데이터를 보유하는 바인딩 데이터를 수동으로 수정할 수 없으며 사용자를 관리하기 위해 브로커가 실행 중이어야 하므로 처음 부팅할 때 브로커를 보호하는 것이 좋습니다. 이를 위해 인증 정보를 사용하여 다른 사용자를 추가할 수 있는 부트스트랩 사용자를 정의합니다.

security-manager 요소에서 bootstrapUser,bootstrapPassword, bootstrapRole 속성을 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<broker xmlns="http://activemq.org/schema">
  ...
  <security-manager class-
name="org.apache.activemq.artemis.spi.core.security.ActiveMQBasicSecurityManager">
      <property key="bootstrapUser" value="myUser"/>
      <property key="bootstrapPassword" value="myPass"/>
      <property key="bootstrapRole" value="myRole"/>
  </security-manager>
  ...
</broker>
```

bootstrapUser

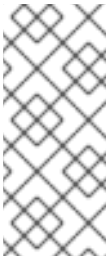
부트스트랩 사용자의 이름입니다.

bootstrapPassword

boostrap 사용자의 본문을 전달합니다. 암호화된 암호를 지정할 수도 있습니다.

bootstrapRole

boostrap 사용자의 역할.



참고

구성에서 부트스트랩 사용자의 이전 속성을 정의하는 경우 브로커가 실행되는 동안 변경한 내용에 관계없이 브로커를 시작할 때마다 해당 인증 정보가 설정됩니다.

4. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

5. broker.xml 구성 파일에서 activemq.management# 주소와 일치하도록 기본적으로 정의된 address-setting 요소를 찾습니다. 이러한 기본 주소 설정은 다음과 같습니다.

```
<address-setting match="activemq.management#">
  <dead-letter-address>DLQ</dead-letter-address>
  <expiry-address>ExpiryQueue</expiry-address>
  <redelivery-delay>0</redelivery-delay>
  <!--...-->
  <max-size-bytes>-1</max-size-bytes>
  <message-counter-history-day-limit>10</message-counter-history-day-limit>
  <address-full-policy>PAGE</address-full-policy>
  <auto-create-queues>true</auto-create-queues>
  <auto-create-addresses>true</auto-create-addresses>
  <auto-create-jms-queues>true</auto-create-jms-queues>
  <auto-create-jms-topics>true</auto-create-jms-topics>
</address-setting>
```

6. activemq.management# 주소의 주소 설정 내에서 이 절차의 앞부분에서 지정한 부트스트랩 역할 이름에 대해 다음과 같은 필수 권한을 추가합니다.

- createNonDurableQueue
- createAddress
- consume
- 관리
- 전송

예를 들면 다음과 같습니다.

```
<address-setting match="activemq.management#">
  ...
  <permission type="createNonDurableQueue" roles="myRole"/>
  <permission type="createAddress" roles="myRole"/>
  <permission type="consume" roles="myRole"/>
  <permission type="manage" roles="myRole"/>
  <permission type="send" roles="myRole"/>
</address-setting>
```

추가 리소스

- **ActiveMQBasicSecurityManager** 클래스에 대한 자세한 내용은 **ActiveMQ Artemis Core API** 설명서의 **Class ActiveMQBasicSecurityManager** 를 참조하십시오.

- 구성 파일에서 암호를 암호화하는 방법을 알아보려면 **5.9절. "구성 파일에서 암호 암호화"** 을 참조하십시오.

5.6.2. 사용자 정의 보안 관리자 지정

다음 절차에서는 브로커 구성에서 사용자 지정 보안 관리자를 지정하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/boostrap.xml** 구성 파일을 엽니다.

2. **security-manager** 요소에서 **class-name** 속성에 대해 **org.apache.activemq.artemis.spi.security.ActiveMQSecurityManager5** 인터페이스의 사용자 정의 구현인 클래스를 지정합니다. 예를 들면 다음과 같습니다.

```
<broker xmlns="http://activemq.org/schema">
  ...
  <security-manager class-name="com.myclass.MySecurityManager">
    <property key="myKey1" value="myValue1"/>
    <property key="myKey2" value="myValue2"/>
  </security-manager>
  ...
</broker>
```

추가 리소스

- *ActiveMQSecurityManager5 인터페이스에 대한 자세한 내용은 ActiveMQ Artemis Core API 설명서의 Interface ActiveMQSecurityManager5 를 참조하십시오.*

5.6.3. 사용자 정의 보안 관리자 예제 프로그램 실행

AMQ Broker에는 사용자 정의 보안 관리자를 구현하는 방법을 보여주는 예제 프로그램이 있습니다. 이 예제에서 사용자 지정 보안 관리자는 인증 및 권한 부여에 대한 세부 정보를 기록한 다음 세부 정보를 ActiveMQJAASSecurityManager (즉, 기본 보안 관리자) 인스턴스에 전달합니다.

다음 절차에서는 사용자 정의 보안 관리자 예제 프로그램을 실행하는 방법을 보여줍니다.

사전 요구 사항

- *시스템이 AMQ Broker 예제 프로그램을 실행하도록 설정되어 있습니다. 자세한 내용은 AMQ Broker 예제 실행을 참조하십시오.*

사용자 정의 보안 관리자 예제 를 다운로드했습니다.

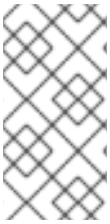
프로세스

1. *사용자 지정 보안 관리자 예제가 포함된 디렉터리로 이동합니다. 다음 예제에서는 예제를 amq-broker-examples 라는 디렉터리에 다운로드했다고 가정합니다.*

```
$ cd amq-broker-examples/examples/features/standard/security-manager
```

2. *예제를 실행합니다.*

```
$ mvn verify
```



참고

예제 프로그램을 실행할 때 브로커 인스턴스를 수동으로 생성하고 시작하려면 이전 단계에서 명령을 mvn -PnoServer 확인 으로 바꿉니다.

추가 리소스

- *ActiveMQJAASSecurityManager 클래스에 대한 자세한 내용은 ActiveMQ Artemis Core*

API 설명서의 Class ActiveMQJAASSecurityManager 를 참조하십시오.

5.7. 보안 비활성화

보안은 기본적으로 활성화되어 있습니다. 다음 절차에서는 브로커 보안을 비활성화하는 방법을 보여줍니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. core 요소에서 security-enabled 값을 false 로 설정합니다.

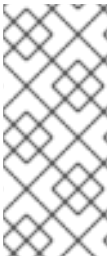
```
<security-enabled>false</security-enabled>
```

3. 필요한 경우 security-invalidation-interval 에 대해 새 값을 밀리초 단위로 지정합니다. 이 속성 값은 브로커가 보안 로그인을 주기적으로 무효화하는 시기를 지정합니다. 기본값은 10000 입니다.

5.8. 검증된 사용자의 메시지 추적

메시지 출처를 추적하고 로깅하려면 (예: 보안 감사의 경우) _AMQ_VALIDATED_USER 메시지 키를 사용할 수 있습니다.

broker.xml 구성 파일에서 populate-validated-user 옵션이 true 로 설정된 경우 브로커는 _AMQ_VALIDATED_USER 키를 사용하여 검증된 사용자의 이름을 메시지에 추가합니다. JMS 및 STOMP 클라이언트의 경우 이 메시지 키는 JMSXUserID 키에 매핑됩니다.



참고

브로커는 AMQP JMS 클라이언트에서 생성한 메시지에 검증된 사용자 이름을 추가할 수 없습니다. 클라이언트에서 AMQP 메시지를 보낸 후 AMQP 메시지 속성을 수정하는 것은 AMQP 프로토콜을 위반하는 것입니다.

SSL 인증서를 기반으로 인증된 사용자의 경우 브로커가 입력한 검증된 사용자 이름은 인증서의 고유 이름(DN)이 매핑되는 이름입니다.

broker.xml 구성 파일에서 security-enabled 가 false 이고 populate-validated-user 가 true 인 경우 브로커는 클라이언트에서 제공하는 사용자 이름을 채웁니다. populate-validated-user 옵션은 기본적으로 false 입니다.

사용자 이름(즉, JMSXUserID 키)이 메시지를 전송할 때 클라이언트에서 이미 채워지는 메시지를 거부하도록 브로커를 구성할 수 있습니다. 브로커가 이러한 클라이언트가 보낸 메시지에 대해 검증된 사용자 이름 자체를 채울 수 없기 때문에 AMQP 클라이언트에 이 옵션이 유용할 수 있습니다.

클라이언트에서 설정한 JMSXUserID 없이 메시지를 거부하도록 브로커를 구성하려면 broker.xml 구성 파일에 다음 구성을 추가합니다.

```
<reject-empty-validated-user>true</reject-empty-validated-user>
```

기본적으로 reject-empty-validated-user 는 false 로 설정됩니다.

5.9. 구성 파일에서 암호 암호화

기본적으로 AMQ Broker는 모든 암호를 구성 파일에 일반 텍스트로 저장합니다. 무단 액세스를 방지하기 위해 올바른 권한으로 모든 구성 파일을 보호하십시오. 또한 원하지 않는 뷰어가 읽지 못하도록 일반 텍스트 암호를 암호화하거나 마스크 할 수도 있습니다.

5.9.1. 암호화된 암호 정보

암호화된 또는 마스크된 암호는 일반 텍스트 암호의 암호화된 버전입니다. 암호화된 버전은 AMQ Broker에서 제공하는 mask 명령줄 유틸리티로 생성됩니다. mask 유틸리티에 대한 자세한 내용은 명령줄 도움말 설명서를 참조하십시오.

```
$ <broker_instance_dir>/bin/artemis help mask
```

암호를 마스킹하려면 일반 텍스트 값을 암호화된 값으로 바꿉니다. 실제 값이 필요할 때 암호가 해독되도록 마스킹된 암호는 식별자 ENC() 로 래핑되어야 합니다.

다음 예에서 구성 파일 < broker_instance_dir> /etc/bootstrap.xml 에는 keyStorePassword 및 trustStorePassword 매개변수에 대한 마스크된 암호가 포함되어 있습니다.

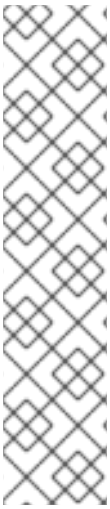
```
<web bind="https://localhost:8443" path="web"
    keyStorePassword="ENC(-342e71445830a32f95220e791dd51e82)"
    trustStorePassword="ENC(32f94e9a68c45d89d962ee7dc68cb9d1)">
```

```
    <app url="activemq-branding" war="activemq-branding.war"/>
</web>
```

다음 구성 파일과 함께 마스킹된 암호를 사용할 수 있습니다.

- **broker.xml**
- **bootstrap.xml**
- **management.xml**
- **artemis-users.properties**
- **login.config (LDAPLoginModule과 함께 사용하는 경우)**

구성 파일은 **< broker_instance_dir> /etc** 에서 확인할 수 있습니다.



참고

Artemis-users.properties 는 해시된 마스크된 암호만 지원합니다. 사용자가 브로커 생성 시 생성되는 경우 **artemis-users.properties** 에는 기본적으로 해시된 암호가 포함됩니다. 기본 **PropertiesLoginModule** 은 **artemis-users.properties** 파일에서 암호를 디코딩하지 않지만 대신 입력을 해시하고 암호 확인을 위해 해시된 두 값을 비교합니다. 해시된 암호를 마스킹된 암호로 변경해도 **AMQ Broker** 관리 콘솔에 액세스할 수 없습니다.

broker.xml,bootstrap.xml,management.xml 및 **login.config** 는 마스킹되었지만 해시되지 않은 암호를 지원합니다.

5.9.2. 구성 파일에서 암호 암호화

다음 예제에서는 **broker.xml** 구성 파일에서 **cluster-password** 값을 마스킹하는 방법을 보여줍니다.

프로세스

1. 명령 프롬프트에서 **mask** 유틸리티를 사용하여 암호를 암호화합니다.

```
$ <broker_instance_dir>/bin/artemis mask <password>
```

```
result: 3a34fd21b82bf2a822fa49a8d8fa115d
```

2. 마스크하려는 일반 텍스트 암호가 포함된 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

```
<cluster-password>
  <password>
</cluster-password>
```

3. 일반 텍스트 암호를 암호화된 값으로 바꿉니다.

```
<cluster-password>
  3a34fd21b82bf2a822fa49a8d8fa115d
</cluster-password>
```

4. 암호화된 값을 식별자 **ENC()** 로 래핑합니다.

```
<cluster-password>
  ENC(3a34fd21b82bf2a822fa49a8d8fa115d)
</cluster-password>
```

이제 설정 파일에 암호화된 암호가 포함됩니다. **ENC()** 식별자로 암호가 래핑되므로 **AMQ Broker**는 사용하기 전에 암호를 해독합니다.

추가 리소스

- **AMQ Broker**에 포함된 구성 파일에 대한 자세한 내용은 1.1절. “AMQ Broker 구성 파일 및 위치” 을 참조하십시오.

5.9.3. 암호를 암호화하고 해독하도록 codec 키 설정

암호를 암호화하고 해독하는 데 **Codec**가 필요합니다. 사용자 정의 **codec**가 구성되지 않은 경우 마스크 유틸리티는 기본 **codec**를 사용하여 암호를 암호화하고 **AMQ Broker**는 동일한 기본 **codec**를 사용하여 암호를 해독합니다. **codec**는 암호를 암호화하고 암호 해독하기 위해 기본 암호화 알고리즘에 제공하

는 기본 키로 구성됩니다. 기본 키를 사용하면 암호를 해독하는 데 악의적인 행위자가 키를 사용할 수 있는 위험이 노출됩니다.

mask 유틸리티를 사용하여 암호를 암호화하면 기본 **codec** 키를 사용하지 않도록 고유 키 문자열을 지정할 수 있습니다. 그런 다음 브로커가 암호를 해독할 수 있도록 **ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY** 환경 변수에서 동일한 키 문자열을 설정해야 합니다. 환경 변수에 키를 설정하면 구성 파일에 유지되지 않으므로 더 안전합니다. 또한 브로커를 시작하고 브로커가 시작된 직후에 키를 설정할 수 있습니다.

프로세스

1. 마스크 유틸리티를 사용하여 구성 파일의 각 암호를 암호화합니다. **key** 매개변수의 경우 암호를 암호화할 문자 문자열을 지정합니다. 동일한 키 문자열을 사용하여 각 암호를 암호화합니다.

```
$ <broker_instance_dir>/bin/artemis mask --key <key> <password>
```

주의

mask 유틸리티를 실행하여 암호를 암호화할 때 지정하는 키 문자열의 레코드를 유지해야 합니다. 브로커가 암호 해독을 허용하도록 환경 변수에 동일한 키 값을 구성해야 합니다.

구성 파일에서 암호를 암호화하는 방법에 대한 자세한 내용은 **5.9.2**절. “구성 파일에서 암호 암호화” 을 참조하십시오.

2. 명령 프롬프트에서 **ARTEMIS_DEFAULT_SENSITIVE_CODEC_KEY** 환경 변수를 각 암호를 암호화할 때 지정한 키 문자열로 설정합니다.

```
$ export ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY= <key>
```

3. 브로커를 시작합니다.

```
$ ./artemis run
```

4. **ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY** 환경 변수를 설정 해제합니다.

```
$ unset ARTEMIS_DEFAULT_SENSITIVE_STRING_CODEC_KEY
```

참고

브로커를 시작한 후 **ARTEMIS_DEFAULT_SENSITIVE_CODEC_KEY** 환경 변수를 설정 해제하는 경우 나중에 브로커를 시작하기 전에 동일한 키 문자열로 다시 설정해야 합니다.

5.10. 인증 및 권한 부여 캐싱 구성

기본적으로 **AMQ Broker**는 인증 및 권한 부여 응답에 대한 정보를 별도의 캐시에 저장합니다. 각 캐시에 허용되는 기본 항목 수와 캐시되는 항목의 기간을 변경할 수 있습니다.

1. **&lt ;broker-instance-dir>/etc/broker.xml** 구성 파일을 엽니다.

2. 각 캐시에 허용되는 기본 최대 항목 수(**1000**)를 변경하려면 **authentication-cache-size** 및 **authorization-cache-size** 매개변수를 설정합니다. 예를 들면 다음과 같습니다.

```
<configuration>
 ...
 <core>
  ...
  <authentication-cache-size>2000</authentication-cache-size>
  <authorization-cache-size>1500</authorization-cache-size>
  ...
 </core>
 ...
</configuration>
```

참고

캐시가 제한 세트에 도달하면 최근 사용된 항목이 캐시에서 제거됩니다.

3. 기본 기간, **10000** 밀리초의 항목을 캐시하려면 **security-invalidation-interval** 매개변수를 설정합니다. 예를 들면 다음과 같습니다.

```
<configuration>
```

```
...
<core>
  ...
  <security-invalidation-interval>20000</security-invalidation-interval>
  ...
</core>
...
</configuration>
```

참고

security-invalidation-interval 매개변수를 **0** 으로 설정하면 인증 및 권한 부여 캐싱이 비활성화됩니다.

6장. 메시지 데이터 유지

AMQ Broker에는 메시지 데이터를 유지(즉, 저장)하기위한 두 가지 옵션이 있습니다.

저널에 메시지 저장

이는 기본 옵션입니다. 저널 기반 지속성은 파일 시스템의 저널에 메시지를 쓰는 고성능 옵션입니다.

데이터베이스에 메시지 저장

이 옵션은 JDBC(Java Database Connectivity) 연결을 사용하여 선택한 데이터베이스에 메시지를 저장합니다.

또는 메시지 데이터를 유지하지 않도록 브로커를 구성할 수도 있습니다. 자세한 내용은 6.3절. “지속성 비활성화”의 내용을 참조하십시오.

브로커는 메시지 저널 외부에 큰 메시지를 저장하기 위해 다른 솔루션을 사용합니다. 자세한 내용은 8장. 대용량 메시지 처리를 참조하십시오.

메모리 부족 상황에서 브로커를 디스크에 메시지를 페이징하도록 구성할 수도 있습니다. 자세한 내용은 7.1절. “메시지 페이징 구성”를 참조하십시오.

참고

AMQ Broker에서 지원하는 데이터베이스 및 네트워크 파일 시스템에 대한 최신 정보는 Red Hat 고객 포털에서 Red Hat AMQ 7 지원 구성 을 참조하십시오.

6.1. 저널에 메시지 데이터 유지

브로커 저널은 디스크의 추가 전용 파일 집합입니다. 각 파일은 고정된 크기로 미리 생성되며 처음에 패딩으로 채워집니다. 브로커에서 메시징 작업이 수행되면 기록이 저널 끝에 추가됩니다. 레코드를 추가하면 브로커가 디스크 헤드 이동 및 임의의 액세스 작업을 최소화할 수 있으며 일반적으로 디스크에서 가장 느린 작업입니다. 하나의 저널 파일이 가득 차면 브로커는 새 파일을 생성합니다.

저널 파일 크기는 구성 가능하며 각 파일에서 사용하는 디스크의 수를 최소화합니다. 그러나 최신 디스크 토폴로지는 복잡하며 브로커는 파일이 매핑되는 것을 제어할 수 없습니다. 따라서 저널 파일 크기 조정은 정확하게 제어하기 어렵습니다.

브로커에서 사용하는 기타 지속성 관련 기능은 다음과 같습니다.

- 특정 저널 파일이 아직 사용 중인지 여부를 결정하는 가비지 컬렉션 알고리즘입니다. 저널 파일이 더 이상 사용되지 않는 경우 브로커는 재사용할 수 있도록 파일을 회수할 수 있습니다.
- 저널에서 종료된 공간을 제거하고 데이터를 압축하는 압축 알고리즘입니다. 그러면 저널에서 디스크에서 파일을 더 적게 사용합니다.
- 로컬 트랜잭션을 지원합니다.
- JMS 클라이언트를 사용할 때 XA(확장 아키텍처) 트랜잭션 지원.

대부분의 저널은 Java로 작성됩니다. 그러나 다양한 플러그형 구현을 사용할 수 있도록 실제 파일 시스템과의 상호 작용이 요약됩니다. AMQ Broker에는 다음 구현이 포함됩니다.

NIO

NIO(New I/O)는 표준 Java NIO를 사용하여 파일 시스템과 연결합니다. 이는 매우 우수한 성능을 제공하며 Java 6 이상 런타임을 사용하는 모든 플랫폼에서 실행됩니다. Java NIO에 대한 자세한 내용은 Java NIO 를 참조하십시오.

AIO

AIO(Aynshcronous I/O)는 씬 네이티브 래퍼를 사용하여 Linux 비동기 I/O 라이브러리(libaio)와 통신합니다. AIO를 사용하면 데이터를 디스크로 만든 후 브로커가 다시 호출되어 명시적 동기화를 완전히 방지합니다. 기본적으로 브로커는 AIO 저널을 사용하려고 하며 AIO를 사용할 수 없는 경우 NIO를 사용합니다.

AIO는 일반적으로 Java NIO보다 더 나은 성능을 제공합니다. libaio 를 설치하는 방법에 대한 자세한 내용은 6.1.1절. “Linux 비동기 I/O 라이브러리 설치” 을 참조하십시오.

다음 하위 섹션의 절차에서는 저널 기반 지속성을 위해 브로커를 구성하는 방법을 보여줍니다.

6.1.1. Linux 비동기 I/O 라이브러리 설치

Red Hat은 지속성 성능을 위해 AIO 저널(NIO 대신)을 사용하는 것이 좋습니다.

참고

AIO 저널을 다른 운영 체제 또는 이전 버전의 Linux 커널과 함께 사용할 수 없습니다.

AIO 저널을 사용하려면 Linux 비동기 I/O 라이브러리(libaio)를 설치해야 합니다. libaio 를 설치하려면 다음과 같이 yum 명령을 사용합니다.

```
yum install libaio
```

6.1.2. 저널 기반 지속성 구성

다음 절차에서는 브로커가 저널 기반 지속성에 사용하는 기본 구성을 검토하는 방법을 설명합니다. 이 설명을 사용하여 필요에 따라 구성을 조정할 수 있습니다.

1. <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

기본적으로 브로커는 다음과 같이 저널 기반 지속성을 사용하도록 구성됩니다.

```
<configuration>
 <core>
  ...
  <persistence-enabled>true</persistence-enabled>
  <journal-type>ASYNCIO</journal-type>
  <bindings-directory>./data/bindings</bindings-directory>
  <journal-directory>./data/journal</journal-directory>
  <journal-datasync>true</journal-datasync>
  <journal-min-files>2</journal-min-files>
  <journal-pool-files>-1</journal-pool-files>
  <journal-device-block-size>4096</journal-device-block-size>
  <journal-file-size>10M</journal-file-size>
  <journal-buffer-timeout>12000</journal-buffer-timeout>
  <journal-max-io>4096</journal-max-io>
  ...
 </core>
</configuration>
```

persistence-enabled

이 매개변수의 값이 true 로 설정된 경우 브로커는 메시지 지속성을 위해 파일 기반 저널을 사용합니다.

journal-type

사용할 저널 유형입니다. ASYNCIO 로 설정하면 브로커가 먼저 AIO를 사용하려고 시도

합니다. **AIO**를 찾을 수 없는 경우 브로커는 **NIO**를 사용합니다.

bindings-directory

바인딩 저널의 파일 시스템 위치입니다. 기본값은 **< broker_instance_dir>** 디렉터리에 상대적 값 입니다.

journal-directory

메시지 저널의 파일 시스템 위치입니다. 기본값은 **< broker_instance_dir>** 디렉터리에 상대적 값 입니다.

journal-datasync

이 매개변수의 값이 **true** 로 설정된 경우 브로커는 **fdatasync** 함수를 사용하여 디스크 쓰기를 확인합니다.

journal-min-files

브로커가 시작될 때 처음에 생성할 저널 파일 수입니다.

journal-pool-files

사용되지 않는 파일을 회수한 후 보관할 파일 수입니다. 기본값 **-1** 은 정리 중에 파일이 삭제되지 않음을 의미합니다.

journal-device-block-size

스토리지 장치의 저널에서 사용하는 데이터 블록의 최대 크기(바이트)입니다. 기본값은 **4096**바이트입니다.

journal-file-size

지정된 저널 디렉터리에 있는 각 저널 파일의 최대 크기(바이트)입니다. 이 제한에 도달하면 브로커가 새 파일을 시작합니다. 이 매개변수는 바이트 표기법(예: **K, M, G**) 또는 바이너리 동등한 바이너리(**Ki, Mi, Gi**)도 지원합니다. 이 매개변수가 구성에 명시적으로 지정되지 않은 경우 기본값은 **10485760**바이트(**10MiB**)입니다.

journal-buffer-timeout

나노초 단위에서 브로커가 저널 버퍼를 플러시하는 빈도를 지정합니다. **AIO**는 일반적으로 **NIO**보다 높은 플러시 속도를 사용하므로 브로커는 **NIO** 및 **AIO** 모두에 대해 다른 기본값을 유지합니다. 이 매개변수가 구성에 명시적으로 지정되지 않은 경우 **NIO**의 기본값은 **3333333** 나노초입니다(즉, 초당 **300**회). **AIO**의 기본값은 **50000** 나노초(즉, 초당 **2000**회)입니다.

journal-max-io

한 번에 **IO** 큐에 있을 수 있는 최대 쓰기 요청 수입니다. 큐가 가득 차면 브로커는 공간을 사용할 수 있을 때까지 추가 쓰기를 차단합니다.

NIO를 사용하는 경우 이 값은 항상 1 이어야 합니다. 구성에 명시적으로 지정되지 않은 AIO 및this 매개변수를 사용하는 경우 기본값은 500 입니다.

2. 이전 설명을 기반으로 스토리지 장치에 필요한 대로 지속성 구성을 조정합니다.

추가 리소스

- 저널 기반 지속성 구성에 사용할 수 있는 모든 매개변수에 대한 자세한 내용은 부록 E. 메시징 저널 구성 Cryostat 을 참조하십시오.

6.1.3. 바인딩 저널 정보

바인딩 저널은 브로커 및 해당 속성에 배포된 대기열 세트와 같은 바인딩 관련 데이터를 저장하는 데 사용됩니다. ID 시퀀스 카운터와 같은 데이터도 저장합니다.

바인딩 저널은 일반적으로 메시지 저널에 비해 처리량이 낮기 때문에 항상 NIO를 사용합니다. 이 저널의 파일은 activemq-bindings 접두사가 붙습니다. 각 파일에는 .bindings 의 확장자와 기본 크기가 1048576바이트가 있습니다.

바인딩 저널을 구성하려면 < broker_instance_dir> /etc/broker.xml 구성 파일의 core 요소에 다음 매개변수를 포함합니다.

bindings-directory

바인딩 저널의 디렉터리입니다. 기본값은 < broker_instance_dir>/data/bindings 입니다.

create-bindings-dir

이 매개변수의 값을 true 로 설정하면 브로커가 바인딩 디렉터리에 지정된 위치에 바인딩 디렉터리를 자동으로 생성합니다(아직 없는 경우). 기본값은 true입니다.

6.1.4. JMS 저널 정보

JMS 저널은 JMS 대기열, 주제, 연결 팩토리를 비롯한 모든 JMS 관련 데이터와 이러한 리소스에 대한 JNDI 바인딩을 저장합니다. 관리 API를 통해 생성된 모든 JMS 리소스는 이 저널에 유지되지만 구성 파일을 통해 구성된 리소스는 유지되지 않습니다. 브로커는 JMS가 사용되는 경우에만 JMS 저널을 생성합니다.

JMS 저널의 파일 앞에 activemq-jms 가 붙습니다. 각 파일에는 .jms 의 확장자와 기본 크기 1048576 바이트도 있습니다.

JMS 저널은 바인딩 저널과 구성을 공유합니다.

추가 리소스

- 바인딩 저널에 대한 자세한 내용은 6.1.3절. “바인딩 저널 정보” 을 참조하십시오.

6.1.5. 저널 보존 구성

생성된 각 저널 파일의 사본을 유지하도록 AMQ Broker를 구성할 수 있습니다. 저널 보존을 구성한 후 저널 파일 복사본에서 메시지를 재생하여 브로커로 메시지를 보낼 수 있습니다.

6.1.5.1. 저널 보존 구성

특정 기간 동안 또는 스토리지 제한에 도달하거나 둘 다에 도달할 때까지 저널 파일의 사본을 유지하도록 AMQ Broker를 구성할 수 있습니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. core 요소 내에서 journal-retention-directory 특성을 추가합니다. 저널 파일의 보존을 제어하려면 마침표 또는 스토리지 제한을 지정합니다. 또한 저널 파일 사본의 파일 시스템 위치를 지정합니다. 다음 예제에서는 AMQ Broker를 구성하여 저널 파일 복사본을 7 일 동안 또는 파일에 10GB 의 스토리지를 사용할 때까지 데이터/ 유지 관리 디렉토리에 보관합니다.

```
<configuration>
 <core>
  ...
  <journal-retention-directory period="7" unit="DAYS" storage-limit="10G">data/retention</journal-retention-directory>
  ...
 </core>
</configuration>
```

기간

저널 파일 사본을 보관하기 위한 기간입니다. 시간 기간이 만료되면 AMQ Broker는 지정된 기간

보다 오래된 파일을 제거합니다.

단위

보존 기간에 적용할 측정값 단위입니다. 기본값은 **DAYS** 입니다. 다른 유효한 값은 **HOURS,MINUTES** 및 **SECONDS** 입니다.

디렉터리

저널 파일 복사의 파일 시스템 위치입니다. 지정된 디렉터리는 **< broker_instance_dir>** 디렉터리를 기준으로 합니다.

storage-limit

모든 저널 파일 사본에서 사용할 수 있는 최대 스토리지입니다. 스토리지 제한에 도달하면 브로커는 가장 오래된 저널 파일을 제거하여 새 저널 파일 사본의 공간을 제공합니다. 스토리지 제한을 설정하는 것은 저널 파일 보존으로 인해 브로커가 디스크 공간이 부족하고 종료되지 않도록 하는 효과적인 방법입니다.

6.1.5.2. 브로커에 존재하는 주소에 대한 저널 파일 사본의 메시지 재생

저널 파일 사본에서 재생하려는 메시지의 주소가 **AMQ Broker**에 존재하는 경우 다음 절차를 사용하여 메시지를 재생합니다. 브로커의 원래 주소 또는 다른 주소로 메시지를 재생할 수 있습니다.

프로세스

1. **AMQ** 관리 콘솔에 로그인합니다. 자세한 내용은 *AMQ 관리 콘솔* 액세스를 참조하십시오.
2. 메인 메뉴에서 **Artemis** 를 클릭합니다.
3. 폴더 트리에서 주소를 클릭하여 주소 목록을 표시합니다.
4. 주소 탭을 클릭합니다.
5. 메시지를 재생하려는 주소의 **Action** 열에서 작업을 클릭합니다.
6. 재생 작업을 선택합니다.

재생 작업에서 모든 저널 파일 복사본에서 재생할 메시지를 검색하려면 재생(문자열, 문자열) 작업을 클릭합니다.

- 재생 작업에서 특정 기간 내에 생성된 저널 파일 복사본에서만 재생할 메시지를 검색하려면 재생(문자열, 문자열, 문자열, 문자열) 작업을 선택합니다. **startScanDate** 및 **endScanDate** 필드에서 기간을 지정합니다.

7. 재생 옵션을 지정합니다.

   a. 대상 필드에서 재생된 메시지를 보낼 브로커의 주소를 지정합니다. 이 필드를 비워 두면 브로커의 원래 주소로 메시지를 재생합니다.

   b. (선택 사항) 필터 필드에서 필터 문자열과 일치하는 메시지만 재생할 문자열을 지정합니다. 예를 들어 메시지에 **storeID** 속성이 있는 경우 **storeID="1000"** 필터를 사용하여 저장소 **ID** 값이 **1000**인 모든 메시지를 재생할 수 있습니다. 필터를 지정하지 않으면 스캔한 저널 파일 복사본의 모든 메시지가 **AMQ Broker**로 재생됩니다.

8. 실행을 클릭합니다.

추가 리소스

- **AMQ Management Console** 사용에 대한 자세한 내용은 **AMQ** 관리 콘솔 사용을 참조하십시오.

6.1.5.3. 브로커에서 제거된 주소의 저널 파일 사본에서 메시지 재생

저널 파일 사본에서 재생하려는 메시지의 주소가 **AMQ Broker**에서 제거된 경우 다음 절차를 사용하여 브로커의 다른 주소로 메시지를 재생합니다.

프로세스

1. **AMQ** 관리 콘솔에 로그인합니다. 자세한 내용은 **AMQ** 관리 콘솔 액세스를 참조하십시오.

2. 메인 메뉴에서 **Artemis** 를 클릭합니다.

3. 폴더 트리에서 최상위 서버를 클릭합니다.

4. **Operations** 탭을 클릭합니다.

5. 재생 작업을 선택합니다.

   - 재생 작업에서 모든 저널 파일 복사본에서 재생할 메시지를 검색하려면 재생 **(String,String,String)** 작업을 클릭합니다.

   - 재생 작업에서 특정 기간 내에 생성된 저널 파일 복사본에서만 재생되도록 하려면 재생 **(String,String, string,String,String,String)** 작업을 선택합니다. **startScanDate** 및 **endScanDate** 필드에서 기간을 지정합니다.

6. 재생 옵션을 지정합니다.

   a. 주소 필드에서 재생할 메시지의 주소를 지정합니다.

   b. 대상 필드에서 재생된 메시지를 보낼 브로커의 주소를 지정합니다.

   c. (선택 사항) 필터 필드에서 필터 문자열과 일치하는 메시지만 재생할 문자열을 지정합니다. 예를 들어 메시지에 **storeID** 속성이 있는 경우 **storeID="1000"** 필터를 사용하여 저장소 **ID** 값이 **1000**인 모든 메시지를 재생할 수 있습니다. 필터를 지정하지 않으면 스캔한 저널 파일 복사본의 모든 메시지가 **AMQ Broker**로 재생됩니다.

7. 실행을 클릭합니다.

추가 리소스

- **AMQ Management Console** 사용에 대한 자세한 내용은 **AMQ** 관리 콘솔 사용을 참조하십시오.

6.1.6. 저널 파일 압축

AMQ Broker에는 저널에서 dead space를 제거하고 디스크 공간을 줄일 수 있도록 데이터를 압축하는 압축 알고리즘이 포함되어 있습니다.

다음 하위 섹션에서는 다음을 수행하는 방법을 보여줍니다.

- 특정 기준이 충족될 때 자동으로 컴팩트 저널 파일을 사용하도록 브로커 구성
- 명령줄 인터페이스에서 압축 프로세스를 수동으로 실행

6.1.6.1. 저널 파일 압축 구성

브로커는 다음 기준을 사용하여 압축 작업을 시작할 시기를 결정합니다.

- 저널에 대해 생성된 파일 수입니다.
- 저널 파일에 있는 라이브 데이터의 백분율입니다.

이러한 기준 모두에 대해 구성된 값에 도달한 후 압축 프로세스는 저널을 구문 분석하고 모든 종료된 레코드를 제거합니다. 결과적으로 저널은 더 적은 파일로 구성됩니다.

다음 절차에서는 저널 파일 압축을 위해 브로커를 구성하는 방법을 보여줍니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.
2. core 요소 내에서 journal-compact-min-files 및 journal-compact-percentage 매개변수를 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<configuration>
 <core>
  ...
  <journal-compact-min-files>15</journal-compact-min-files>
  <journal-compact-percentage>25</journal-compact-percentage>
```

```
   ...
  </core>
</configuration>
```

journal-compact-min-files

압축이 시작되기 전에 브로커가 생성해야 하는 최소 저널 파일 수입니다. 기본값은 **10** 입니다. 값을 **0** 으로 설정하면 압축이 비활성화됩니다. 저널의 크기가 무기한 증가할 수 있으므로 압축 기능을 비활성화할 때 주의해야 합니다.

journal-compact-percentage

저널 파일에 있는 라이브 데이터의 백분율입니다. 이 백분율보다 적은 수의 라이브 데이터(및 **journal-compact-min-files** 의 구성된 값도 도달한 경우) 압축이 시작됩니다. 기본값은 **30** 입니다.

6.1.6.2. 명령줄 인터페이스에서 압축 실행

다음 절차에서는 **CLI(**명령줄 인터페이스)를 사용하여 저널 파일을 압축하는 방법을 보여줍니다.

프로세스

1. **< broker_instance_dir>** 디렉터리의 소유자로서 브로커를 중지합니다. 아래 예제에서는 사용자 **amq-broker** 를 보여줍니다.

```
su - amq-broker
cd <broker_instance_dir>/bin
$ ./artemis stop
```

2. (선택 사항) 다음 **CLI** 명령을 실행하여 데이터 툴의 전체 매개변수 목록을 가져옵니다. 기본적으로 툴은 **< broker_instance_dir> /etc/broker.xml** 에 있는 설정을 사용합니다.

```
$ ./artemis help data compact.
```

3. 다음 **CLI** 명령을 실행하여 데이터를 압축합니다.

```
$ ./artemis data compact.
```

4. 도구가 데이터를 성공적으로 압축한 후 브로커를 다시 시작합니다.

```
$ ./artemis run
```

추가 리소스

- AMQ Broker에는 저널 파일을 관리하기 위한 여러 CLI 명령이 포함되어 있습니다. 자세한 내용은 부록의 명령줄 툴 을 참조하십시오.

6.1.7. 디스크 쓰기 캐시 비활성화

대부분의 디스크에는 하드웨어 쓰기 캐시가 포함되어 있습니다. 쓰기는 나중에 디스크에 지연 기록되므로 쓰기 캐시는 디스크의 명확한 성능을 향상시킬 수 있습니다. 기본적으로 많은 시스템은 디스크 쓰기 캐시가 활성화된 상태로 제공됩니다. 즉, 운영 체제에서 동기화한 후에도 데이터가 실제로 디스크에 생성되었음을 보장하지 않습니다. 따라서 오류가 발생하면 중요한 데이터가 손실될 수 있습니다.

일부 더 많은 디스크에는 장애 발생 시 데이터가 손실되는 것은 아니지만 테스트해야 하는 비휘발성 또는 배터리 지원 쓰기 캐시가 있습니다. 디스크에 이러한 기능이 없는 경우 쓰기 캐시가 비활성화되어 있는지 확인해야 합니다. 디스크 쓰기 캐시를 비활성화하면 성능에 부정적인 영향을 미칠 수 있습니다.

다음 절차에서는 Windows에서 Linux에서 디스크 쓰기 캐시를 비활성화하는 방법을 보여줍니다.

프로세스

1. Linux에서 디스크 쓰기 캐시 설정을 관리하려면 hdparm (IDE 디스크용) 또는 sdparm 또는 sginfo (SDSI/SATA 디스크용)를 사용합니다.

2. Windows에서 디스크 작성기 캐시 설정을 관리하려면 디스크를 마우스 오른쪽 버튼으로 클릭합니다. 속성을 선택합니다.

6.2. 데이터베이스에 메시지 데이터 유지

데이터베이스에서 메시지 데이터를 저장하면 브로커는 JDBC(Java Database Connectivity) 연결을 사용하여 메시지 및 바인딩 데이터를 데이터베이스 테이블에 저장합니다. 테이블의 데이터는 AMQ Broker 저널 인코딩을 사용하여 인코딩됩니다. 지원되는 데이터베이스에 대한 자세한 내용은 Red Hat Customer Portal의 Red Hat AMQ 7 지원 구성 을 참조하십시오.

중요

관리자는 조직의 광범위한 IT 인프라의 요구 사항에 따라 메시지 데이터를 데이터베이스에 저장하도록 선택할 수 있습니다. 그러나 데이터베이스를 사용하면 메시징 시스템의 성능에 부정적인 영향을 미칠 수 있습니다. 특히 JDBC를 통해 데이터베이스 테이블에 메시징 데이터를 작성하면 브로커에 상당한 성능 오버헤드가 발생합니다.

6.2.1. JDBC 지속성 구성

다음 절차에서는 데이터베이스 테이블에 메시지 및 바인딩 데이터를 저장하도록 브로커를 구성하는 방법을 보여줍니다.

프로세스

1. 브로커 런타임에 적절한 JDBC 클라이언트 라이브러리를 추가합니다. 이렇게 하려면 관련 .JAR 파일을 < broker_instance_dir> /lib 디렉터리에 추가합니다.

   .JAR 파일을 다른 디렉토리에 추가하려면 Java classpath에 해당 디렉토리를 추가해야 합니다. 자세한 내용은 다음을 참조하십시오. 1.5절. “JAVA Classpath 확장”

2. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

3. core 요소 내에서 database- store 요소가 포함된 Store 요소를 추가합니다.

```
<configuration>
 <core>
  <store>
   <database-store>
   </database-store>
  </store>
 </core>
</configuration>
```

4. database-store 요소 내에서 JDBC 지속성을 위한 구성 매개변수를 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<configuration>
 <core>
  <store>
   <database-store>
    <jdbc-connection-url>jdbc:oracle:data/oracle/database-store;create=true</jdbc-
connection-url>
    <jdbc-user>ENC(5493dd76567ee5ec269d11823973462f)</jdbc-user>
    <jdbc-password>ENC(56a0db3b71043054269d11823973462f)</jdbc-password>
    <bindings-table-name>BIND_TABLE</bindings-table-name>
    <message-table-name>MSG_TABLE</message-table-name>
    <large-message-table-name>LGE_TABLE</large-message-table-name>
    <page-store-table-name>PAGE_TABLE</page-store-table-name>
    <node-manager-store-table-name>NODE_TABLE</node-manager-store-table-name>
    <jdbc-driver-class-name>oracle.jdbc.driver.OracleDriver</jdbc-driver-class-name>
    <jdbc-network-timeout>10000</jdbc-network-timeout>
```

```
        <jdbc-lock-renew-period>2000</jdbc-lock-renew-period>
        <jdbc-lock-expiration>20000</jdbc-lock-expiration>
        <jdbc-journal-sync-period>5</jdbc-journal-sync-period>
        <jdbc-max-page-size-bytes>100K</jdbc-max-page-size-bytes>
      </database-store>
    </store>
  </core>
</configuration>
```

jdbc-connection-url

데이터베이스 서버의 전체 JDBC 연결 URL입니다. 연결 URL에는 모든 구성 매개변수와 데이터베이스 이름이 포함되어야 합니다.

jdbc-user

데이터베이스 서버의 암호화된 사용자 이름입니다. 구성 파일에 사용할 사용자 이름 및 암호를 암호화하는 방법에 대한 자세한 내용은 5.9절. “구성 파일에서 암호 암호화”을 참조하십시오.

jdbc-password

데이터베이스 서버의 암호화된 암호입니다. 구성 파일에 사용할 사용자 이름 및 암호를 암호화하는 방법에 대한 자세한 내용은 5.9절. “구성 파일에서 암호 암호화”을 참조하십시오.

bindings-table-name

바인딩 데이터가 저장되는 테이블의 이름입니다. 테이블 이름을 지정하면 간섭 없이 여러 서버 간에 단일 데이터베이스를 공유할 수 있습니다.

message-table-name

메시지 데이터가 저장되는 테이블의 이름입니다. 이 테이블 이름을 지정하면 간섭 없이 여러 서버 간에 단일 데이터베이스를 공유할 수 있습니다.

large-message-table-name

큰 메시지 및 관련 데이터가 유지되는 테이블의 이름입니다. 또한 클라이언트가 청크에서 큰 메시지를 스트리밍하면 청크가 이 테이블에 저장됩니다. 이 테이블 이름을 지정하면 간섭 없이 여러 서버 간에 단일 데이터베이스를 공유할 수 있습니다.

page-store-table-name

페이지 저장된 저장소 디렉터리 정보가 저장되는 테이블의 이름입니다. 이 테이블 이름을 지정하면 간섭 없이 여러 서버 간에 단일 데이터베이스를 공유할 수 있습니다.

node-manager-store-table-name

라이브 및 백업 브로커에 대한 공유 저장소 HA(고가용성) 잠금 및 기타 HA 관련 데이터가 브로커 서버에 저장되는 테이블의 이름입니다. 이 테이블 이름을 지정하면 간섭 없이 여러 서버 간에 단일 데이터베이스를 공유할 수 있습니다. 공유 저장소 HA를 사용하는 각 라이브

백업 쌍은 동일한 테이블 이름을 사용해야 합니다. 동일한 테이블을 여러 개(및 관련이 없는) 라이브 백업 쌍 간에 공유할 수 없습니다.

jdbc-driver-class-name

JDBC 데이터베이스 드라이버의 정규화된 클래스 이름입니다. 지원되는 데이터베이스에 대한 자세한 내용은 Red Hat Customer Portal의 Red Hat AMQ 7 지원 구성 을 참조하십시오.

jdbc-network-timeout

JDBC 네트워크 연결 제한 시간(밀리초)입니다. 기본값은 20000밀리초입니다. 공유 저장소 HA에 JDBC를 사용하는 경우 시간 제한을 jdbc-lock-expiration 보다 작거나 같은 값으로 설정하는 것이 좋습니다.

jdbc-lock-renew-period

현재 JDBC 잠금에 대한 갱신 기간의 길이(밀리초)입니다. 이 시간이 지나면 브로커는 잠금을 갱신할 수 있습니다. jdbc-lock-expiration 값보다 작은 값을 설정하는 것이 좋습니다. 이를 통해 브로커는 리스를 연장할 수 있는 충분한 시간을 제공하고 브로커에게 연결 문제 발생 시 잠금을 갱신할 수 있는 시간을 제공합니다. 기본값은 2000밀리초입니다.

jdbc-lock-expiration

jdbc-lock-renew-period 값이 경과하더라도 현재 JDBC 잠금이 보유(즉, 획득 또는 갱신됨)로 간주되는 시간(밀리초)입니다.

브로커는 jdbc-lock-renew-period 의 값에 따라 소유되는 잠금을 주기적으로 갱신하려고 합니다. 브로커가 잠금을 갱신 하지 못하는 경우(예: 연결 문제로 인해) 브로커는 잠금을 성공적으로 취득하거나 갱신한 이후 jdbc-lock-expiration 값이 전달될 때까지 잠금을 갱신하려고 합니다.

위에서 설명한 갱신 동작의 예외는 다른 브로커가 잠금을 취득하는 경우입니다. 이는 DBMS(데이터베이스 관리 시스템)와 브로커 간에 시간 불일치가 있거나 가비지 수집에 대한 일시 중지가 긴 경우 발생할 수 있습니다. 이 경우 원래 잠금을 보유한 브로커는 잠금을 손실하고 갱신하려고 시도하지 않습니다.

만료 시간이 경과하면 JDBC 잠금이 현재 소유하고 있는 브로커에 의해 갱신되지 않은 경우 다른 브로커는 JDBC 잠금을 설정할 수 있습니다.

jdbc-lock-expiration 의 기본값은 20000밀리초입니다.

jdbc-journal-sync-period

브로커 저널이 JDBC와 동기화되는 기간(밀리초)입니다. 기본값은 5밀리초입니다.

jdbc-max-page-size-bytes

AMQ Broker가 JDBC 데이터베이스에 메시지를 저장할 때 각 페이지 파일의 최대 크기(바이트)입니다. 기본값은 102400 이며, 이는 100KB입니다. 지정한 값은 "K" "MB" 및 "GB"와 같은 바이트 표기법도 지원합니다.

6.2.2. JDBC 연결 풀링 구성

JDBC 지속성을 위해 브로커를 구성한 경우 브로커는 JDBC 연결을 사용하여 메시지 및 바인딩 데이터를 데이터베이스 테이블에 저장합니다.

JDBC 연결 오류가 발생하고 오류가 발생할 때 데이터베이스 읽기 또는 쓰기와 같은 활성 연결 활동(예: 데이터베이스 읽기 또는 쓰기)이 없는 경우 브로커는 계속 실행되고 데이터베이스 연결을 다시 설정하려고 합니다. 이를 위해 AMQ Broker는 JDBC 연결 풀링 을 사용합니다.

일반적으로 연결 풀은 여러 애플리케이션 간에 공유할 수 있는 지정된 데이터베이스에 대한 열린 연결 세트를 제공합니다. 브로커의 경우 브로커와 데이터베이스 간의 연결이 실패하면 브로커는 풀과 다른 연결을 사용하여 데이터베이스에 다시 연결을 시도합니다. 풀은 브로커가 수신하기 전에 새 연결을 테스트합니다.

다음 예제에서는 JDBC 연결 풀링을 구성하는 방법을 보여줍니다.

중요

JDBC 연결 풀링을 명시적으로 구성하지 않으면 브로커는 기본 구성과 연결 풀링을 사용합니다. 기본 구성에서는 기존 JDBC 구성의 값을 사용합니다. 자세한 내용은 기본 연결 풀링 구성 을 참조하십시오.

사전 요구 사항

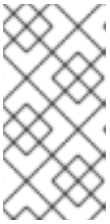
- 이 예제에서는 JDBC 지속성을 구성하기 위한 예제에 빌드됩니다. 참조 6.2.1절. “JDBC 지속성 구성”
- 연결 풀링을 활성화하기 위해 AMQ Broker는 Apache Commons DBCP 패키지를 사용합니다. 브로커에 대한 JDBC 연결 풀링을 구성하기 전에 이 패키지에서 제공하는 내용에 대해 잘 알고 있어야 합니다. 자세한 내용은 다음을 참조하십시오.

- *Apache Commons DBCP 개요*
- *Apache Commons DBCP 구성 매개변수*

프로세스

1. *&lt ;broker-instance-dir> /etc/broker.xml 구성 파일을 엽니다.*

2. *이전에 JDBC 구성에 추가한 database-store 요소 내에서 jdbc-driver-class-name,jdbc-connection-url,jdbc-user,jdbc-password, 매개변수를 제거합니다. 이 절차의 뒷부분에서 해당 DBCP 구성 매개변수로 교체합니다.*



참고

이전 매개변수를 명시적으로 제거하지 않으면 이 절차의 뒷부분에서 추가하는 해당 DBCP 매개변수가 우선합니다.

3. *database-store 요소 내에서 data-source-properties 요소를 추가합니다. 예를 들면 다음과 같습니다.*

```
<store>
   <database-store>
      <data-source-properties>
      </data-source-properties>
      <bindings-table-name>BINDINGS</bindings-table-name>
      <message-table-name>MESSAGES</message-table-name>
      <large-message-table-name>LARGE_MESSAGES</large-message-table-name>
      <page-store-table-name>PAGE_STORE</page-store-table-name>
      <node-manager-store-table-name>NODE_MANAGER_STORE</node-manager-store-
table-name>
      <jdbc-network-timeout>10000</jdbc-network-timeout>
      <jdbc-lock-renew-period>2000</jdbc-lock-renew-period>
      <jdbc-lock-expiration>20000</jdbc-lock-expiration>
      <jdbc-journal-sync-period>5</jdbc-journal-sync-period>
   </database-store>
</store>
```

4. *새로운 data-source-properties 요소 내에서 연결 풀링을 위한 DBCP 데이터 소스 속성을 추가합니다. 키-값 쌍을 지정합니다. 예를 들면 다음과 같습니다.*

```
<store>
```

```
   <database-store>
      <data-source-properties>
         <data-source-property key="driverClassName" value="com.mysql.jdbc.Driver" />
         <data-source-property key="url" value="jdbc:mysql://localhost:3306/artemis" />
         <data-source-property key="username"
value="ENC(5493dd76567ee5ec269d1182397346f)"/>
         <data-source-property key="password"
value="ENC(56a0db3b71043054269d1182397346f)"/>
         <data-source-property key="poolPreparedStatements" value="true" />
         <data-source-property key="maxTotal" value="-1" />
      </data-source-properties>
      <bindings-table-name>BINDINGS</bindings-table-name>
      <message-table-name>MESSAGES</message-table-name>
      <large-message-table-name>LARGE_MESSAGES</large-message-table-name>
      <page-store-table-name>PAGE_STORE</page-store-table-name>
      <node-manager-store-table-name>NODE_MANAGER_STORE</node-manager-store-
table-name>
      <jdbc-network-timeout>10000</jdbc-network-timeout>
      <jdbc-lock-renew-period>2000</jdbc-lock-renew-period>
      <jdbc-lock-expiration>20000</jdbc-lock-expiration>
      <jdbc-journal-sync-period>5</jdbc-journal-sync-period>
   </database-store>
</store>
```

driverClassName

JDBC 데이터베이스 드라이버의 정규화된 클래스 이름입니다.

url

데이터베이스 서버의 전체 JDBC 연결 URL입니다.

사용자 이름

데이터베이스 서버의 암호화된 사용자 이름입니다. 이 값을 암호화되지 않은 일반 텍스트로 지정할 수도 있습니다. 구성 파일에 사용할 사용자 이름 및 암호를 암호화하는 방법에 대한 자세한 내용은 5.9절. "구성 파일에서 암호 암호화" 을 참조하십시오.

암호

데이터베이스 서버의 암호화된 암호입니다. 이 값을 암호화되지 않은 일반 텍스트로 지정할 수도 있습니다. 구성 파일에 사용할 사용자 이름 및 암호를 암호화하는 방법에 대한 자세한 내용은 5.9절. "구성 파일에서 암호 암호화" 을 참조하십시오.

poolPreparedStatements

이 매개 변수의 값을 true 로 설정하면 풀에 캐시된 준비된 문 수가 무제한일 수 있습니다. 이렇게 하면 초기화 비용이 줄어듭니다.

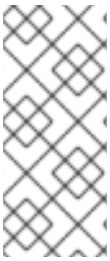
maxTotal

*풀의 최대 연결 수입니다. 이 매개 변수의 값을 **-1** 로 설정하면 제한이 없습니다.*

JDBC 연결 풀링을 명시적으로 구성하지 않으면 브로커는 기본 구성과 연결 풀링을 사용합니다. 기본 구성은 표에 설명되어 있습니다.

표 6.1. 기본 연결 풀링 구성

DBCP 구성 매개변수	기본값
driverClassName	기존 **jdbc-driver-class-name** 매개변수의 값
url	기존 **jdbc-connection-url** 매개변수의 값
사용자 이름	기존 **jdbc-user** 매개변수의 값
암호	기존 **jdbc-password** 매개변수의 값
poolPreparedStatements	**true**
maxTotal	**-1**



참고

재연결은 클라이언트가 브로커에 적극적으로 메시지를 보내지 않는 경우에만 작동합니다. 다시 연결하는 동안 데이터베이스 테이블에 쓰기 시도가 있으면 브로커가 실패하고 종료됩니다.

추가 리소스

- *AMQ Broker에서 지원하는 데이터베이스에 대한 자세한 내용은 Red Hat Customer Portal 의 Red Hat AMQ 7 지원 구성 을 참조하십시오.*
- *Apache Commons DBCP 패키지에서 사용할 수 있는 모든 구성 옵션에 대해 알아보려면 Apache Commons DBCP 구성 매개 변수 를 참조하십시오.*

6.3. 지속성 비활성화

경우에 따라 메시징 시스템이 데이터를 저장하지 않는 것이 요구될 수 있습니다. 이러한 상황에서는 브로커에서 지속성을 비활성화할 수 있습니다.

다음 절차에서는 지속성을 비활성화하는 방법을 보여줍니다.

프로세스

1. <broker_instance_dir>/etc/broker.xml 구성 파일을 엽니다.

2. core 요소 내에서 persistence-enabled 매개변수의 값을 false 로 설정합니다.

```
<configuration>
 <core>
  ...
  <persistence-enabled>false</persistence-enabled>
  ...
 </core>
</configuration>
```

메시지 데이터, 바인딩 데이터, 대용량 메시지 데이터, 중복 ID 캐시 또는 페이징 데이터는 유지됩니다.

7장. 주소에 대한 메모리 사용량 구성

AMQ Broker는 브로커가 제한된 메모리로 실행되는 경우에도 수백만 개의 메시지가 포함된 대규모 큐를 투명하게 지원합니다.

이러한 상황에서는 한 번에 모든 큐를 메모리에 저장할 수 없습니다. 과도한 메모리 소비로부터 보호하기 위해 브로커의 각 주소에 허용되는 최대 메모리 사용량을 구성할 수 있습니다. 또한 주소의 메모리 사용량이 구성된 제한에 도달하면 다음 작업 중 하나를 수행하도록 브로커를 구성할 수 있습니다.

- 페이지 메시지
- 자동으로 메시지 삭제
- 메시지 삭제 및 전송 클라이언트 알림
- 클라이언트에서 메시지를 전송하지 못하도록 차단

주소의 최대 메모리 사용량에 도달할 때 브로커를 페이지 메시지로 구성하는 경우 특정 주소에 대한 제한을 다음과 같이 구성할 수 있습니다.

- 들어오는 메시지를 페이징하는 데 사용되는 디스크 공간 제한
- 클라이언트가 메시지를 사용할 준비가 되면 브로커가 디스크에서 메모리로 전송하는 페이징된 메시지에 사용되는 메모리를 제한합니다.

디스크 사용량 임계값을 설정하여 구성된 모든 페이징 제한을 덮어쓸 수도 있습니다. 디스크 사용량 임계값에 도달하면 브로커는 페이징을 중지하고 들어오는 모든 메시지를 차단합니다.

중요

트랜잭션을 사용할 때 브로커는 트랜잭션 일관성을 보장하기 위해 추가 메모리를 할당할 수 있습니다. 이 경우 브로커가 보고한 메모리 사용량은 메모리에 사용되는 총 바이트 수를 반영하지 못할 수 있습니다. 따라서 지정된 최대 메모리 사용량을 기반으로 브로커를 **page**, **drop** 또는 **block** 메시지로 구성하는 경우 트랜잭션을 사용해서는 안 됩니다.

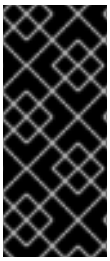
7.1. 메시지 페이징 구성

최대 메모리 사용량 제한이 지정된 주소의 경우 해당 사용량 제한에 도달할 때 브로커가 수행하는 작업을 지정할 수도 있습니다. 구성할 수 있는 옵션 중 하나는 페이징 입니다.

페이징 옵션을 구성하는 경우 주소의 최대 크기에 도달하면 브로커는 해당 주소에 대한 메시지를 페이지 파일이라는 파일에 저장하기 시작합니다. 각 페이지 파일에는 구성할 수 있는 최대 크기가 있습니다. 이러한 방식으로 구성하는 각 주소에는 페이지된 메시지를 저장할 파일 시스템에 전용 폴더가 있습니다.

큐 브라우저와 소비자 모두 큐의 메시지를 검사할 때 페이지 파일을 탐색할 수 있습니다. 그러나 매우 구체적인 필터를 사용하는 소비자는 큐의 기존 메시지를 먼저 사용할 때까지 페이지 파일에 저장된 메시지를 사용하지 못할 수 있습니다. 예를 들어 소비자 필터에 **"color='red'"** 와 같은 문자열 표현식이 포함되어 있다고 가정합니다. **If a message that meets this condition follows one million messages with the property "color='blue', the consumer cannot consume the message until those with those with " color='blue' s have been consumed first.**

클라이언트가 사용할 준비가 되면 브로커 전송(즉, 해제됨) 메시지를 디스크에서 메모리로 전송합니다. 해당 파일의 모든 메시지가 승인되면 브로커가 디스크에서 페이지 파일을 제거합니다.



중요

AMQ Broker는 **JMS** 메시지 우선 순위에 따라 큐에 보류 중인 메시지를 주문합니다. 그러나 호출되는 메시지는 우선 순위에 따라 정렬되지 않습니다. 순서를 유지하려면 페이징을 구성하고 브로커 크기를 충분히 구성하지 말고 모든 메시지를 메모리에 유지합니다.

7.1.1. 페이징 디렉터리 지정

다음 절차에서는 페이징 디렉터리의 위치를 지정하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **core** 요소 내에서 **paging-directory** 요소를 추가합니다. 파일 시스템에서 페이징 디렉터리의 위치를 지정합니다.

```
<configuration ...>
  <core ...>
    ...
```

```
    <paging-directory>/path/to/paging-directory</paging-directory>
    ...
  </core>
</configuration>
```

이후에 페이징을 구성할 때마다 브로커는 지정한 페이징 디렉터리에 전용 디렉터리를 추가합니다.

7.1.2. 페이징을 위한 주소 구성

다음 절차에서는 페이징을 위해 주소를 구성하는 방법을 보여줍니다.

사전 요구 사항

- 주소 및 주소 설정을 구성하는 방법에 대해 잘 알고 있어야 합니다. 자세한 내용은 4장. 주소 및 큐 구성의 내용을 참조하십시오.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. 일치하는 주소 또는 주소 집합에 대해 구성한 address-setting 요소의 경우 구성 요소를 추가하여 최대 메모리 사용량을 지정하고 페이징 동작을 정의합니다. 예를 들면 다음과 같습니다.

```
<address-settings>
  <address-setting match="my.paged.address">
    ...
    <max-size-bytes>104857600</max-size-bytes>
    <max-size-messages>20000</max-size-messages>
    <page-size-bytes>10485760</page-size-bytes>
    <address-full-policy>PAGE</address-full-policy>
    ...
  </address-setting>
</address-settings>
```

max-size-bytes

브로커가 address-full-policy 특성에 지정된 작업을 실행하기 전에 주소에 허용되는 메모리의 최대 크기(바이트)입니다. 기본값은 -1 이며 이는 제한이 없음을 의미합니다. 지정한 값은 "K", "MB" 및 "GB"와 같은 바이트 표기법도 지원합니다.

max-size-messages

브로커가 address-full-policy 특성에 지정된 작업을 실행하기 전에 주소에 허용되는 최

대 메시지 수입니다. 기본값은 **-1**이며, 이는 메시지 제한이 없음을 의미합니다.

page-size-bytes

페이징 시스템에서 사용되는 각 페이지 파일의 크기(바이트)입니다. 기본값은 **10485760** (즉, **10MiB**)입니다. 지정한 값은 **"K"**, **"MB"** 및 **"GB"**와 같은 바이트 표기법도 지원합니다.

address-full-policy

주소의 최대 크기에 도달하면 브로커가 사용하는 작업입니다. 기본값은 **PAGE** 입니다. 유효한 값은 다음과 같습니다.

PAGE

브로커는 추가 메시지를 디스크에 페이지합니다.

DROP

브로커는 추가 메시지를 자동으로 삭제합니다.

FAIL

브로커는 추가 메시지를 삭제하고 클라이언트 메시지 생산자에 예외를 발행합니다.

블록

클라이언트 메시지 생산자는 추가 메시지를 전송하려고 할 때 차단됩니다.

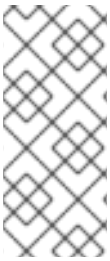
max-size- bytes 및 **max-size -message** 속성에 대한 제한을 설정하면 브로커는 제한에 도달할 때 **address-full-policy** 특성에 지정된 작업을 실행합니다. 이전 예제의 구성에서 브로커는 메모리의 전체 메시지가 **20,000**를 초과하거나 사용 가능한 메모리의 **104857600** 바이트를 사용하는 경우 **my.paged.address** 주소에 대한 페이징 메시지를 시작합니다.

이전 예에 표시되지 않는 추가 페이징 구성 요소는 다음과 같습니다.

page-sync-timeout

주기적인 페이지 동기화 사이의 시간(나노초)입니다. 비동기 **IO** 저널(즉, **journal-type** 이 **broker.xml** 구성 파일에서 **ASYNCIO** 로 설정됨)을 사용하는 경우 기본값은 **3333333** 입니다. 표준 **Java NIO** 저널(즉, **journal-type** 은 **NIO**로 설정됨)을 사용하는 경우 기본값은 **journal-buffer-timeout** 매개변수의 구성된 값입니다.

이전 예에서 **my.paged.address** 주소로 전송된 메시지가 메모리에서 **104857600**바이트를 초과하면 브로커는 페이징을 시작합니다.



참고

address-setting 요소에 **max-size-bytes** 를 지정하는 경우 값은 일치하는 각 주소에 적용됩니다. 이 값을 지정하면 일치하는 모든 주소의 총 크기가 **max-size-bytes** 값으로 제한되는 것은 아닙니다.

7.1.3. 글로벌 페이징 크기 구성

경우에 따라 주소당 메모리 제한을 구성하는 것은 실용적이지 않습니다(예: 브로커가 사용 패턴이 다른 많은 주소를 관리하는 경우). 이러한 경우 글로벌 메모리 제한을 지정할 수 있습니다. 글로벌 제한은 브로커가 모든 주소에 사용할 수 있는 총 메모리 양입니다. 이 메모리 제한에 도달하면 브로커는 새로 들어오는 각 메시지와 연결된 주소에 대해 **address-full-policy** 특성에 지정된 작업을 실행합니다.

다음 절차에서는 글로벌 페이징 크기를 구성하는 방법을 보여줍니다.

사전 요구 사항

- 페이징을 위해 주소를 구성하는 방법에 대해 잘 알고 있어야 합니다. 자세한 내용은 7.1.2절. “페이징을 위한 주소 구성”의 내용을 참조하십시오.

프로세스

1. 브로커를 중지합니다.

   a. **Linux**의 경우:

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. **Windows**에서:

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

3. **core** 요소 내에서 **global-max-size** 요소를 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<configuration>
 <core>
  ...
  <global-max-size>1GB</global-max-size>
  <global-max-messages>900000</global-max-messages>
  ...
 </core>
</configuration>
```

global-max-size

브로커가 모든 주소에 사용할 수 있는 총 메모리 양(바이트)입니다. 이 제한에 도달하면 브로커는 각 수신 메시지와 연결된 주소에 대해 **address-full-policy** 특성에 지정된 작업을 실행합니다. 브로커를 호스팅하는 **JVM(Java** 가상 머신)에서 사용할 수 있는 최대 메모리의 기본값은 **global-max-size** 의 절반입니다.

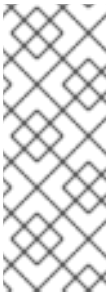
global-max-size 의 값은 바이트 단위이지만 바이트 표기법(예: **"K", "Mb", "GB")**도 지원합니다.

이전 예에서 브로커는 메시지를 처리할 때 최대 **1GB**의 사용 가능한 메모리를 사용하도록 구성되어 있습니다.

global-max-messages

모든 주소에 허용되는 총 메시지 수입니다. 이 제한에 도달하면 브로커는 각 수신 메시지와 연결된 주소에 대해 **address-full-policy** 특성에 지정된 작업을 실행합니다. 기본값은 **-1** 이며, 이는 메시지 제한이 없음을 의미합니다.

global-max- size 및 **global-max -messages** 속성에 대한 제한을 설정하면 브로커는 제한에 도달할 때 **address-full-policy** 특성에 지정된 작업을 실행합니다. 이전 예제의 구성에서 브로커는 메모리의 메시지 수가 **900,000**를 초과하거나 **1GB**의 사용 가능한 메모리를 사용하는 경우 모든 주소에 대한 페이징 메시지를 시작합니다.



참고

global-max-size 또는 **global-max-messages** 속성에 설정된 제한 전에 **max-size- bytes** 또는 **max-size-messages** 속성을 사용하여 개별 주소에 설정된 제한에 도달하면 브로커는 해당 주소에 대해 **address-full-policy** 에 지정된 작업을 실행합니다.

4. 브로커를 시작합니다.

   a. **Linux**의 경우:

   ```
   <broker_instance_dir>/bin/artemis run
   ```

   b. **Windows**에서:

   ```
   <broker_instance_dir>\bin\artemis-service.exe start
   ```

7.1.4. 특정 주소에 대해 페이징하는 동안 디스크 사용량 제한

브로커가 개별 주소 또는 주소 집합에 대해 들어오는 메시지 페이징을 중지하기 전에 브로커가 사용할 수 있는 디스크 공간의 양을 제한할 수 있습니다.

프로세스

1. 브로커를 중지합니다.

   a. **Linux**의 경우:

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. **Windows**에서:

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

3. **core** 요소 내에서 디스크 사용량 또는 메시지 수 또는 둘 다에 따라 페이징 제한을 지정하는 속성을 추가하고 제한에 도달한 경우 수행할 작업을 지정합니다. 예를 들면 다음과 같습니다.

```
<address-settings>
  <address-setting match="match="my.paged.address"">
   ...
   <page-limit-bytes>10G</page-limit-bytes>
```

```
    <page-limit-messages>1000000</page-limit-messages>
    <page-full-policy>FAIL</page-full-policy>
    ...
   </address-setting>
  </address-settings>
```

page-limit-bytes

*브로커가 **page-full-policy** 특성에 지정된 작업을 실행하기 전에 주소에 대해 들어오는 메시지를 페이징하는 데 허용되는 최대 디스크 공간(바이트)입니다. 지정한 값은 **"K", "MB"** 및 **"GB"**와 같은 바이트 표기법을 지원합니다. 기본값은 **-1**이며 이는 제한이 없음을 의미합니다.*

page-limit-messages

*브로커가 **page-full-policy** 특성에 지정된 작업을 실행하기 전에 주소로 페이징할 수 있는 최대 메시지 수입니다. 기본값은 **-1**이며, 이는 메시지 제한이 없음을 의미합니다.*

page-full-policy

*주소에 대해 **page-limit-bytes** 또는 **page-limit-messages** 속성에 설정된 제한에 도달할 때 브로커가 사용하는 작업입니다. 유효한 값은 다음과 같습니다.*

브로커가 추가 메시지를 자동으로 삭제합니다.

***FAIL** 브로커에서 추가 메시지를 삭제하고 전송 클라이언트에 알립니다.*

*이전 예에서 페이징이 **10GB**의 디스크 공간을 사용하거나 총 **100**만 개의 메시지가 표시될 때까지 **my.paged.address** 주소에 대한 브로커 메시지가 표시됩니다.*

4. *브로커를 시작합니다.*

   a. ***Linux**의 경우:*

   ```
   <broker_instance_dir>/bin/artemis run
   ```

   b. ***Windows**에서:*

   ```
   <broker_instance_dir>\bin\artemis-service.exe start
   ```

7.1.5. 메모리로 페이지링된 메시지의 흐름 제어

AMQ Broker가 디스크에 메시지를 페이징하도록 구성된 경우 브로커는 페이지가 지정된 메시지를 읽고 클라이언트가 메시지를 사용할 준비가 되면 메시지를 메모리로 전송합니다. 메시지가 초과 메모리를 사용하지 않도록 하려면 브로커가 디스크에서 메모리로 전송하는 메시지에 대해 각 주소에서 사용하는 메모리를 제한할 수 있습니다.

중요

클라이언트 애플리케이션이 승인 보류 중 메시지를 너무 많이 남겨 두면 브로커는 보류 중인 메시지가 승인될 때까지 페이지가 지정된 메시지를 읽지 않으므로 브로커에서 메시지 중단을 유발할 수 있습니다.

예를 들어, 페이지링된 메시지를 메모리로 전송하는 데 대한 제한이 기본적으로 **20MB**인 경우 브로커는 더 많은 메시지를 읽기 전에 클라이언트에서 승인을 기다립니다. 동시에 클라이언트가 브로커에 승인을 보내기 전에 충분한 메시지를 받기를 기다리는 경우 클라이언트가 사용하는 배치 크기에 따라 결정됩니다.

중단을 방지하기 위해 페이지링된 메시지를 메모리로 전송하거나 메시지 전달 수를 줄이는 브로커 제한을 늘립니다. 메시지를 더 빨리 커밋하거나 브로커에서 더 이상 메시지가 수신되지 않으면 시간 초과를 커밋하고 승인을 커밋하도록 하여 전송 메시지 수를 줄일 수 있습니다.

AMQ Management Console에서 큐의 제공 수 및 **Delivering Cryostats** 메트릭에서 메시지 전달 의 수와 크기를 확인할 수 있습니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 일치하는 주소 또는 주소 집합에 대해 구성한 **address-settings** 요소의 경우 페이지링된 메시지를 메모리로 전송할 수 있는 제한을 지정합니다. 예를 들면 다음과 같습니다.

```
address-settings>
  <address-setting match="my.paged.address">
    ...
    <max-read-page-messages>104857600</max-read-page-messages>
    <max-read-page-bytes>20MB</max-read-page-bytes>
    ...
  </address-setting>
</address-settings>
```

max-read-page-messages 브로커가 주소당 디스크에서 메모리로 읽을 수 있는 최대 페이지 메시지 수입니다. 기본값은 **-1**이며, 이는 제한이 적용되지 않음을 의미합니다.

Max-read-page-bytes 브로커는 주소당 디스크에서 메모리로 읽을 수 있는 페이징된 메시지의 최대 바이트 단위 크기입니다. 기본값은 **20MB**입니다.

이러한 제한을 적용하면 브로커는 메모리에 두 메시지를 모두 계산하여 현재 제공되는 소비자와 메시지에 전달할 준비가 되어 있습니다. 소비자가 메시지를 확인하는 속도가 느리면 메시지를 전달하면 메모리 또는 메시지 제한에 도달하여 브로커가 새 메시지를 메모리에 읽지 못하게 할 수 있습니다. 그 결과 브로커에게 메시지가 표시될 수 있습니다.

3.

소비자가 메시지를 승인하는 속도가 느리고 구성된 **max-read-page-messages** 또는 **max-read-page-bytes** 제한이 현재 전달 중인 메시지에 대해 페이지 지정된 메시지를 메모리에 전송할 때 별도의 제한 사항을 지정합니다. 예를 들면 다음과 같습니다

```
address-settings>
  <address-setting match="my.paged.address">
    ...
    <prefetch-page-bytes>20MB</prefetch-page-bytes>
    <prefetch-page-messages>104857600</prefetch-page-messages>
    ...
  </address-setting>
</address-settings>
```

호출된 메시지를 큐당 메모리로 읽는 데 사용할 수 있는 미리 페이지바이트 메모리(바이트)입니다. 기본값은 **20MB**입니다.

prefetch-page-messages 브로커가 디스크에서 큐당 메모리로 읽을 수 있는 페이지링된 메시지 수입니다. 기본값은 **-1**이며, 이는 제한이 적용되지 않음을 의미합니다.

현재 전달 중인 메시지에서 사용하는 메모리 또는 메시지 수를 제한하기 위해 **prefetch-page-bytes** 또는 **prefetch-page-messages** 매개변수에 대한 제한을 지정하는 경우, 새 메시지를 메모리에 읽을 수 있는 용량을 제공하기 위해 **max-read-page-bytes** 또는 **max-read-page-message** 매개변수에 대한 높은 제한을 설정합니다.

참고

prefetch-page-bytes 매개변수의 값 앞에 **max-read-page-bytes** 매개변수의 값에 도달하면 브로커는 추가 페이지가 지정된 메시지를 메모리로 읽는 것을 중지합니다.

7.1.6. 디스크 사용량 임계값 설정

디스크 사용량 임계값을 설정하면 브로커가 페이징을 중지하고 들어오는 모든 메시지를 차단할 수 있습니다.

프로세스

1. 브로커를 중지합니다.

   a. **Linux**의 경우:

   ```
   <broker_instance_dir>/bin/artemis stop
   ```

   b. **Windows**에서:

   ```
   <broker_instance_dir>\bin\artemis-service.exe stop
   ```

2. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

3. **core** 요소 내에서 **max-disk-usage** 구성 요소를 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

   ```
   <configuration>
    <core>
     ...
     <max-disk-usage>80</max-disk-usage>
     ...
    </core>
   </configuration>
   ```

   max-disk-usage

   브로커가 사용할 수 있는 사용 가능한 디스크 공간의 최대 백분율입니다. 이 제한에 도달하면 브로커는 수신되는 메시지를 차단합니다. 기본값은 **90** 입니다.

   이전 예에서 브로커는 사용 가능한 디스크 공간의 **80%**를 사용하는 것으로 제한됩니다.

4. 브로커를 시작합니다.

a. **Linux의 경우:**

```
<broker_instance_dir>/bin/artemis run
```

b. **Windows에서:**

```
<broker_instance_dir>\bin\artemis-service.exe start
```

7.2. 메시지 삭제 구성

7.1.2절. "페이징을 위한 주소 구성" 페이징할 주소를 구성하는 방법을 표시합니다. 이 절차의 일부로 **address-full-policy** 값을 **PAGE** 로 설정합니다.

주소가 지정된 최대 크기에 도달하면 메시지(페키지 대신)를 삭제하려면 **address-full-policy** 값을 다음 중 하나로 설정합니다.

DROP

지정된 주소의 최대 크기에 도달하면 브로커는 추가 메시지를 자동으로 삭제합니다.

FAIL

지정된 주소의 최대 크기에 도달하면 브로커는 추가 메시지를 삭제하고 생산자에게 예외를 발행합니다.

7.3. 메시지 차단 구성

다음 절차에서는 지정된 주소가 지정한 최대 크기 제한에 도달하면 메시지 차단을 구성하는 방법을 보여줍니다.

참고

Core, OpenWire 및 **AMQP** 프로토콜에 만 대한 메시지 차단을 구성할 수 있습니다.

7.3.1. Core 및 OpenWire 생산자 차단

다음 절차에서는 지정된 주소가 지정한 최대 크기 제한에 도달하면 **Core** 및 **OpenWire** 메시지 생산자에 대한 메시지 차단을 구성하는 방법을 보여줍니다.

사전 요구 사항

- 주소 및 주소 설정을 구성하는 방법에 대해 잘 알고 있어야 합니다. 자세한 내용은 4장. 주소 및 큐 구성의 내용을 참조하십시오.

프로세스

1. <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. 일치하는 주소 또는 주소 집합에 대해 구성한 address-setting 요소의 경우 구성 요소를 추가하여 메시지 차단 동작을 정의합니다. 예를 들면 다음과 같습니다.

```
<address-settings>
  <address-setting match="my.blocking.address">
    ...
    <max-size-bytes>300000</max-size-bytes>
    <address-full-policy>BLOCK</address-full-policy>
    ...
  </address-setting>
</address-settings>
```

max-size-bytes

브로커가 address-full-policy 에 지정된 정책을 실행하기 전에 주소에 허용되는 메모리의 최대 크기(바이트)입니다. 지정한 값은 "K", "MB" 및 "GB"와 같은 바이트 표기법도 지원합니다.

참고

address-setting 요소에 max-size-bytes 를 지정하는 경우 값은 일치하는 각 주소에 적용됩니다. 이 값을 지정하면 일치하는 모든 주소의 총 크기가 max-size-bytes 값으로 제한되는 것은 아닙니다.

address-full-policy

그러면 주소의 최대 크기에 도달할 때 브로커가 수행하는 작업입니다.

이전 예에서 my.blocking.address 주소로 전송된 메시지가 메모리의 300000 바이트를 초과하면 브로커는 Core 또는 OpenWire 메시지 생산자의 추가 메시지를 차단하기 시작합니다.

7.3.2. AMQP 생산자 차단

Core 및 OpenWire와 같은 프로토콜은 창 크기 흐름 제어 시스템을 사용합니다. 이 시스템에서 크레딧은 바이트를 나타내며 생산자에게 할당됩니다. 생산자가 메시지를 보내려는 경우 생산자는 메시지 크기에 충분한 크레딧이 있을 때까지 기다려야 합니다.

반면 AMQP 흐름 제어 크레딧은 바이트를 나타내지 않습니다. 대신 AMQP 크레딧은 메시지 크기와 관계없이 생산자가 보낼 수 있는 메시지 수를 나타냅니다. 따라서 AMQP 생산자가 주소의 max-size-bytes 값을 훨씬 초과할 수 있는 경우도 있습니다.

따라서 AMQP 생산자를 차단하려면 다른 구성 요소 max-size-bytes-reject-threshold 를 사용해야 합니다. 일치하는 주소 또는 주소 집합의 경우 이 요소는 메모리에 있는 모든 AMQP 메시지의 최대 크기(바이트)를 지정합니다. 메모리에 있는 모든 메시지의 총 크기가 지정된 제한에 도달하면 브로커는 AMQP 생산자가 추가 메시지를 전송하지 못하도록 차단합니다.

다음 절차에서는 AMQP 메시지 생산자에 대한 메시지 차단을 구성하는 방법을 보여줍니다.

사전 요구 사항

- 주소 및 주소 설정을 구성하는 방법에 대해 잘 알고 있어야 합니다. 자세한 내용은 4장. 주소 및 큐 구성의 내용을 참조하십시오.

프로세스

1. <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. 일치하는 주소 또는 주소 집합에 대해 구성한 address-setting 요소의 경우 메모리에 있는 모든 AMQP 메시지의 최대 크기를 지정합니다. 예를 들면 다음과 같습니다.

```
<address-settings>
  <address-setting match="my.amqp.blocking.address">
    ...
    <max-size-bytes-reject-threshold>300000</max-size-bytes-reject-threshold>
    ...
  </address-setting>
</address-settings>
```

max-size-bytes-reject-threshold

브로커가 추가 AMQP 메시지를 차단하기 전에 주소에 허용되는 메모리의 최대 크기(바이트)입니다. 지정한 값은 "K", "MB" 및 "GB"와 같은 바이트 표기법도 지원합니다. 기본적으

로 **max-size-bytes-reject-threshold** 는 **-1** 로 설정되어 최대 크기가 없음을 의미합니다.

참고

address-setting 요소에 **max-size-bytes-reject-threshold** 를 지정하면 일치하는 각 주소에 값이 적용됩니다. 이 값을 지정하면 일치하는 모든 주소의 총 크기가 **max-size-bytes-reject-threshold** 로 제한되는 것은 아닙니다.

이전 예에서 **my.amqp.blocking.address** 주소로 전송된 메시지가 메모리에서 **300000**바이트를 초과하면 브로커는 **AMQP** 생산자의 추가 메시지를 차단하기 시작합니다.

7.4. 멀티 캐스트 주소의 메모리 사용량 이해

메시지가 바인딩된 멀티캐스트 큐가 있는 주소로 라우팅되는 경우 메모리에 메시지의 복사본이 하나뿐입니다. 각 큐에는 메시지에 대한 참조 만 있습니다. 이로 인해 메시지를 참조하는 모든 대기열이 전달된 후에만 연결된 메모리가 릴리스됩니다.

이러한 유형의 상황에서는 느린 소비자가 있는 경우 전체 주소가 성능에 부정적인 영향을 미칠 수 있습니다.

예를 들어 다음 시나리오를 고려하십시오.

- 주소에는 멀티 캐스트 라우팅 유형을 사용하는 **10**개의 큐가 있습니다.
- 느린 소비자로 인해 대기열 중 하나가 메시지를 전달하지 않습니다. 나머지 **9**개의 대기열은 계속 메시지를 전달하며 비어 있습니다.
- 메시지는 계속 주소에 도달합니다. 느린 소비자를 가진 큐는 메시지에 대한 참조를 계속 누적하여 브로커가 메시지를 메모리에 유지합니다.
- 주소의 최대 크기에 도달하면 브로커가 페이지 메시지로 시작합니다.

이 시나리오에서는 느린 단일 소비자로 인해 모든 큐에 있는 소비자는 페이지 시스템의 메시지를 사용해야 하므로 추가 **IO**가 필요합니다.

추가 리소스

- *브로커와 생산자와 소비자 간의 데이터 흐름을 규제하기 위해 흐름 제어를 구성하는 방법을 알아보려면 **AMQ Core Protocol JMS** 문서의 흐름 제어를 참조하십시오.*

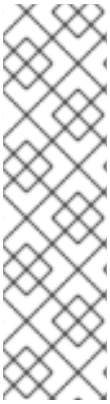
8장. 대용량 메시지 처리

클라이언트는 브로커의 내부 버퍼 크기를 초과할 수 있는 대규모 메시지를 보내 예기치 않은 오류가 발생할 수 있습니다. 이 상황을 방지하려면 메시지가 지정된 최소값보다 큰 경우 메시지를 파일로 저장하도록 브로커를 구성할 수 있습니다. 이러한 방식으로 대용량 메시지를 처리한다는 것은 브로커가 메시지를 메모리에 보유하지 않음을 의미합니다. 대신 디스크 또는 브로커가 큰 메시지 파일을 저장하는 데이터베이스 테이블에 디렉터리를 지정합니다.

브로커가 메시지를 큰 메시지로 저장하면 큐는 대규모 메시지 디렉터리 또는 데이터베이스 테이블에 있는 파일에 대한 참조를 유지합니다.

코어 프로토콜, **AMQP**, **OpenWire** 및 **STOMP** 프로토콜에 대규모 메시지 처리를 사용할 수 있습니다.

코어 프로토콜 및 **OpenWire** 프로토콜의 경우 클라이언트는 연결 구성에 최소 큰 메시지 크기를 지정합니다. **AMQP** 및 **STOMP** 프로토콜의 경우 브로커 구성의 각 프로토콜에 대해 정의된 수락자에서 최소 큰 메시지 크기를 지정합니다.



참고

대용량 메시지를 생성하고 사용하는 데 다른 프로토콜을 사용하지 않는 것이 좋습니다. 이렇게 하려면 브로커가 메시지의 여러 변환을 수행해야 할 수 있습니다. 예를 들어 **AMQP** 프로토콜을 사용하여 메시지를 보내고 **OpenWire**를 사용하여 수신하려는 경우를 예로 들 수 있습니다. 이러한 상황에서 브로커는 먼저 큰 메시지의 전체 본문을 읽고 **Core** 프로토콜을 사용하도록 변환해야합니다. 그런 다음 브로커는 **OpenWire** 프로토콜로 다른 변환을 수행해야합니다. 이와 같은 메시지 변환으로 인해 브로커에 상당한 처리 오버헤드가 발생합니다.

이전 프로토콜에 대해 지정하는 최소 큰 메시지 크기는 사용 가능한 디스크 공간 양 및 메시지의 크기와 같은 시스템 리소스의 영향을 받습니다. 여러 값을 사용하여 성능 테스트를 실행하여 적절한 크기를 결정하는 것이 좋습니다.

이 섹션의 절차에서는 다음을 수행하는 방법을 보여줍니다.

- 대규모 메시지를 저장하도록 브로커 구성
- 대용량 메시지 처리를 위해 **AMQP** 및 **STOMP** 프로토콜에 대한 어셉터 구성

이 섹션에서는 대규모 메시지로 작동하도록 **AMQ Core Protocol** 및 **AMQ OpenWire JMS** 클라이언트 구성에 대한 추가 리소스에도 연결됩니다.

8.1. 대규모 메시지 처리를 위해 브로커 구성

다음 절차에서는 브로커가 대용량 메시지 파일을 저장하는 디스크 또는 데이터베이스 테이블에 디렉터리를 지정하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 브로커가 큰 메시지 파일을 저장할 위치를 지정합니다.

   a. 디스크에 대용량 메시지를 저장하는 경우 **core** 요소 내에 **large-messages-directory** 매개변수를 추가하고 파일 시스템 위치를 지정합니다. 예를 들면 다음과 같습니다.

```
<configuration>
  <core>
    ...
    <large-messages-directory>/path/to/my-large-messages-directory</large-
messages-directory>
    ...
  </core>
</configuration>
```

참고

large-messages-directory 의 값을 명시적으로 지정하지 않으면 브로커는 기본값 **< broker_instance_dir> /data/largemessages**를 사용합니다.

   b. 데이터베이스 테이블에 대용량 메시지를 저장하는 경우 **database-store** 요소에 **large-message-table** 매개변수를 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<store>
  <database-store>
    ...
    <large-message-table>MY_TABLE</large-message-table>
```

```
    ...
  </database-store>
</store>
```

참고

Large -message-table 의 값을 명시적으로 지정하지 않으면 브로커는 기본값 LARGE_MESSAGE_ Cryostat를 사용합니다.

추가 리소스

- 데이터베이스 저장소 구성에 대한 자세한 내용은 6.2절. “데이터베이스에 메시지 데이터 유지” 을 참조하십시오.

8.2. 대용량 메시지 처리를 위한 AMQP 어셉터 구성

다음 절차에서는 지정된 크기보다 큰 AMQP 메시지를 큰 메시지로 처리하도록 AMQP 어셉터를 구성하는 방법을 보여줍니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

   브로커 구성 파일의 기본 AMQP 수락자는 다음과 같습니다.

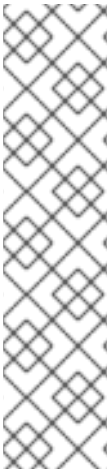
```
<acceptors>
   ...
   <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpol
l=true;amqpCredits=1000;amqpLowCredits=300</acceptor>
   ...
</acceptors>
```

2. 기본 AMQP 수락자(또는 구성한 다른 AMQP 수락자)에서 amqpMinLargeMessageSize 속성을 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<acceptors>
   ...
   <acceptor name="amqp">tcp://0.0.0.0:5672?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=AMQP;useEpol
l=true;amqpCredits=1000;amqpLowCredits=300;amqpMinLargeMessageSize=204800</
```

```
acceptor>
   ...
</acceptors>
```

이전 예에서 브로커는 포트 **5672**에서 **AMQP** 메시지를 수락하도록 구성되어 있습니다. **amqpMinLargeMessageSize** 값을 기반으로 수락자가 **204800**바이트보다 크거나 같은 본문이 있는 **AMQP** 메시지를 수신하는 경우 브로커는 메시지를 큰 메시지로 저장합니다. 이 속성의 값을 명시적으로 지정하지 않으면 브로커는 기본값 **102400**(즉, **100KB**)을 사용합니다.



참고

- **amqpMinLargeMessageSize** 를 **-1**로 설정하면 **AMQP** 메시지의 대규모 메시지 처리가 비활성화됩니다.
- 브로커가 **amqpMinLargeMessageSize** 값을 초과하지 않는 영구 **AMQP** 메시지를 수신하지만 메시징 저널 버퍼 크기(**journal-buffer-size** 구성 매개변수를 사용하여 지정됨)를 초과하는 경우 브로커는 메시지를 저널에 저장하기 전에 메시지를 대규모 코어 프로토콜 메시지로 변환합니다.

8.3. 대용량 메시지 처리를 위한 STOMP 어셉터 구성

다음 절차에서는 지정된 크기보다 큰 메시지로 **STOMP** 메시지를 처리하도록 **STOMP** 수락자를 구성하는 방법을 보여줍니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

   브로커 구성 파일의 기본 **AMQP** 수락자는 다음과 같습니다.

```
<acceptors>
   ...
   <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEp
oll=true</acceptor>
   ...
</acceptors>
```

2. 기본 **STOMP** 허용자(또는 구성한 다른 **STOMP** 허용자)에서 **stompMinLargeMessageSize** 속성을 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<acceptors>
   ...
   <acceptor name="stomp">tcp://0.0.0.0:61613?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=STOMP;useEp
oll=true;stompMinLargeMessageSize=204800</acceptor>
   ...
</acceptors>
```

이전 예에서 브로커는 포트 61613에서 STOMP 메시지를 수락하도록 구성되어 있습니다. 수락자가 204800 바이트(즉, 200 kilobytes)보다 크거나 같은 STOMP 메시지를 수신하는 경우 브로커는 메시지를 큰 메시지로 저장합니다. 이 속성의 값을 명시적으로 지정하지 않으면 브로커는 기본값 102400(즉, 100KB)을 사용합니다.

참고

대규모 메시지를 STOMP 소비자에게 제공하기 위해 브로커는 메시지를 클라이언트로 보내기 전에 메시지를 큰 메시지에서 일반 메시지로 자동으로 변환합니다. 큰 메시지가 압축되면 브로커는 STOMP 클라이언트에 전송하기 전에 압축을 풉니다.

8.4. 대용량 메시지 및 JAVA 클라이언트

대규모 메시지를 사용하는 클라이언트를 작성하는 Java 개발자가 사용할 수 있는 두 가지 옵션이 있습니다.

한 가지 옵션은 InputStream 및 OutputStream 의 인스턴스를 사용하는 것입니다. 예를 들어 FileInputStream 을 사용하여 물리적 디스크의 대용량 파일에서 가져온 메시지를 보낼 수 있습니다. 그러면 수신자가 메시지를 로컬 파일 시스템의 위치로 스트리밍하는 데 FileOutputStream 을 사용할 수 있습니다.

또 다른 옵션은 JMS Cryostats Message 또는 StreamMessage 를 직접 스트리밍하는 것입니다. 예를 들면 다음과 같습니다.

```
BytesMessage rm = (BytesMessage)cons.receive(10000);
byte data[] = new byte[1024];
for (int i = 0; i < rm.getBodyLength(); i += 1024)
{
  int numberOfBytes = rm.readBytes(data);
  // Do whatever you want with the data
}
```

추가 리소스

- AMQ Core Protocol JMS 클라이언트의 대규모 메시지 작업에 대한 자세한 내용은 다음을

참조하십시오.

- 큰 메시지 옵션
- 스트리밍된 큰 메시지에 쓰기
- 스트리밍된 대규모 메시지에서 읽기

- 큰 메시지로 작업하는 예제는 큰 메시지 예제 를 참조하십시오. 예제 프로그램에 대한 자세한 내용은 **AMQ Broker** 실행 예제를 참조하십시오.

9장. 데드된 연결 감지

경우에 따라 클라이언트가 예기치 않게 중지되고 리소스를 정리할 기회가 없습니다. 이 경우 리소스를 잘못된 상태로 두고 브로커가 메모리 부족 또는 기타 시스템 리소스가 부족해질 수 있습니다. 브로커는 클라이언트 연결이 가비지 컬렉션 시 제대로 종료되지 않았음을 감지합니다. 그런 다음 연결이 닫히고 아래 메시지와 유사한 메시지가 로그에 기록됩니다. 로그는 클라이언트 세션이 인스턴스화된 정확한 코드 행을 캡처합니다. 이를 통해 오류를 식별하고 수정할 수 있습니다.

```
[Finalizer] 20:14:43,244 WARNING [org.apache.activemq.artemis.core.client.impl.DelegatingSession]
I'm closing a JMS Conection you left open. Please make sure you close all connections explicitly
before let
ting them go out of scope!
[Finalizer] 20:14:43,244 WARNING [org.apache.activemq.artemis.core.client.impl.DelegatingSession]
The session you didn't close was created here:
java.lang.Exception
   at org.apache.activemq.artemis.core.client.impl.DelegatingSession.<init>
(DelegatingSession.java:83)
   at org.acme.yourproject.YourClass (YourClass.java:666) 1
```

1

연결이 인스턴스화된 클라이언트 코드의 줄입니다.

9.1. 연결 시간-TO-LIVE

클라이언트와 서버 간의 네트워크 연결이 실패하고 온라인 상태가 되어 클라이언트가 다시 다시 연결할 수 있으므로 **AMQ Broker**는 비활성 서버 측 리소스를 정리할 때까지 기다립니다. 이 대기 기간을 **TTL(Time-to-Live)**이라고 합니다. 네트워크 기반 연결의 기본 **TTL**은 **60000** 밀리초(1분)입니다. **in-VM** 연결의 기본 **TTL**은 **-1** 이며, 이는 브로커가 브로커 측의 연결을 시간 초과하지 않음을 의미합니다.

브로커에서 Time-To-Live 구성

클라이언트가 자체 연결 **TTL**을 지정하지 않으려면 브로커 측에 글로벌 값을 설정할 수 있습니다. 이 작업은 브로커 구성에 **connection-ttl-override** 요소를 지정하여 수행할 수 있습니다.

TTL 위반의 연결을 확인하는 논리는 **connection-ttl-check-interval** 요소에 의해 결정된 브로커에서 주기적으로 실행됩니다.

프로세스

- **connection-ttl-override** 구성 요소를 추가하고 아래 예제와 같이 **time-to-live** 값을 제공하여 **< broker_instance_dir> /etc/broker.xml** 을 편집합니다.

```
<configuration>
 <core>
  ...
  <connection-ttl-override>30000</connection-ttl-override> (1)
  <connection-ttl-check-interval>1000</connection-ttl-check-interval> (2)
  ...
 </core>
</configuration>
```

(1) 모든 연결에 대한 글로벌 **TTL**은 **30000**밀리초로 설정됩니다. 기본값은 클라이언트가 고유한 **TTL**을 설정할 수 있는 **-1** 입니다.

(2) 실패한 연결에 대한 검사 간격은 **1000**밀리초로 설정됩니다. 기본적으로 검사는 **2000** 밀리초마다 수행됩니다.

9.2. 비동기 연결 실행 비활성화

브로커 측에서 수신된 대부분의 패킷은 리모팅 스레드에서 실행됩니다. 이러한 패킷은 단기 실행 작업을 나타내며 성능상의 이유로 원격 스레드에서 항상 실행됩니다. 그러나 일부 패킷 유형은 리모팅 스레드 대신 스레드 풀을 사용하여 실행되므로 약간의 네트워크 대기 시간이 추가됩니다.

스레드 풀을 사용하는 패킷 유형은 아래에 나열된 **Java** 클래스 내에서 구현됩니다. 클래스는 모두 **org.apache.actiinvemq.artemis.core.protocol.core.impl.wireformat** 패키지에 있습니다.

- **RollbackMessage**
- **SessionCloseMessage**
- **SessionCommitMessage**
- **SessionXACommitMessage**
- **SessionXAPrepareMessage**

- ***SessionXARollbackMessage***

프로세스

- *비동기 연결 실행을 비활성화하려면 아래 예제와 같이 **< broker_instance_dir> /etc/broker.xml** 에 **async-connection-execution-enabled** 구성 요소를 추가하고 **false** 로 설정합니다. 기본값은 **true**입니다.*

```
<configuration>
 <core>
  ...
  <async-connection-execution-enabled>false</async-connection-execution-enabled>
  ...
 </core>
</configuration>
```

추가 리소스

- *장애가 발생한 연결을 감지하도록 **AMQ Core Protocol JMS** 클라이언트를 구성하는 방법을 알아보려면 **AMQ Core Protocol JMS** 문서에서 중단된 연결 탐지 를 참조하십시오.*

- ***AMQ Core Protocol JMS** 클라이언트에서 연결 **time-to-live**를 구성하는 방법을 알아보려면 **AMQ Core Protocol JMS** 문서에서 **time-to-live** 구성 을 참조하십시오.*

10장. 중복 메시지 감지

중복 메시지를 자동으로 감지하고 필터링하도록 브로커를 구성할 수 있습니다. 즉, 중복 탐지 논리를 구현할 필요가 없습니다.

중복 감지가 없으면 예기치 않은 연결 오류가 발생할 경우 클라이언트는 브로커로 전송된 메시지가 수신되었는지 여부를 확인할 수 없습니다. 이 경우 클라이언트는 브로커가 메시지를 수신하지 않았다고 가정하고 다시 보낼 수 있습니다. 이로 인해 중복 메시지가 생성됩니다.

예를 들어 클라이언트가 브로커에 메시지를 전송한다고 가정합니다. 브로커가 메시지를 수신하고 처리하기 전에 브로커 또는 연결이 실패하면 메시지는 해당 주소에 도달하지 못합니다. 실패로 인해 클라이언트에서 브로커로부터 응답을 받지 않습니다. 브로커가 메시지를 수신하고 처리한 후 브로커 또는 연결이 실패하면 메시지가 올바르게 라우팅되지만 클라이언트는 여전히 응답을 수신하지 못합니다.

또한 트랜잭션을 사용하여 성공을 결정하는 것이 이러한 경우에 반드시 도움이 되는 것은 아닙니다. 트랜잭션 커밋이 처리되는 동안 브로커 또는 연결이 실패하면 클라이언트에서 메시지를 성공적으로 전송했는지 여부를 확인할 수 없습니다.

이러한 상황에서 가정된 오류를 해결하기 위해 클라이언트는 최신 메시지를 다시 보냅니다. 결과적으로 시스템에 부정적인 영향을 미치는 메시지가 중복될 수 있습니다. 예를 들어 주문 처리 시스템에서 브로커를 사용하는 경우 중복 메시지가 구매 발주서가 두 번 처리됨을 의미할 수 있습니다.

다음 절차에서는 이러한 유형의 상황에서 보호하기 위해 중복 메시지 탐지를 구성하는 방법을 보여줍니다.

10.1. 중복 ID 캐시 구성

브로커가 중복된 메시지를 감지할 수 있도록 하려면 각 메시지를 전송할 때 생산자가 메시지 속성 **_AMQ_DUPL_ID** 에 대한 고유한 값을 제공해야 합니다. 브로커는 **_AMQ_DUPL_ID** 속성의 수신된 값 캐시를 유지 관리합니다. 브로커가 주소에서 새 메시지를 수신하면 해당 주소에 대한 캐시를 확인하여 이전에 이 속성에 동일한 값을 가진 메시지를 처리하지 않았는지 확인합니다.

각 주소에는 자체 캐시가 있습니다. 각 캐시는 원형이며 크기가 고정되어 있습니다. 즉, 새 항목이 캐시 공간 요구로 가장 오래된 항목을 대체합니다.

다음 절차에서는 브로커의 각 주소에서 사용하는 **ID** 캐시를 전역적으로 구성하는 방법을 보여줍니다.

프로세스

프로세스

1. <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. core 요소 내에서 id-cache-size 및 persist-id-cache 속성을 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.

```
<configuration>
 <core>
  ...
  <id-cache-size>5000</id-cache-size>
  <persist-id-cache>false</persist-id-cache>
 </core>
</configuration>
```

id-cache-size

캐시의 개별 항목 수로 지정된 ID 캐시의 최대 크기입니다. 기본값은 20,000 항목입니다. 이 예에서 캐시 크기는 5,000개의 항목으로 설정됩니다.

참고

최대 캐시 크기에 도달하면 브로커가 중복 메시지 처리를 시작할 수 있습니다. 예를 들어 캐시 크기를 3000 으로 설정했다고 가정합니다. 이전 메시지가 _AMQ_DUPL_ID 값이 동일한 새 메시지가 도착하기 전에 3000개 이상의 메시지가 도착하면 브로커는 중복을 감지할 수 없습니다. 이로 인해 브로커에서 두 메시지를 모두 처리합니다.

persist-id-cache

이 속성의 값을 true 로 설정하면 브로커가 수신 시 ID를 디스크에 유지합니다. 기본값은 true입니다. 위의 예제에서는 값을 false 로 설정하여 지속성을 비활성화합니다.

추가 리소스

- AMQ Core Protocol JMS 클라이언트를 사용하여 중복 ID 메시지 속성을 설정하는 방법을 알아보려면 AMQ Core Protocol JMS 클라이언트 설명서에서 중복 메시지 탐지 사용을 참조하십시오.

10.2. 클러스터 연결에 대한 중복 탐지 구성

클러스터 연결을 구성하여 클러스터 전체에서 이동하는 각 메시지에 대해 중복 ID 헤더를 삽입할 수 있습니다.

사전 요구 사항

- *브로커 클러스터가 이미 구성되어 있어야 합니다. 자세한 내용은 다음을 참조하십시오.* ***14.2 절. "브로커 클러스터 생성"***

프로세스

1. ***&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. ***core** 요소 내에서 지정된 클러스터 연결에 대해 **use-duplicate-detection** 속성을 추가하고 값을 지정합니다. 예를 들면 다음과 같습니다.*

```
<configuration>
 <core>
  ...
  <cluster-connections>
   <cluster-connection name="my-cluster">
    <use-duplicate-detection>true</use-duplicate-detection>
    ...
   </cluster-connection>
    ...
  </cluster-connections>
 </core>
</configuration>
```

use-duplicate-detection

*이 속성의 값을 **true** 로 설정하면 클러스터 연결에서 처리하는 각 메시지에 대해 중복 **ID** 헤더를 삽입합니다.*

11장. 메시지 가로채기

AMQ Broker를 사용하면 브로커로 들어오거나 종료하는 패킷을 가로채서 패킷을 감사하거나 메시지를 필터링할 수 있습니다. 인터셉터는 가로채는 패킷을 변경할 수 있으므로 강력하지만 잠재적으로 위험할 수도 있습니다.

비즈니스 요구 사항을 충족하기 위해 인터셉터를 개발할 수 있습니다. 인터셉터는 프로토콜 고유의 프로토콜이며 적절한 인터페이스를 구현해야 합니다.

인터셉터는 부울 값을 반환하는 **intercept()** 메서드를 구현해야 합니다. 값이 **true** 이면 메시지 패킷이 계속됩니다. **false** 인 경우 프로세스가 중단되고 다른 인터셉터가 호출되지 않으며 메시지 패킷이 더 이상 처리되지 않습니다.

11.1. 인터셉터 생성

자체 수신 및 발신 인터셉터를 생성할 수 있습니다. 모든 인터셉터는 프로토콜별 프로토콜이며 각각 서버를 입력하거나 종료하는 모든 패킷에 대해 호출됩니다. 이를 통해 패킷 감사와 같은 비즈니스 요구 사항을 충족하기 위해 인터셉터를 생성할 수 있습니다. 인터셉터는 가로채는 패킷을 변경할 수 있습니다. 이로 인해 잠재적으로 위험할 수 있을 뿐만 아니라 강력하므로 주의해서 사용해야 합니다.

인터셉터 및 해당 종속 항목은 브로커의 **Java** 클래스 경로에 배치해야 합니다. 기본적으로 클래스 경로의 일부이므로 **<broker_instance_dir> /lib** 디렉터리를 사용할 수 있습니다.

프로세스

다음 예제에서는 전달된 각 패킷의 크기를 확인하는 인터셉터를 만드는 방법을 보여줍니다. 예제에서는 각 프로토콜에 대한 특정 인터페이스를 구현합니다.

- 적절한 인터페이스를 구현하고 인터셉터 () 메서드를 재정의합니다.
  - **AMQP** 프로토콜을 사용하는 경우 **org.apache.activemq.artemis.protocol.amqp.broker.AmqpInterceptor** 인터페이스를 구현합니다.

```
package com.example;

import org.apache.activemq.artemis.protocol.amqp.broker.AMQPMessage;
import org.apache.activemq.artemis.protocol.amqp.broker.AmqpInterceptor;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;
```

```
public class MyInterceptor implements AmqpInterceptor
{
  private final int ACCEPTABLE_SIZE = 1024;

  @Override
  public boolean intercept(final AMQPMessage message, RemotingConnection
connection)
  {
    int size = message.getEncodeSize();
    if (size <= ACCEPTABLE_SIZE) {
      System.out.println("This AMQPMessage has an acceptable size.");
      return true;
    }
    return false;
  }
}
```

- 코어 프로토콜을 사용하는 경우 인터셉터는 org.apache.artemis.activemq.api.core.Interceptor 인터페이스를 구현해야 합니다.

```
package com.example;

import org.apache.artemis.activemq.api.core.Interceptor;
import org.apache.activemq.artemis.core.protocol.core.Packet;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements Interceptor
{
  private final int ACCEPTABLE_SIZE = 1024;

  @Override
  boolean intercept(Packet packet, RemotingConnection connection)
  throws ActiveMQException
  {
    int size = packet.getPacketSize();
    if (size <= ACCEPTABLE_SIZE) {
      System.out.println("This Packet has an acceptable size.");
      return true;
    }
    return false;
  }
}
```

- MQTT 프로토콜을 사용하는 경우 org.apache.activemq.artemis.core.protocol.mqtt.MQTTInterceptor 인터페이스를 구현합니다.

```
package com.example;

import org.apache.activemq.artemis.core.protocol.mqtt.MQTTInterceptor;
import io.netty.handler.codec.mqtt.MqttMessage;
```

```
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements Interceptor
{
 private final int ACCEPTABLE_SIZE = 1024;

 @Override
 boolean intercept(MqttMessage mqttMessage, RemotingConnection connection)
 throws ActiveMQException
 {
  byte[] msg = (mqttMessage.toString()).getBytes();
  int size = msg.length;
  if (size <= ACCEPTABLE_SIZE) {
   System.out.println("This MqttMessage has an acceptable size.");
   return true;
  }
  return false;
 }
}
```

- *STOMP 프로토콜을 사용하는 경우 org.apache.activemq.artemis.core.protocol.stompFrameInterceptor 인터페이스를 구현합니다.*

```
package com.example;

import org.apache.activemq.artemis.core.protocol.stomp.StompFrameInterceptor;
import org.apache.activemq.artemis.core.protocol.stomp.StompFrame;
import org.apache.activemq.artemis.spi.core.protocol.RemotingConnection;

public class MyInterceptor implements Interceptor
{
 private final int ACCEPTABLE_SIZE = 1024;

 @Override
 boolean intercept(StompFrame stompFrame, RemotingConnection connection)
 throws ActiveMQException
 {
  int size = stompFrame.getEncodedSize();
  if (size <= ACCEPTABLE_SIZE) {
   System.out.println("This StompFrame has an acceptable size.");
   return true;
  }
  return false;
 }
}
```

11.2. 인터셉터를 사용하도록 브로커 구성

인터셉터를 생성한 후에는 해당 브로커를 사용하도록 브로커를 구성해야 합니다.

사전 요구 사항

*브로커가 사용할 수 있도록 구성하기 전에 인터셉터 클래스를 생성하고 브로커의 **Java** 클래스 경로에 추가 (및 해당 종속 항목)를 추가해야 합니다. 기본적으로 클래스 경로의 일부이므로 **<broker_instance_dir> /lib** 디렉터리를 사용할 수 있습니다.*

프로세스

- *__<broker _instance_dir> /etc/broker.xml__에 구성을 추가하여 인터셉터를 사용하도록 브로커를 구성합니다.*
  - *인터셉터가 수신되는 메시지를 위한 경우, 해당 클래스 이름을 __Remoting- incoming-interceptors__ 목록에 추가합니다.*

```
<configuration>
 <core>
  ...
  <remoting-incoming-interceptors>
    <class-name>org.example.MyIncomingInterceptor</class-name>
  </remoting-incoming-interceptors>
  ...
 </core>
</configuration>
```

  - *인터셉터가 발신 메시지를 대상으로 하는 경우 __remoting-outgoing-interceptors__ 목록에 __class-name__ 을 추가합니다.*

```
<configuration>
 <core>
  ...
  <remoting-outgoing-interceptors>
    <class-name>org.example.MyOutgoingInterceptor</class-name>
  </remoting-outgoing-interceptors>
 </core>
</configuration>
```

추가 리소스

- *__AMQ Core Protocol JMS__ 클라이언트에서 인터셉터를 구성하는 방법을 알아보려면 __AMQ Core Protocol JMS__ 문서의 메시지 인터셉터 사용을 참조하십시오.*

12장. 메시지 분할 및 메시지 흐름 분할

AMQ Broker에서는 클라이언트 애플리케이션 논리를 변경하지 않고 메시지를 한 주소에서 다른 주소로 투명하게 전환할 수 있도록 다양한 오브젝트를 구성할 수 있습니다. 메시지 사본을 지정된 전달 주소로 전달하여 메시지 흐름을 효과적으로 분할하도록 다각형을 구성할 수도 있습니다.

12.1. 메시지 다이렉트 작동 방식

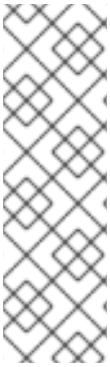
다각형을 사용하면 클라이언트 애플리케이션 논리를 변경하지 않고도 한 주소로 라우팅되는 메시지를 투명하게 다른 주소로 전환할 수 있습니다. 브로커 서버의 다양한 유형을 메시지의 라우팅 테이블 유형으로 생각하십시오.

분산은 배타적 일 수 있습니다. 즉, 원래 주소로 이동하지 않고도 메시지가 지정된 전달 주소로 전환됩니다.

다각형은 비독점 일 수도 있습니다. 즉, 메시지는 원래 주소로 계속되는 반면 브로커는 지정된 전달 주소로 메시지 사본을 보냅니다. 따라서 메시지 흐름을 분할하는 데 비독점 다각형을 사용할 수 있습니다. 예를 들어 주문 큐로 전송된 모든 순서를 별도로 모니터링하려는 경우 메시지 흐름을 분할할 수 있습니다.

단일 주소에 대해 다중 다각형을 구성할 수 있습니다. 주소에 배타적 및 비독점 다이렉트가 구성되어 있는 경우 브로커는 독점적인 다이버를 먼저 처리합니다. 특정 메시지가 이미 배타적인 다이버트에 의해 변형된 경우 브로커는 해당 메시지에 대해 비독점 다이버를 처리하지 않습니다. 이 경우 메시지는 원래 주소로 이동하지 않습니다.

브로커가 메시지를 분산하면 브로커는 새 메시지 ID를 할당하고 메시지 주소를 새 전달 주소로 설정합니다. _AMQ_ORIG_ADDRESS (문자열 유형) 및 _AMQ_ORIG_MESSAGE_ID (long type) 메시지 속성을 통해 원래 메시지 ID 및 주소 값을 검색할 수 있습니다. Core API를 사용하는 경우 Message.HDR_ORIGINAL_ADDRESS 및 Message.HDR_ORIG_MESSAGE_ID 속성을 사용합니다.



참고

동일한 브로커 서버의 주소로만 메시지를 전달할 수 있습니다. 다른 서버의 주소로 전환하려는 경우 일반적인 해결 방법은 먼저 메시지를 로컬 저장소 및 전달 큐로 전달하는 것입니다. 그런 다음 해당 대기열에서 사용하는 브릿지를 설정하고 다른 브로커의 주소로 메시지를 전달합니다. 다양한 브릿지를 브리지와 결합하면 지리적으로 분산된 브로커 서버 간에 분산 라우팅 연결 네트워크를 만들 수 있습니다. 이렇게 하면 글로벌 메시징 메시를 생성할 수 있습니다.

12.2. 메시지 다이버 구성

브로커 인스턴스에서 다각형을 구성하려면 **broker.xml** 구성 파일의 **core** 요소 내에 **divert** 요소를 추가합니다.

```
<core>
...
  <divert name= >
      <address> </address>
      <forwarding-address> </forwarding-address>
      <filter string= >
      <routing-type> </routing-type>
      <exclusive> </exclusive>
  </divert>
...
</core>
```

divert

다양한 형식의 이름이 지정된 인스턴스입니다. 각 다이버 에 고유한 이름이 있는 한 **broker.xml** 구성 파일에 다양한 요소를 여러 개 추가할 수 있습니다.

address

다양한 메시지를 전달할 주소

forwarding-address

메시지를 전달할 주소

filter

선택적 메시지 필터입니다. 필터를 구성하면 필터 문자열과 일치하는 메시지만 전환됩니다. 필터를 지정하지 않으면 모든 메시지는 다이버트에 의해 일치하는 것으로 간주됩니다.

routing-type

다양한 메시지의 라우팅 유형입니다. **vart**를 다음과 같이 구성할 수 있습니다.

- 메시지에 **anycast** 또는 멀티 캐스트 라우팅 유형을 적용
- 스트립 (즉, 기존 라우팅 유형 제거)
- 기존 라우팅 유형을 통과 (즉, 보존)

라우팅 유형의 제어는 메시지에 이미 라우팅 유형이 설정되어 있지만 다른 라우팅 유형을 사용하는 주소로 메시지를 전환하려는 경우 유용합니다. 예를 들어 브로커는 **anycast** 라우팅 유형의 메시지를 다이버트의 **routing-type** 매개변수를 **MULTICAST** 로 설정하지 않는 한 멀티캐스트 를 사용하는 큐로 라우팅할 수 없습니다. 다각형의 **routing-type** 매개변수에 유효한 값은 **ANYCAST,MULTICAST,PASS, STRIP** 입니다. 기본값은 **STRIP** 입니다.

exclusive

다이버트가 배타적인지(속성을 **true**로 설정) 또는 배타적이지 않은지(속성 속성을 **false**로 설정) 할지 여부를 지정합니다.

다음 하위 섹션에서는 배타적이고 포괄적이지 않은 다이버에 대한 구성 예를 보여줍니다.

12.2.1. 예외적인 다각형 예

다음은 예외적인 다이버에 대한 구성의 예입니다. 배타적으로 구성된 메시지는 원래 구성된 주소에서 새 주소로 일치하는 모든 메시지를 다릅니다. 일치하는 메시지는 원래 주소로 라우팅되지 않습니다.

```
<divert name="prices-divert">
  <address>priceUpdates</address>
  <forwarding-address>priceForwarding</forwarding-address>
  <filter string="office='New York'"/>
  <exclusive>true</exclusive>
</divert>
```

이전 예제에서는 **address priceUpdates** 로 전송된 모든 메시지를 다른 로컬 주소 **priceForwarding** 으로 전달하는 다양한 **price -divert** 를 정의합니다. 메시지 필터 문자열도 지정합니다. 메시지 속성 사무실과 값이 있는 메시지만 노크래프트 됩니다. 다른 모든 메시지는 원래 주소로 라우팅됩니다. 마지막으로, 다각형이 배타적임을 지정합니다.

12.2.2. 지원되지 않는 다양한 예

다음은 포함되지 않은 다양한 구성에 대한 예제입니다. 비독점에서 메시지는 원래 주소로 계속 진행되지만 브로커는 지정된 전달 주소로 메시지 사본을 보냅니다. 따라서 비독점 다각형은 메시지 흐름을 분할하는 방법입니다.

```
<divert name="order-divert">
  <address>orders</address>
  <forwarding-address>spyTopic</forwarding-address>
  <exclusive>false</exclusive>
</divert>
```

*이전 예제에서는 주소 순으로 전송된 모든 메시지의 복사본을 가져와서 **spyTopic** 이라는 로컬 주소로 전송하는 다양한 주문 **-divert** 를 정의합니다. 또한 다이버가 포함되지 않도록 지정합니다.*

추가 리소스

배타적 및 비독점 다이버를 사용하는 자세한 예제와 브리지를 사용하여 다른 브로커로 메시지를 전달하는 방법은 다양한 예제(외부) 를 참조하십시오.

13장. 메시지 필터링

AMQ Broker는 SQL 92 표현식 구문의 하위 집합을 기반으로 강력한 필터 언어를 제공합니다. 필터 언어는 JMS 선택기에 사용되는 것과 동일한 구문을 사용하지만 사전 정의된 식별자는 다릅니다. 아래 표에는 AMQ Broker 메시지에 적용되는 식별자가 나열되어 있습니다.

표 13.1. 메시지 필터링을 위한 Identfiers

identifier	속성
AMQPriority	메시지의 우선순위입니다. 메시지 우선순위는 **0** 에서 **9** 까지의 유효한 값이 있는 정수입니다. **0** 이 가장 낮은 우선순위이고 **9** 가 가장 높습니다.
AMQExpiration	메시지의 만료 시간입니다. 값은 긴 정수입니다.
AMQDurable	메시지가 유효한지 여부입니다. 값은 문자열입니다. 유효한 값은 **DURABLE** 또는 **NON_DURABLE** 입니다.
AMQTimestamp	메시지가 생성된 시점의 타임스탬프입니다. 값은 긴 정수입니다.
AMQSize	메시지의 **encodeSize** 속성 값입니다. **encodeSize** 의 값은 메시지가 저널에 걸리는 공간(바이트)입니다. 브로커는 메시지를 인코딩하도록 설정된 이중 바이트 문자를 사용하므로 메시지의 실제 크기는 **encodeSize** 값의 절반입니다.

코어 필터 표현식에 사용되는 다른 모든 식별자는 메시지의 속성으로 간주됩니다. JMS Messages의 선택기 구문에 대한 문서는 Java EE API 를 참조하십시오.

13.1. 필터를 사용하도록 큐 구성

< broker_instance_dir> /etc/broker.xml 에서 구성한 큐에 필터를 추가할 수 있습니다. 필터 표현식과 일치하는 메시지만 큐를 입력합니다.

프로세스

- 원하는 큐에 필터 요소를 추가하고 요소의 값으로 적용할 필터를 포함합니다. 아래 예제에서 NEWS='Technical'이 큐 기술 Queue 에 추가되었습니다.

```
<configuration>
 <core>
  ...
  <addresses>
     <address name="myQueue">
        <anycast>
           <queue name="myQueue">
```

```
            <filter string="NEWS='technology'"/>
          </queue>
        </anycast>
      </address>
    </addresses>
  </core>
</configuration>
```

13.2. JMS 메시지 속성 필터링

JMS 사양은 선택기에서 사용할 때 **String** 속성을 숫자 유형으로 변환해서는 안 됩니다. 예를 들어 메시지에 **String** 값 **21** 로 설정된 **age** 속성이 있는 경우 선택기 **age > 18** 이 일치하지 않아야 합니다. 이 제한은 문자열 속성을 사용하여 메시지만 보낼 수 있으므로 **STOMP** 클라이언트를 제한합니다.

문자열을 숫자로 변환하도록 필터 구성

문자열 속성을 숫자 형식으로 변환하려면 접두사 **convert_string_expressions:** 를 필터 값에 추가합니다.

프로세스

- **convert _string_expressions:**접두사를 원하는 필터에 적용하여 **<broker_instance_dir > /etc/broker.xml** 을 편집합니다. 아래 예제에서는 **age > 18** 의 필터 값을 편집하여 **convert_string_expressions:age > 18.**

```
<configuration>
  <core>
    ...
    <addresses>
      <address name="myQueue">
        <anycast>
          <queue name="myQueue">
            <filter string="convert_string_expressions='age > 18'"/>
          </queue>
        </anycast>
      </address>
    </addresses>
  </core>
</configuration>
```

13.3. 주석에 속성을 기반으로 AMQP 메시지 필터링

브로커가 만료된 또는 전달되지 않은 **AMQP** 메시지를 구성한 만료 또는 **dead letter** 큐로 이동하기 전에 브로커는 주석과 속성을 메시지에 적용합니다. 클라이언트는 속성 또는 주석을 기반으로 필터를 생성하여 만료 또는 배달 못 한 큐에서 사용할 특정 메시지를 선택할 수 있습니다.

참고

브로커가 적용되는 속성은 내부 속성입니다. 이러한 속성은 정기적으로 사용하기 위해 클라이언트에 노출되지 않지만 필터의 클라이언트에서 지정할 수 있습니다.

다음은 메시지 속성 및 주석을 기반으로 하는 필터의 예입니다. 가능한 경우 브로커에 의한 처리가 줄어들기 때문에 속성을 기반으로 필터링하는 것이 권장되는 접근 방식입니다.

메시지 속성을 기반으로 필터링

```
ConnectionFactory factory = new JmsConnectionFactory("amqp://localhost:5672");
Connection connection = factory.createConnection();
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
connection.start();
javax.jms.Queue queue = session.createQueue("my_DLQ");
MessageConsumer consumer = session.createConsumer(queue,
"_AMQ_ORIG_ADDRESS='original_address_name'");
Message message = consumer.receive();
```

메시지 주석을 기반으로 필터링

```
ConnectionFactory factory = new JmsConnectionFactory("amqp://localhost:5672");
Connection connection = factory.createConnection();
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
connection.start();
javax.jms.Queue queue = session.createQueue("my_DLQ");
MessageConsumer consumer = session.createConsumer(queue, "\"m.x-opt-ORIG-
ADDRESS\"='original_address_name'");
Message message = consumer.receive();
```

참고

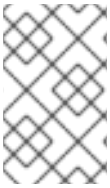
주석을 기반으로 AMQP 메시지를 사용할 때 클라이언트는 위 예와 같이 메시지 주석에 m. 접두사를 추가해야 합니다.

추가 리소스

- 브로커가 만료된 AMQP 메시지에 적용되는 주석 및 속성에 대한 자세한 내용은 4.14절. “만료된 또는 전달되지 않은 AMQP 메시지의 주석 및 속성” 에서 참조하십시오.

13.4. XML 메시지 필터링

AMQ Broker는 Cryostat를 사용하여 XML 본문이 포함된 text 메시지를 필터링하는 방법을 제공합니다. Cryostat (XML 경로 언어)는 XML 문서에서 노드를 선택하기 위한 쿼리 언어입니다.



참고

텍스트 기반 메시지만 지원됩니다. 큰 메시지 필터링은 지원되지 않습니다.

텍스트 기반 메시지를 필터링하려면 XPATH ' 형식 의 Message Selector를 생성해야 합니다 {expressi}.

메시지 본문의 예

```
<root>
   <a key='first' num='1'/>
   <b key='second' num='2'>b</b>
</root>
```

Cryostat 쿼리를 기반으로 필터링

```
XPATH 'root/a'
```

주의

Cryostat는 메시지의 본문에 적용되며 **XML** 구문 분석이 필요하므로 필터링이 일반 필터보다 훨씬 느릴 수 있습니다.

Cryostat 필터는 다음 프로토콜을 사용하는 생산자와 소비자 간에 지원됩니다.

- **OpenWire JMS**
- 코어(및 코어 **JMS**)
- **STOMP**
- **AMQP**

XML 구문 분석 구성

기본적으로 **Broker**에서 사용하는 **XML Parser**는 **JDK**에서 사용하는 플랫폼 기본 **DocumentBuilderFactory** 인스턴스입니다.

Cryostat 기본 구성에 사용되는 **XML** 구문 분석에는 다음과 같은 설정이 포함됩니다.

- **http://xml.org/sax/features/external-general-entities: false**
- **http://xml.org/sax/features/external-parameter-entities: false**
- **http://apache.org/xml/features/disallow-doctype-decl: true**

그러나 구현별 문제를 처리하기 위해 **artemis.profile** 구성 파일에서 시스템 속성을 구성하여 기능을 사용자 지정할 수 있습니다.

org.apache.activemq.documentBuilderFactory.feature:prefix

기능 구성 예

```
-Dorg.apache.activemq.documentBuilderFactory.feature:http://xml.org/sax/features/external-general-entities=true
```

14장. 브로커 클러스터 설정

클러스터는 함께 그룹화된 여러 브로커 인스턴스로 구성됩니다. 브로커 클러스터는 메시지 처리 부하를 여러 브로커에 배포하여 성능을 향상시킵니다. 또한 브로커 클러스터는 고가용성을 통해 다운타임을 최소화할 수 있습니다.

다양한 클러스터 토폴로지에서 브로커를 연결할 수 있습니다. 클러스터 내에서 각 활성 브로커는 자체 메시지를 관리하고 자체 연결을 처리합니다.

또한 클러스터 전체에서 클라이언트 연결의 균형을 유지하고 메시지를 재배포하여 브로커의 중단을 방지할 수도 있습니다.

14.1. 브로커 클러스터 이해

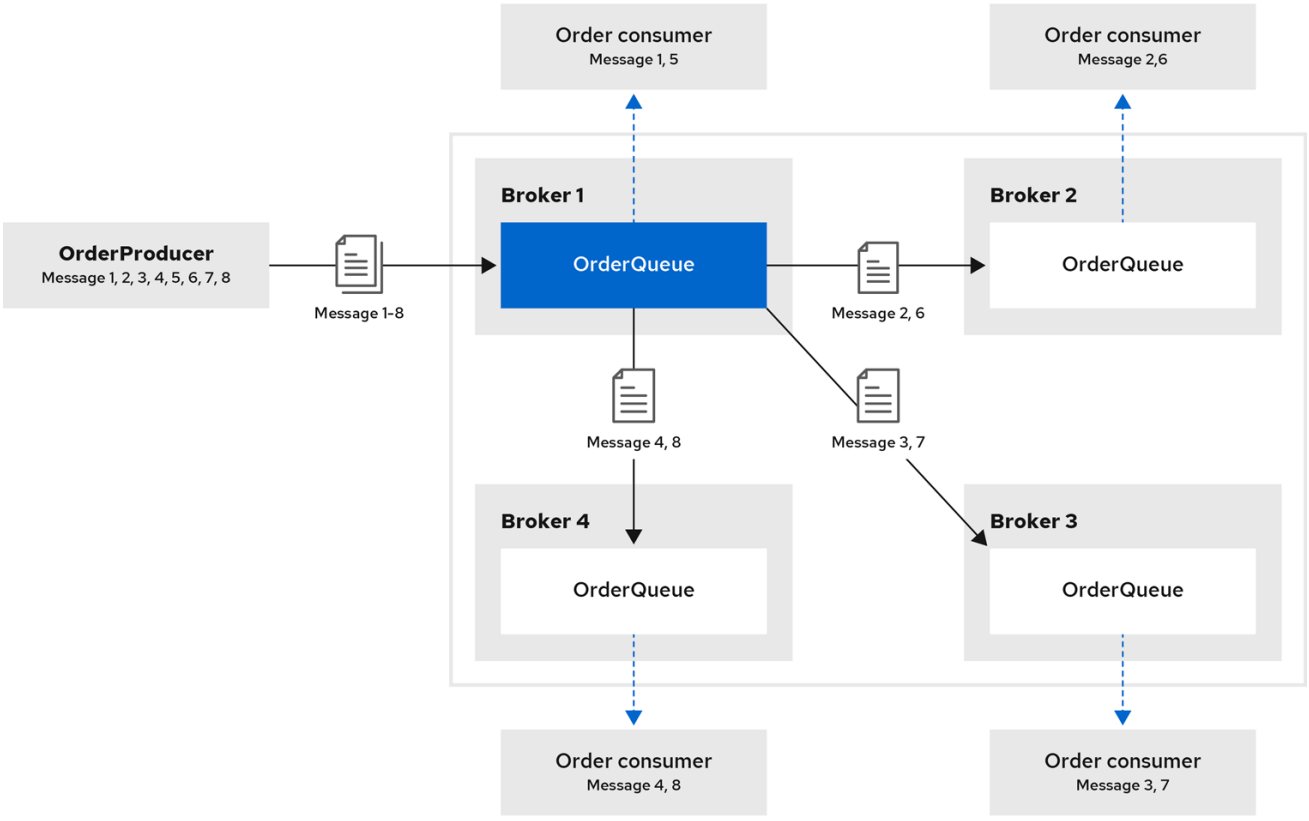
브로커 클러스터를 생성하기 전에 몇 가지 중요한 클러스터링 개념을 이해해야 합니다.

14.1.1. 브로커 클러스터의 메시지 부하를 분산하는 방법

브로커가 클러스터를 형성하기 위해 연결되어 있으면 **AMQ Broker**는 브로커 간 메시지 로드의 균형을 자동으로 조정합니다. 이렇게 하면 클러스터가 높은 메시지 처리량을 유지할 수 있습니다.

네 브로커로 구성된 대칭 클러스터를 고려하십시오. 각 브로커는 **OrderQueue** 라는 큐로 구성됩니다. **OrderProducer** 클라이언트는 **Broker1** 에 연결하고 **OrderQueue** 로 메시지를 보냅니다. **Broker1** 은 라운드 로빈 방식으로 메시지를 다른 브로커에 전달합니다. 각 브로커에 연결된 **OrderConsumer** 클라이언트는 메시지를 사용합니다.

그림 14.1. 클러스터의 메시지 로드 밸런싱



*메시지 로드 밸런싱이 없으면 **Broker1** 로 전송된 메시지는 **Broker1** 에 남아 있으며 **OrderConsumer1** 만 사용할 수 있습니다.*

***AMQ Broker**는 기본적으로 첫 번째 메시지 그룹을 첫 번째 브로커에 배포하고 두 번째 메시지 그룹을 두 번째 브로커에 분산합니다. 브로커가 시작된 순서에 따라 첫 번째, 두 번째 등을 결정합니다.*

다음을 구성할 수 있습니다.

- *일치하는 큐가 있는 브로커에 메시지를 로드 밸런싱하는 클러스터입니다.*
- *활성 소비자와 일치하는 큐가 있는 브로커에 메시지를 로드 밸런싱하는 클러스터입니다.*
- *부하 분산이 아니라 소비자가 있는 대기열에서 소비자가 없는 대기열의 메시지 재배포를 수행하기 위한 클러스터입니다.*
-

소비자가 있는 대기열에 대해 소비자가 없는 대기열에서 메시지를 자동으로 재배포하는 주소입니다.

추가 리소스

- 메시지 로드 밸런싱 정책은 각 브로커의 클러스터 연결에 **message-load-balancing** 속성을 사용하여 구성됩니다. 자세한 내용은 부록 C. 클러스터 연결 구성의 내용을 참조하십시오.
- 메시지 재배포에 대한 자세한 내용은 14.4.2절. “메시지 재배포 구성” 을 참조하십시오.

14.1.2. 브로커 클러스터의 안정성 개선 방법

브로커 클러스터는 고가용성 및 페일오버를 가능하게 하므로 독립 실행형 브로커보다 더 안정적으로 사용할 수 있습니다. 고가용성을 구성하면 브로커가 실패 이벤트에 직면하더라도 클라이언트 애플리케이션이 계속 메시지를 보내고 받을 수 있도록 할 수 있습니다.

고가용성을 사용하면 클러스터의 브로커가 라이브 백업 그룹으로 그룹화됩니다. 라이브 백업 그룹은 클라이언트 요청에 서비스를 제공하는 라이브 브로커와 실패하는 경우 라이브 브로커를 교체하기 위해 수동적으로 기다리는 백업 브로커로 구성됩니다. 오류가 발생하면 백업 브로커가 라이브 백업 그룹의 라이브 브로커를 교체하고 클라이언트는 다시 연결하고 작업을 계속합니다.

14.1.3. 클러스터 제한

다음 제한 사항은 클러스터형 환경에서 **AMQ** 브로커를 사용하는 경우에 적용됩니다.

임시 대기열

장애 조치 중에 클라이언트에 임시 대기열을 사용하는 소비자가 있는 경우 이러한 큐가 자동으로 다시 생성됩니다. 다시 생성된 큐 이름이 원래 큐 이름과 일치하지 않으므로 메시지 재배포가 실패하고 기존 임시 큐에서 메시지가 중단된 상태로 유지할 수 있습니다. 클러스터에서 임시 대기열을 사용하지 않는 것이 좋습니다. 예를 들어 요청/응답 패턴을 사용하는 애플리케이션은 **JMSReplyTo** 주소에 고정 큐를 사용해야 합니다.

14.1.4. 노드 ID 이해

브로커 노드 **ID** 는 브로커 인스턴스의 저널이 처음 생성되고 초기화될 때 프로그래밍 방식으로 생성된 **GUID(Globally Unique Identifier)**입니다. 노드 **ID**는 **server.lock** 파일에 저장됩니다. 노드 **ID**는 브로커가 독립 실행형 인스턴스인지 아니면 클러스터의 일부인지에 관계없이 브로커 인스턴스를 고유하게 식별하는 데 사용됩니다. 라이브 백업 브로커 쌍은 동일한 저널을 공유하므로 동일한 노드 **ID**를 공유합니다.

브로커 클러스터에서 브로커 인스턴스(노드)는 서로 연결하고 브리지 및 내부 "store-and-forward" 큐를 생성합니다. 이러한 내부 대기열의 이름은 다른 브로커 인스턴스의 노드 ID를 기반으로 합니다. 브로커 인스턴스도 자체 노드와 일치하는 노드 ID의 클러스터 브로드캐스트를 모니터링합니다. 브로커는 중복 ID를 식별하는 경우 로그에 경고 메시지를 생성합니다.

복제 HA(고가용성) 정책을 사용하는 경우 시작되며 check-for-live-server 가 true 로 설정된 마스터 브로커는 노드 ID를 사용하는 브로커를 검색합니다. 마스터 브로커가 동일한 노드 ID를 사용하여 다른 브로커를 발견하면 HA 구성에 따라 failback이 시작되지 않습니다.

노드 ID는 내구성이 있습니다. 즉, 브로커를 다시 시작할 수 있습니다. 그러나 브로커 인스턴스(journal 포함)를 삭제하면 노드 ID도 영구적으로 삭제됩니다.

추가 리소스

- 복제 HA 정책 구성에 대한 자세한 내용은 복제 고가용성 구성을 참조하십시오.

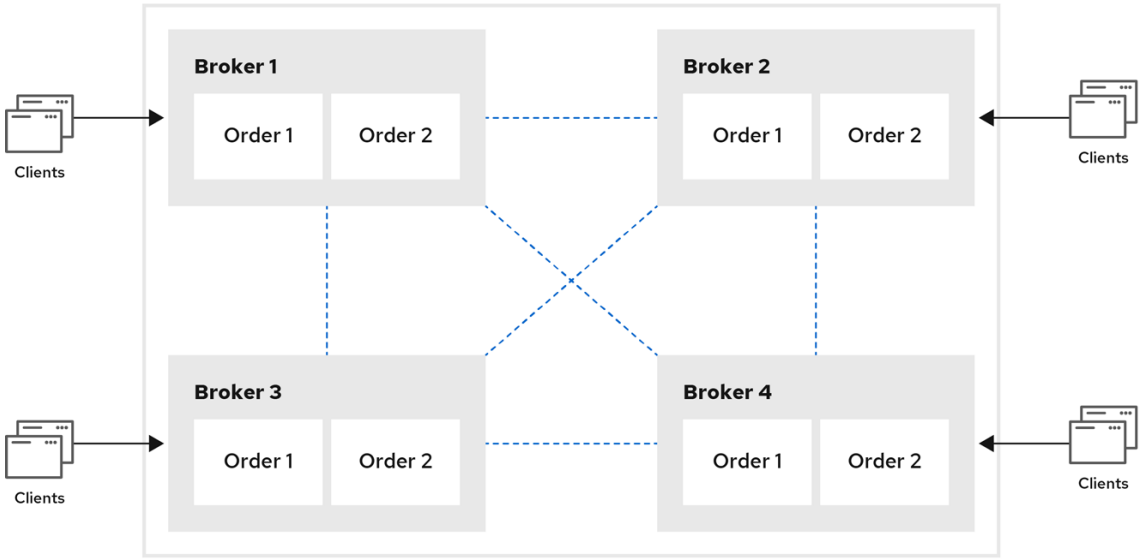
14.1.5. 일반적인 브로커 클러스터 토폴로지

브로커를 연결하여 대칭 또는 체인 클러스터 토폴로지를 형성할 수 있습니다. 구현하는 토폴로지는 환경 및 메시징 요구 사항에 따라 다릅니다.

대칭 클러스터

대칭 클러스터에서 모든 브로커는 다른 모든 브로커에 연결됩니다. 즉, 모든 브로커가 다른 브로커에서 1 홉 이상 떨어져 있지 않습니다.

그림 14.2. 대칭 클러스터 토폴로지



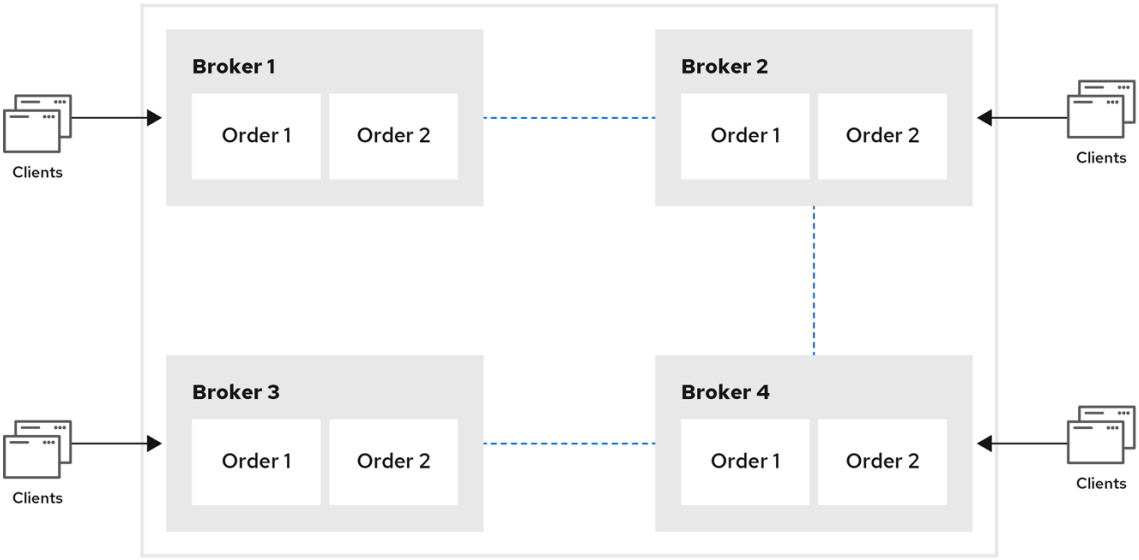
대칭 클러스터의 각 브로커는 클러스터의 다른 모든 브로커에 존재하는 모든 큐와 해당 대기열에서 수신 대기하는 소비자를 알고 있습니다. 따라서 대칭 클러스터는 체인 클러스터보다 최적의 메시지를 로드 밸런싱하고 재배포할 수 있습니다.

대칭 클러스터는 체인 클러스터보다 쉽게 설정할 수 있지만 네트워크 제한으로 인해 브로커가 직접 연결되지 않는 환경에서 사용하기 어려울 수 있습니다.

체인 클러스터

체인 클러스터에서 클러스터의 각 브로커는 클러스터의 모든 브로커에 직접 연결되지 않습니다. 대신 브로커는 체인의 각 끝에 브로커와 체인을 형성하고 다른 모든 브로커는 체인의 이전 및 다음 브로커에 연결합니다.

그림 14.3. 체인 클러스터 토폴로지



체인 클러스터는 대칭 클러스터보다 설정하기가 더 어렵지만 브로커가 별도의 네트워크에 있고 직접 연결할 수 없는 경우 유용할 수 있습니다. 체인 클러스터를 사용하면 중개 브로커가 두 브로커를 간접적으로 연결하여 두 브로커가 직접 연결되지 않더라도 메시지 간에 전달될 수 있습니다.

14.1.6. 브로커 검색 방법

Discovery는 클러스터의 브로커가 연결 세부 정보를 서로 전파하는 메커니즘입니다. **AMQ Broker**는 동적 검색 및 정적 검색을 모두 지원합니다.

동적 검색

클러스터의 각 브로커는 **UDP** 멀티 캐스트 또는 **Cryostat**를 통해 다른 멤버에게 연결 설정을 브로드캐스트합니다. 이 방법에서는 각 브로커는 다음을 사용합니다.

- 클러스터 연결에 대한 정보를 다른 잠재적인 클러스터 구성원으로 푸시하는 브로드캐스트 그룹 입니다.
- 클러스터의 다른 브로커에 대한 클러스터 연결 정보를 수신하고 저장하는 검색 그룹 입니다.

정적 검색

네트워크에서 **UDP** 또는 **Cryostat**를 사용할 수 없거나 클러스터의 각 멤버를 수동으로 지정하려면 정적 검색을 사용할 수 있습니다. 이 방법에서는 브로커가 두 번째 브로커에 연결하고 연결 세부 정보를 전송하여 클러스터를 "함께"합니다. 그런 다음 두 번째 브로커는 해당 세부 정보를 클러스터의 다른 브로커에 전파합니다.

14.1.7. 클러스터 크기 조정 고려 사항

브로커 클러스터를 생성하기 전에 메시징 처리량, 토폴로지 및 고가용성 요구 사항을 고려하십시오. 이러한 요소는 클러스터에 포함할 브로커 수에 영향을 미칩니다.

참고

클러스터를 생성한 후 브로커를 추가 및 제거하여 크기를 조정할 수 있습니다. 메시지를 손실하지 않고 브로커를 추가하고 제거할 수 있습니다.

메시징 처리량

클러스터에 필요한 메시징 처리량을 제공할 수 있는 충분한 브로커가 포함되어야 합니다. 클러스터에서 브로커가 많을수록 처리량이 향상됩니다. 그러나 대규모 클러스터는 관리하기가 복잡할 수 있습니다.

토폴로지

대칭 클러스터 또는 체인 클러스터를 생성할 수 있습니다. 선택한 토폴로지 유형은 필요한 브로커 수에 영향을 미칩니다.

자세한 내용은 14.1.5절. “일반적인 브로커 클러스터 토폴로지”의 내용을 참조하십시오.

고가용성

HA(고가용성)가 필요한 경우 클러스터를 생성하기 전에 HA 정책을 선택하는 것이 좋습니다. 각 마스터 브로커에는 하나 이상의 슬레이브 브로커가 있어야 하므로 HA 정책은 클러스터 크기에 영향을 미칩니다.

자세한 내용은 14.3절. “고가용성 구현”의 내용을 참조하십시오.

14.2. 브로커 클러스터 생성

클러스터에 참여해야 하는 각 브로커에 클러스터 연결을 구성하여 브로커 클러스터를 생성합니다. 클러스터 연결은 브로커가 다른 브로커에 연결하는 방법을 정의합니다.

정적 검색 또는 동적 검색(UDP 멀티 캐스트 또는 Cryostat)을 사용하는 브로커 클러스터를 생성할 수 있습니다.

사전 요구 사항

- 브로커 클러스터의 크기를 결정해야합니다.

  자세한 내용은 14.1.7절. “클러스터 크기 조정 고려 사항”의 내용을 참조하십시오.

14.2.1. 정적 검색을 사용하여 브로커 클러스터 생성

브로커의 정적 목록을 지정하여 브로커 클러스터를 생성할 수 있습니다. 네트워크에서 UDP 멀티 캐스트 또는 Cryostat를 사용할 수 없는 경우 이 정적 검색 방법을 사용합니다.

프로세스

1. <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. < core> 요소 내에서 다음 커넥터를 추가합니다.

   - 다른 브로커가 이 계정에 연결하는 방법을 정의하는 커넥터

   - 이 브로커가 클러스터의 다른 브로커에 연결하는 방법을 정의하는 하나 이상의 커넥터

```
<configuration>
  <core>
    ...
    <connectors>
      <connector name="netty-connector">tcp://localhost:61617</connector> 1
      <connector name="broker2">tcp://localhost:61618</connector> 2
      <connector name="broker3">tcp://localhost:61619</connector>
    </connectors>
    ...
  </core>
</configuration>
```

1 이 커넥터는 다른 브로커가 이 옵션에 연결하는 데 사용할 수 있는 연결 정보를 정의합니다. 이 정보는 검색 중에 클러스터의 다른 브로커로 전송됩니다.

2 broker2 및 broker3 커넥터는 이 브로커가 클러스터의 다른 두 브로커에 연결하는 방법을 정의합니다. 이 중 하나는 항상 사용할 수 있습니다. 클러스터에 다른 브로커가 있는 경우 초기 연결이 생성되면 이러한 커넥터 중 하나에 의해 검색됩니다.

커넥터에 대한 자세한 내용은 2.3절. “커넥터 정보” 을 참조하십시오.

3. 클러스터 연결을 추가하고 정적 검색을 사용하도록 구성합니다.

기본적으로 클러스터 연결은 대칭 토폴로지의 모든 주소에 대한 메시지를 로드 밸런싱합니다.

```
<configuration>
  <core>
    ...
    <cluster-connections>
      <cluster-connection name="my-cluster">
        <connector-ref>netty-connector</connector-ref>
        <static-connectors>
          <connector-ref>broker2-connector</connector-ref>
          <connector-ref>broker3-connector</connector-ref>
        </static-connectors>
      </cluster-connection>
    </cluster-connections>
    ...
  </core>
</configuration>
```

cluster-connection

name 속성을 사용하여 클러스터 연결의 이름을 지정합니다.

Connector-ref

다른 브로커가 이 계정에 연결하는 방법을 정의하는 커넥터입니다.

static-connectors

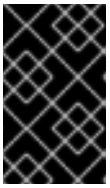
이 브로커가 클러스터의 다른 브로커에 초기 연결을 만드는 데 사용할 수 있는 하나 이상의 커넥터입니다. 이 초기 연결을 사용하면 브로커가 클러스터의 다른 브로커를 검색합니다. 클러스터가 정적 검색을 사용하는 경우에만 이 속성을 구성해야 합니다.

4. 클러스터 연결에 대한 추가 속성을 구성합니다.

이러한 추가 클러스터 연결 속성에는 대부분의 일반적인 사용 사례에 적합한 기본값이 있습니다. 따라서 기본 동작을 원하지 않는 경우에만 이러한 속성을 구성해야 합니다. 자세한 내용은 부록 C. 클러스터 연결 구성의 내용을 참조하십시오.

5. 클러스터 사용자 및 암호를 만듭니다.

AMQ Broker는 기본 클러스터 인증 정보와 함께 제공되지만 권한이 없는 원격 클라이언트가 이러한 기본 인증 정보를 사용하여 브로커에 연결하지 못하도록 변경해야 합니다.



중요

클러스터 암호는 클러스터의 모든 브로커에서 동일해야 합니다.

```
<configuration>
  <core>
    ...
    <cluster-user>cluster_user</cluster-user>
    <cluster-password>cluster_user_password</cluster-password>
    ...
  </core>
</configuration>
```

6. 추가 브로커마다 이 절차를 반복합니다.

클러스터 구성을 추가 브로커마다 복사할 수 있습니다. 그러나 다른 AMQ Broker 데이터 파일(예: 바인딩, 저널, 대규모 메시지 디렉터리)을 복사하지 마십시오. 이러한 파일은 클러스터의 노드 간에 고유해야 합니다. 그렇지 않으면 클러스터가 올바르게 포맷되지 않습니다.

추가 리소스

- 정적 검색을 사용하는 브로커 클러스터의 예는 clustered-static-discovery 예제 를 참조하십시오.

14.2.2. UDP 기반 동적 검색을 사용하여 브로커 클러스터 생성

브로커가 UDP 멀티 캐스트를 통해 동적으로 서로를 검색하는 브로커 클러스터를 생성할 수 있습니다.

프로세스

1. &lt ;broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

2. < core> 요소 내에서 커넥터를 추가합니다.

이 커넥터는 다른 브로커가 이 옵션에 연결하는 데 사용할 수 있는 연결 정보를 정의합니다. 이 정보는 검색 중에 클러스터의 다른 브로커로 전송됩니다.

```
<configuration>
  <core>
    ...
    <connectors>
      <connector name="netty-connector">tcp://localhost:61617</connector>
    </connectors>
    ...
  </core>
</configuration>
```

3. ***UDP** 브로드캐스트 그룹을 추가합니다.*

*브로드캐스트 그룹을 사용하면 브로커가 클러스터 연결에 대한 정보를 클러스터의 다른 브로커로 내보낼 수 있습니다. 이 브로드캐스트 그룹은 **UDP**를 사용하여 연결 설정을 브로드캐스트합니다.*

```
<configuration>
  <core>
    ...
    <broadcast-groups>
      <broadcast-group name="my-broadcast-group">
        <local-bind-address>172.16.9.3</local-bind-address>
        <local-bind-port>-1</local-bind-port>
        <group-address>231.7.7.7</group-address>
        <group-port>9876</group-port>
        <broadcast-period>2000</broadcast-period>
        <connector-ref>netty-connector</connector-ref>
      </broadcast-group>
    </broadcast-groups>
    ...
  </core>
</configuration>
```

달리 명시하지 않는 한 다음 매개변수가 필요합니다.

broadcast-group

***name** 속성을 사용하여 브로드캐스트 그룹의 고유 이름을 지정합니다.*

local-bind-address

***UDP** 소켓이 바인딩된 주소입니다. 브로커에 네트워크 인터페이스가 여러 개 있는 경우 브로드캐스트에 사용할 네트워크 인터페이스를 지정해야 합니다. 이 속성을 지정하지 않으면*

소켓이 운영 체제에서 선택한 IP 주소에 바인딩됩니다. UDP 특정 속성입니다.

local-bind-port

데이터그램 소켓이 바인딩된 포트입니다. 대부분의 경우 기본값 -1 을 사용하여 익명 포트를 지정합니다. 이 매개변수는 local-bind-address 와 관련하여 사용됩니다. UDP 특정 속성입니다.

group-address

데이터를 브로드캐스트할 멀티 캐스트 주소입니다. 224.0.0.0 - 239.255.255.255 범위의 D IP 주소입니다. 주소 224.0.0.0 이 예약되었으며 사용할 수 없습니다. UDP 특정 속성입니다.

group-port

브로드캐스트에 사용되는 UDP 포트 번호입니다. UDP 특정 속성입니다.

broadcast-period (선택 사항)

연속 브로드캐스트 사이의 간격(밀리초)입니다. 기본값은 2000밀리초입니다.

Connector-ref

브로드캐스트해야 하는 이전에 구성된 클러스터 커넥터입니다.

4. UDP 검색 그룹을 추가합니다.

검색 그룹은 이 브로커가 다른 브로커에서 커넥터 정보를 수신하는 방법을 정의합니다. 브로커는 커넥터 목록을 유지 관리합니다 (각 브로커마다 하나의 항목). 브로커에서 브로드캐스트를 수신하므로 해당 항목을 업데이트합니다. 브로커에서 오랜 시간 동안 브로드캐스트를 수신하지 못하면 해당 항목을 제거합니다.

이 검색 그룹은 UDP를 사용하여 클러스터의 브로커를 검색합니다.

```
<configuration>
   <core>
      ...
      <discovery-groups>
         <discovery-group name="my-discovery-group">
            <local-bind-address>172.16.9.7</local-bind-address>
            <group-address>231.7.7.7</group-address>
            <group-port>9876</group-port>
            <refresh-timeout>10000</refresh-timeout>
         </discovery-group>
      <discovery-groups>
```

```
      ...
   </core>
</configuration>
```

달리 명시하지 않는 한 다음 매개변수가 필요합니다.

discovery-group

__name__ 속성을 사용하여 검색 그룹의 고유 이름을 지정합니다.

***local-bind-address** (선택 사항)*

브로커가 실행 중인 시스템이 여러 네트워크 인터페이스를 사용하는 경우 검색 그룹이 수신해야 하는 네트워크 인터페이스를 지정할 수 있습니다. __UDP__ 특정 속성입니다.

group-address

수신 대기할 그룹의 멀티 캐스트 주소입니다. 수신 대기하려는 브로드캐스트 그룹의 __group-address__ 와 일치해야 합니다. __UDP__ 특정 속성입니다.

group-port

멀티 캐스트 그룹의 __UDP__ 포트 번호입니다. 수신 대기하려는 브로드캐스트 그룹의 __group-port__ 와 일치해야 합니다. __UDP__ 특정 속성입니다.

***refresh-timeout** (선택 사항)*

해당 브로커의 커넥터 쌍 항목을 목록에서 제거하기 전에 검색 그룹이 특정 브로커에서 마지막 브로드캐스트를 수신한 후 대기하는 시간(밀리초)입니다. 기본값은 __10000__밀리초(__10__초)입니다.

브로드캐스트 그룹의 __broadcast-period__ 보다 훨씬 높은 값으로 설정합니다. 그렇지 않으면 브로커가 여전히 브로드캐스트되어 있어도 목록에서 주기적으로 사라질 수 있습니다 (시간의 약간의 차이로 인해).

5. *클러스터 연결을 생성하고 동적 검색을 사용하도록 구성합니다.*

기본적으로 클러스터 연결은 대칭 토폴로지의 모든 주소에 대한 메시지를 로드 밸런싱합니다.

```
<configuration>
  <core>
    ...
    <cluster-connections>
      <cluster-connection name="my-cluster">
```

```
            <connector-ref>netty-connector</connector-ref>
            <discovery-group-ref discovery-group-name="my-discovery-group"/>
         </cluster-connection>
      </cluster-connections>
      ...
   </core>
</configuration>
```

cluster-connection

name 속성을 사용하여 클러스터 연결의 이름을 지정합니다.

Connector-ref

다른 브로커가 이 계정에 연결하는 방법을 정의하는 커넥터입니다.

discovery-group-ref

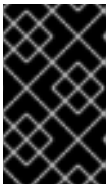
이 브로커가 클러스터의 다른 멤버를 찾는 데 사용해야 하는 검색 그룹입니다. 클러스터가 동적 검색을 사용하는 경우에만 이 속성을 구성해야 합니다.

6. 클러스터 연결에 대한 추가 속성을 구성합니다.

이러한 추가 클러스터 연결 속성에는 대부분의 일반적인 사용 사례에 적합한 기본값이 있습니다. 따라서 기본 동작을 원하지 않는 경우에만 이러한 속성을 구성해야 합니다. 자세한 내용은 부록 C. 클러스터 연결 구성의 내용을 참조하십시오.

7. 클러스터 사용자 및 암호를 만듭니다.

AMQ Broker는 기본 클러스터 인증 정보와 함께 제공되지만 권한이 없는 원격 클라이언트가 이러한 기본 인증 정보를 사용하여 브로커에 연결하지 못하도록 변경해야 합니다.



중요

클러스터 암호는 클러스터의 모든 브로커에서 동일해야 합니다.

```
<configuration>
   <core>
      ...
      <cluster-user>cluster_user</cluster-user>
      <cluster-password>cluster_user_password</cluster-password>
      ...
   </core>
</configuration>
```

8. 추가 브로커마다 이 절차를 반복합니다.

클러스터 구성을 추가 브로커마다 복사할 수 있습니다. 그러나 다른 **AMQ Broker** 데이터 파일(예: 바인딩, 저널, 대규모 메시지 디렉터리)을 복사하지 마십시오. 이러한 파일은 클러스터의 노드 간에 고유해야 합니다. 그렇지 않으면 클러스터가 올바르게 포맷되지 않습니다.

추가 리소스

- **UDP**와 함께 동적 검색을 사용하는 브로커 클러스터 구성의 예는 클러스터형 대기열 예제 를 참조하십시오.

14.2.3. Cryostat 기반 동적 검색을 사용하여 브로커 클러스터 생성

이미 사용자 환경에서 **Cryostat**를 사용하는 경우 이를 사용하여 브로커가 서로 동적으로 검색하는 브로커 클러스터를 만들 수 있습니다.

사전 요구 사항

- **Cryostat**를 설치하고 구성해야 합니다.

Cryostat 구성 파일의 예는 클러스터형-**jgroups** 예제 를 참조하십시오.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **< core>** 요소 내에서 커넥터를 추가합니다.

이 커넥터는 다른 브로커가 이 옵션에 연결하는 데 사용할 수 있는 연결 정보를 정의합니다. 이 정보는 검색 중에 클러스터의 다른 브로커로 전송됩니다.

```
<configuration>
  <core>
    ...
    <connectors>
      <connector name="netty-connector">tcp://localhost:61617</connector>
    </connectors>
    ...
  </core>
</configuration>
```

3. **&lt ;core>** 요소 내에서 **Cryostat** 브로드캐스트 그룹을 추가합니다.

브로드캐스트 그룹을 사용하면 브로커가 클러스터 연결에 대한 정보를 클러스터의 다른 브로커로 내보낼 수 있습니다. 이 브로드캐스트 그룹은 **Cryostat**를 사용하여 연결 설정을 브로드캐스트합니다.

```
<configuration>
   <core>
      ...
      <broadcast-groups>
         <broadcast-group name="my-broadcast-group">
            <jgroups-file>test-jgroups-file_ping.xml</jgroups-file>
            <jgroups-channel>activemq_broadcast_channel</jgroups-channel>
            <broadcast-period>2000</broadcast-period>
            <connector-ref>netty-connector</connector-ref>
         </broadcast-group>
      </broadcast-groups>
      ...
   </core>
</configuration>
```

달리 명시하지 않는 한 다음 매개변수가 필요합니다.

broadcast-group

name 속성을 사용하여 브로드캐스트 그룹의 고유 이름을 지정합니다.

Cryostat-file

Cryostat 채널을 초기화할 **Cryostat** 구성 파일의 이름입니다. 브로커가 로드할 수 있도록 **Java** 리소스 경로에 파일이 있어야 합니다.

jgroups-channel

브로드캐스트를 위해 연결할 **Cryostat** 채널의 이름입니다.

broadcast-period (선택 사항)

연속 브로드캐스트 사이의 간격(밀리초)입니다. 기본값은 **2000**밀리초입니다.

Connector-ref

브로드캐스트해야 하는 이전에 구성된 클러스터 커넥터입니다.

4.

Cryostat 검색 그룹을 추가합니다.

검색 그룹은 커넥터 정보가 수신되는 방법을 정의합니다. 브로커는 커넥터 목록을 유지 관리합니다 (각 브로커마다 하나의 항목). 브로커에서 브로드캐스트를 수신하므로 해당 항목을 업데이트합니다. 브로커에서 오랜 시간 동안 브로드캐스트를 수신하지 못하면 해당 항목을 제거합니다.

이 검색 그룹은 **Cryostat**를 사용하여 클러스터의 브로커를 검색합니다.

```
<configuration>
  <core>
    ...
    <discovery-groups>
      <discovery-group name="my-discovery-group">
        <jgroups-file>test-jgroups-file_ping.xml</jgroups-file>
        <jgroups-channel>activemq_broadcast_channel</jgroups-channel>
        <refresh-timeout>10000</refresh-timeout>
      </discovery-group>
    <discovery-groups>
    ...
  </core>
</configuration>
```

달리 명시하지 않는 한 다음 매개변수가 필요합니다.

discovery-group

name 속성을 사용하여 검색 그룹의 고유 이름을 지정합니다.

Cryostat-file

Cryostat 채널을 초기화할 **Cryostat** 구성 파일의 이름입니다. 브로커가 로드할 수 있도록 **Java** 리소스 경로에 파일이 있어야 합니다.

jgroups-channel

브로드캐스트 수신을 위해 연결할 **Cryostat** 채널의 이름입니다.

refresh-timeout (선택 사항)

해당 브로커의 커넥터 쌍 항목을 목록에서 제거하기 전에 검색 그룹이 특정 브로커에서 마지막 브로드캐스트를 수신한 후 대기하는 시간(밀리초)입니다. 기본값은 **10000**밀리초(**10**초)입니다.

브로드캐스트 그룹의 **broadcast-period** 보다 훨씬 높은 값으로 설정합니다. 그렇지 않

으면 브로커가 여전히 브로드캐스트되어 있어도 목록에서 주기적으로 사라질 수 있습니다 (시간의 약간의 차이로 인해).

5. 클러스터 연결을 생성하고 동적 검색을 사용하도록 구성합니다.

기본적으로 클러스터 연결은 대칭 토폴로지의 모든 주소에 대한 메시지를 로드 밸런싱합니다.

```
<configuration>
  <core>
    ...
    <cluster-connections>
      <cluster-connection name="my-cluster">
        <connector-ref>netty-connector</connector-ref>
        <discovery-group-ref discovery-group-name="my-discovery-group"/>
      </cluster-connection>
    </cluster-connections>
    ...
  </core>
</configuration>
```

cluster-connection

name 속성을 사용하여 클러스터 연결의 이름을 지정합니다.

Connector-ref

다른 브로커가 이 계정에 연결하는 방법을 정의하는 커넥터입니다.

discovery-group-ref

이 브로커가 클러스터의 다른 멤버를 찾는 데 사용해야 하는 검색 그룹입니다. 클러스터가 동적 검색을 사용하는 경우에만 이 속성을 구성해야 합니다.

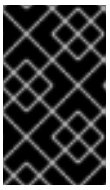
6. 클러스터 연결에 대한 추가 속성을 구성합니다.

이러한 추가 클러스터 연결 속성에는 대부분의 일반적인 사용 사례에 적합한 기본값이 있습니다. 따라서 기본 동작을 원하지 않는 경우에만 이러한 속성을 구성해야 합니다. 자세한 내용은 부록 C. 클러스터 연결 구성의 내용을 참조하십시오.

7. 클러스터 사용자 및 암호를 만듭니다.

AMQ Broker는 기본 클러스터 인증 정보와 함께 제공되지만 권한이 없는 원격 클라이언트가

이러한 기본 인증 정보를 사용하여 브로커에 연결하지 못하도록 변경해야 합니다.



중요

클러스터 암호는 클러스터의 모든 브로커에서 동일해야 합니다.

```
<configuration>
  <core>
    ...
    <cluster-user>cluster_user</cluster-user>
    <cluster-password>cluster_user_password</cluster-password>
    ...
  </core>
</configuration>
```

8. 추가 브로커마다 이 절차를 반복합니다.

클러스터 구성을 추가 브로커마다 복사할 수 있습니다. 그러나 다른 **AMQ Broker** 데이터 파일(예: 바인딩, 저널, 대규모 메시지 디렉터리)을 복사하지 마십시오. 이러한 파일은 클러스터의 노드 간에 고유해야 합니다. 그렇지 않으면 클러스터가 올바르게 포맷되지 않습니다.

추가 리소스

- **Cryostat**와 함께 동적 검색을 사용하는 브로커 클러스터의 예는 **clustered-jgroups 예제** 를 참조하십시오.

14.3. 고가용성 구현

HA(고가용성)를 구현하여 안정성을 개선할 수 있으므로 하나 이상의 브로커가 오프라인 상태인 경우에도 브로커 클러스터가 계속 작동합니다.

HA 구현에는 몇 가지 단계가 필요합니다.

1. **14.2절. “브로커 클러스터 생성”** 에 설명된 대로 **HA** 구현을 위해 브로커 클러스터를 구성합니다.

2. 실시간 백업 그룹을 이해하고 요구 사항에 가장 적합한 **HA** 정책을 선택해야 합니다. **AMQ Broker에서 HA가 작동하는 방법 이해** 를 참조하십시오.

3. 적합한 **HA** 정책을 선택한 경우 클러스터의 각 브로커에서 **HA** 정책을 구성합니다. 다음 내용을 참조하십시오.

- 공유 저장소 고가용성 구성
- 복제 고가용성 구성
- 라이브 전용으로 제한된 고가용성 구성
- 배치된 백업을 사용하여 고가용성 구성

4. 장애 조치를 사용하도록 클라이언트 애플리케이션을 구성합니다.

참고

이후의 경우 고가용성으로 구성된 브로커 클러스터의 문제를 해결해야 하는 경우 클러스터에서 브로커를 실행하는 각 **JVM(Java Virtual Machine)** 인스턴스에 대해 **Garbage Collection(GC)** 로깅을 활성화하는 것이 좋습니다. **JVM**에서 **GC** 로그를 활성화하는 방법을 알아보려면 **JVM**에서 사용하는 **Java Development Kit(JDK)** 버전의 공식 문서를 참조하십시오. **AMQ Broker**에서 지원하는 **JVM** 버전에 대한 자세한 내용은 **Red Hat AMQ 7** 지원 구성 을 참조하십시오.

14.3.1. 고가용성 이해

AMQ Broker에서는 클러스터의 브로커를 라이브 백업 그룹으로 그룹화하여 **HA(**고가용성**)**를 구현합니다. 라이브 백업 그룹에서 라이브 브로커는 백업 브로커에 연결되어 있으며, 실패하는 경우 라이브 브로커를 대신할 수 있습니다. **AMQ Broker**는 라이브 백업 그룹 내에서 장애 조치**(HA** 정책이라고 함**)**를 위한 몇 가지 다른 전략도 제공합니다.

14.3.1.1. 라이브 백업 그룹이 고가용성을 제공하는 방법

AMQ Broker에서는 클러스터의 브로커를 연결하여 **HA(**고가용성**)**를 구현하여 실시간 백업 그룹을 구성합니다. 라이브 백업 그룹은 페일오버 를 제공합니다. 즉, 한 브로커가 실패하면 다른 브로커가 메시지 처리를 인수할 수 있습니다.

라이브 백업 그룹은 하나 이상의 백업 브로커(때때로 슬레이브 브로커라고 함)에 연결된 하나의 라이

브 브로커(때때로 마스터 브로커라고 함)로 구성됩니다. 라이브 브로커는 클라이언트 요청을 제공하며 백업 브로커는 패시브 모드로 대기합니다. 라이브 브로커가 실패하면 백업 브로커가 라이브 브로커를 대체하여 클라이언트가 다시 연결하고 작업을 계속할 수 있습니다.

14.3.1.2. 고가용성 정책

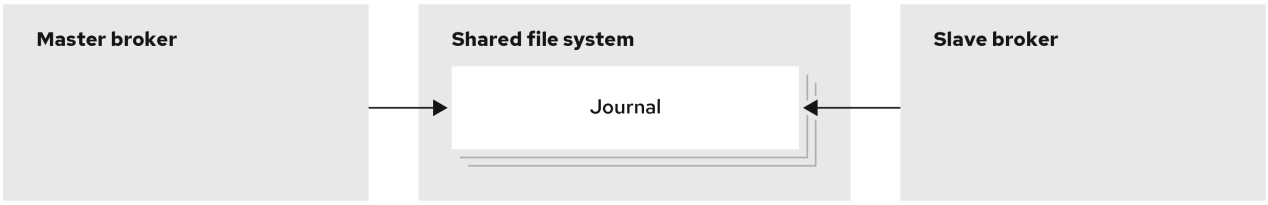
HA(고가용성) 정책은 라이브 백업 그룹에서 장애 조치가 발생하는 방법을 정의합니다. **AMQ Broker**는 다음과 같은 몇 가지 **HA** 정책을 제공합니다.

공유 저장소(권장)

라이브 및 백업 브로커는 메시징 데이터를 공유 파일 시스템의 공통 디렉터리에 저장합니다. 일반적으로 **SAN(Storage Area Network)** 또는 **NFS(Network File System)** 서버. **JDBC** 기반 지속성을 구성한 경우 브로커 데이터를 지정된 데이터베이스에 저장할 수도 있습니다. 공유 저장소를 사용하면 라이브 브로커가 실패하면 백업 브로커가 공유 저장소에서 메시지 데이터를 로드하고 실패한 라이브 브로커를 대신합니다.

대부분의 경우 복제 대신 공유 저장소를 사용해야 합니다. 공유 저장소는 네트워크를 통해 데이터를 복제하지 않으므로 일반적으로 복제보다 성능이 향상됩니다.**Because shared store does not replicate data over the network, it typically provides better performance than replication.** 공유 저장소는 또한 라이브 브로커와 백업이 동시에 활성화되는 네트워크 격리('스플릿'라고도 함) 문제를 방지합니다.

그림 **14.4.** 공유 저장소 고가용성



복제

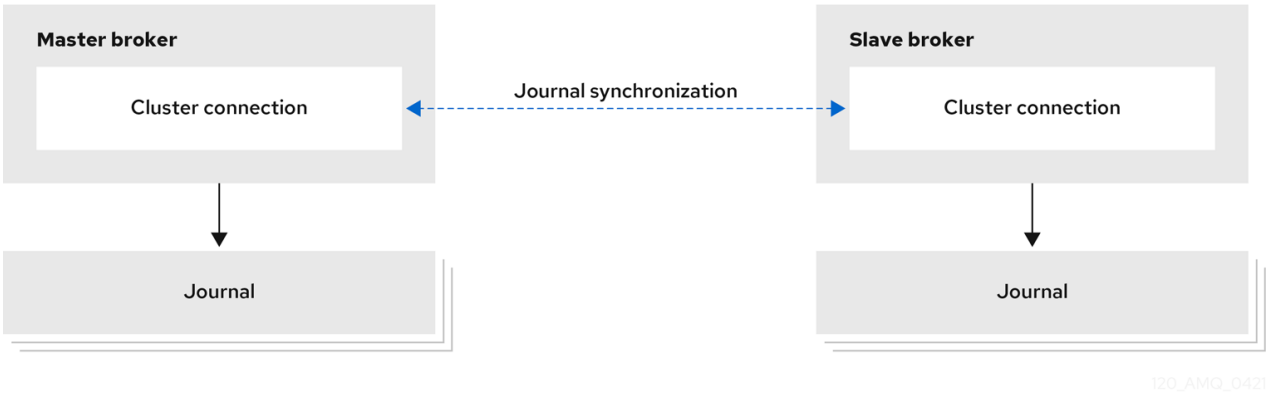
라이브 및 백업 브로커는 네트워크를 통해 메시징 데이터를 지속적으로 동기화합니다. 라이브 브로커가 실패하면 백업 브로커가 동기화된 데이터를 로드하고 실패한 라이브 브로커를 대신 수행합니다.

라이브 및 백업 브로커 간의 데이터 동기화를 통해 라이브 브로커가 실패하면 메시징 데이터가 손실되지 않습니다. 라이브 및 백업 브로커가 처음에 결합되면 라이브 브로커는 기존의 모든 데이터를 네트워크를 통해 백업 브로커에 복제합니다. 이 초기 단계가 완료되면 라이브 브로커는 라이브 브로커가 수신할 때 영구 데이터를 백업 브로커에 복제합니다. 즉, 라이브 브로커가 네트워크에서 떨어지면 백업 브로커는 라이브 브로커가 해당 시점까지 수신한 모든 영구 데이터를 갖습니다.

복제는 네트워크를 통해 데이터를 동기화하므로 라이브 브로커와 해당 백업이 동시에 활성화되

는 네트워크 격리가 발생할 수 있습니다.

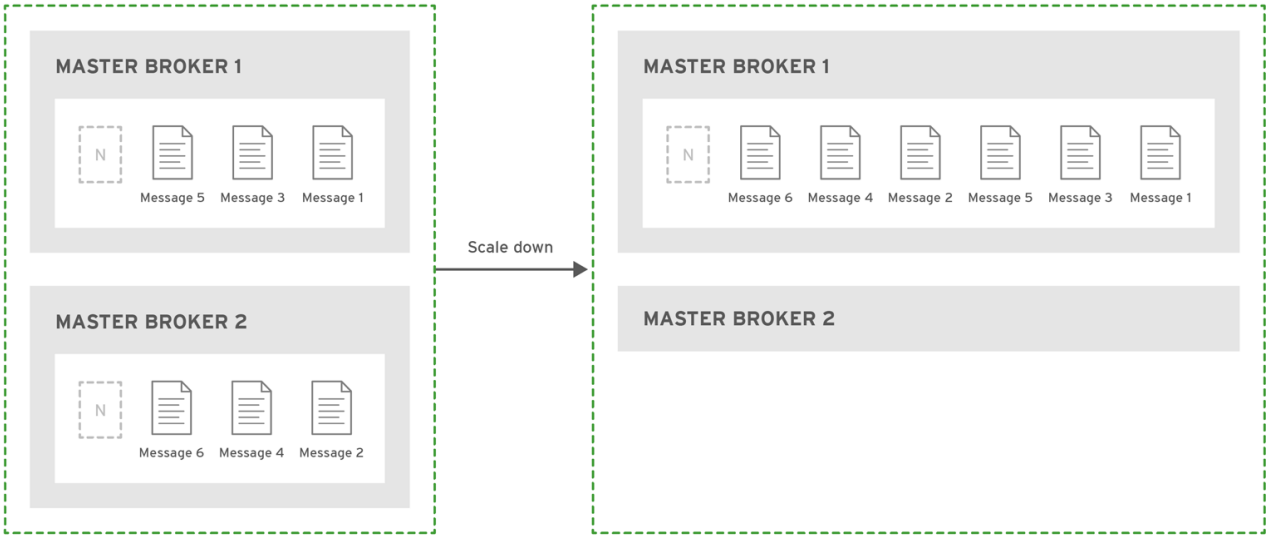
그림 14.5. 복제 고가용성



라이브 전용(제한 HA)

라이브 브로커가 정상적으로 중지되면 메시지와 트랜잭션 상태를 다른 라이브 브로커에 복사한 다음 종료합니다. 그러면 클라이언트가 다른 브로커에 다시 연결하여 메시지를 계속 보내고 받을 수 있습니다.

그림 14.6. 라이브 전용 고가용성



추가 리소스

- 라이브 백업 그룹의 브로커 간에 공유되는 영구 메시지 데이터에 대한 자세한 내용은 6.1절. "저널에 메시지 데이터 유지" 을 참조하십시오.

14.3.1.3. 복제 정책 제한 사항

복제를 사용하여 고가용성을 제공할 때 라이브 및 백업 브로커 둘 다 동시에 존재할 수 있는 위험이 있습니다.

스플릿 뇌는 라이브 브로커와 백업이 연결이 끊어지면 발생할 수 있습니다. 이 경우 라이브 브로커와 백업이 동시에 활성화될 수 있습니다. 이 상황에서 브로커 간에 메시지 복제가 없기 때문에 서로 알 필요 없이 각각 클라이언트와 메시지를 처리합니다. 이 경우 각 브로커는 완전히 다른 저널을 갖습니다. 이 상황에서 복구하는 것은 매우 어려울 수 있으며 경우에 따라 불가능합니다.

- *분할 뇌의 가능성을 제거하려면 공유 저장소 **HA** 정책을 사용하십시오.*
- *복제 **HA** 정책을 사용하는 경우 다음 단계를 수행하여 분할 뇌가 발생할 위험을 줄입니다.*

  *브로커가 **Zoo Cryostat** 코디네이션 서비스를 사용하여 브로커를 조정하려면 최소 **3**개의 노드에 **Zoo Cryostat**를 배포합니다. 브로커가 하나의 **Zoo Cryostat** 노드에 대한 연결이 끊어지면 **3** 개 이상의 노드를 사용하면 라이브 백업 브로커 쌍에 복제 중단이 발생할 때 브로커를 조정할 수 있습니다.*

  *클러스터에서 사용 가능한 다른 브로커를 사용하여 쿼럼 투표를 제공하는 임베디드 브로커 조정을 사용하려면 **3**개 이상의 라이브 백업 쌍을 사용하여 분할 뇌가 발생할 가능성을 줄일 수 있습니다. 세 개 이상의 라이브 백업 쌍을 사용하면 라이브 백업 브로커 쌍에 복제 중단이 발생할 때 발생하는 쿼럼 투표에서 대부분의 결과를 얻을 수 있습니다.*

*복제 **HA** 정책을 사용할 때 고려해야 할 몇 가지 추가 고려 사항은 다음과 같습니다.*

- *라이브 브로커가 실패하고 백업이 라이브로 전환되면 새 백업 브로커가 라이브에 연결되거나 원래 라이브 브로커에 실패할 때까지 추가 복제가 수행되지 않습니다.*
- *라이브 백업 그룹의 백업 브로커가 실패하면 라이브 브로커는 계속 메시지를 제공합니다. 그러나 다른 브로커가 백업으로 추가되거나 원래 백업 브로커가 다시 시작될 때까지 메시지가 복제되지 않습니다. 이 기간 동안 메시지는 라이브 브로커에만 유지됩니다.*
- *브로커가 포함된 브로커 조정을 사용하고 라이브 백업 쌍의 브로커 둘 다 종료되는 경우 메시지 손실을 방지하기 위해 가장 최근의 활성 브로커를 먼저 다시 시작해야 합니다. 가장 최근에 활성 브로커가 백업 브로커인 경우 이 브로커를 마스터 브로커로 수동으로 재구성하여 먼저 다시 시작해야 합니다.*

14.3.2. 공유 저장소 고가용성 구성

공유 저장소 HA(고가용성) 정책을 사용하여 브로커 클러스터에서 HA를 구현할 수 있습니다. 공유 저장소를 사용하면 라이브 및 백업 브로커 모두 공유 파일 시스템의 공통 디렉터리(일반적으로 SAN(Storage Area Network) 또는 NFS(Network File System) 서버에 액세스합니다. JDBC 기반 지속성을 구성한 경우 브로커 데이터를 지정된 데이터베이스에 저장할 수도 있습니다. 공유 저장소를 사용하면 라이브 브로커가 실패하면 백업 브로커가 공유 저장소에서 메시지 데이터를 로드하고 실패한 라이브 브로커를 대신합니다.

일반적으로 SAN은 NFS 서버와 비교하여 더 나은 성능(예: 속도)을 제공하며, 사용 가능한 경우 권장 옵션입니다. NFS 서버를 사용해야 하는 경우 AMQ Broker에서 지원하는 네트워크 파일 시스템에 대한 자세한 내용은 Red Hat AMQ 7 Supported Configurations 를 참조하십시오.

대부분의 경우 복제 대신 공유 저장소 HA를 사용해야 합니다. 공유 저장소는 네트워크를 통해 데이터를 복제하지 않으므로 일반적으로 복제보다 성능이 향상됩니다.Because shared store does not replicate data over the network, it typically provides better performance than replication. 공유 저장소는 또한 라이브 브로커와 백업이 동시에 활성화되는 네트워크 격리('스플릿'라고도 함) 문제를 방지합니다.

참고

공유 저장소를 사용할 때 백업 브로커의 시작 시간은 메시지 저널의 크기에 따라 다릅니다. 백업 브로커가 실패한 라이브 브로커를 인수하면 공유 저장소에서 저널을 로드합니다. 저널에 많은 데이터가 포함된 경우 이 프로세스는 시간이 오래 걸릴 수 있습니다.

14.3.2.1. NFS 공유 저장소 구성

공유 저장소 고가용성을 사용하는 경우 공유 파일 시스템에서 공통 디렉터리를 사용하도록 라이브 및 백업 브로커를 모두 구성해야 합니다. 일반적으로 SAN(Storage Area Network) 또는 NFS(Network File System) 서버를 사용합니다.

다음은 각 브로커 시스템 인스턴스의 NFS 서버에서 내보낸 디렉토리를 마운트할 때 권장되는 몇 가지 구성 옵션입니다.

sync

모든 변경 사항이 디스크에 즉시 플러시되도록 지정합니다.

INTR

서버가 종료되거나 도달할 수 없는 경우 NFS 요청을 중단할 수 있습니다.

noac

속성 캐싱을 비활성화합니다. 여러 클라이언트 간에 특성 캐시 일관성을 얻으려면 이 동작이 필요합니다.

soft

NFS 서버를 사용할 수 없는 경우 서버가 온라인 상태가 될 때까지 기다리는 대신 오류를 보고해야 합니다.

lookupcache=none

조회 캐싱을 비활성화합니다.

timeo=n

NFS 클라이언트(즉, 브로커)가 요청을 재시도하기 전에 NFS 서버의 응답을 기다리는 시간(초)입니다. TCP를 통한 NFS의 경우 기본 timeo 값은 600 (60초)입니다. UDP를 통한 NFS의 경우 클라이언트는 조정 알고리즘을 사용하여 읽기 및 쓰기 요청과 같이 자주 사용되는 요청 유형에 대한 적절한 시간 초과 값을 추정합니다.

retrans=n

NFS 클라이언트가 추가 복구 작업을 시도하기 전에 요청을 재시도하는 횟수입니다. 재전송 옵션을 지정하지 않으면 NFS 클라이언트는 각 요청을 세 번 시도합니다.

중요

timeo 를 구성하고 옵션을 다시 전송할 때 적절한 값을 사용하는 것이 중요합니다. 재전송 값이 5번의 재시도 값과 결합된 기본 시간 초과 시간(60초)으로 인해 AMQ Broker가 NFS 연결 해제를 감지할 때까지 5분 동안 기다릴 수 있습니다.

추가 리소스

- NFS 서버에서 내보낸 디렉토리를 마운트하는 방법을 알아보려면 Red Hat Enterprise Linux 설명서의 mount를 사용하여 NFS 공유 마운트를 참조하십시오.
- AMQ Broker에서 지원하는 네트워크 파일 시스템에 대한 자세한 내용은 Red Hat AMQ 7 지원 구성 을 참조하십시오.

14.3.2.2. 공유 저장소 고가용성 구성

다음 절차에서는 브로커 클러스터의 공유 저장소 고가용성을 구성하는 방법을 보여줍니다.

사전 요구 사항

사전 요구 사항

- 실시간 및 백업 브로커가 공유 스토리지 시스템에 액세스할 수 있어야 합니다.
  - 일반적으로 **SAN(Storage Area Network)** 또는 **NFS(Network File System)** 서버를 사용하여 공유 저장소를 제공합니다. 지원되는 네트워크 파일 시스템에 대한 자세한 내용은 **Red Hat AMQ 7** 지원 구성을 참조하십시오.
  - **JDBC** 기반 지속성을 구성한 경우 지정된 데이터베이스를 사용하여 공유 저장소를 제공할 수 있습니다. **JDBC** 지속성 구성 방법을 알아보려면 **6.2**절. “데이터베이스에 메시지 데이터 유지”를 참조하십시오.

프로세스

1. 클러스터의 브로커를 라이브 백업 그룹으로 그룹화합니다.

   대부분의 경우 라이브 백업 그룹은 라이브 브로커와 백업 브로커의 두 브로커로 구성되어야 합니다. 클러스터에 **6**개의 브로커가 있는 경우 세 개의 라이브 백업 그룹이 필요합니다.

2. 하나의 라이브 브로커와 하나의 백업 브로커로 구성된 첫 번째 라이브 백업 그룹을 생성합니다.

   a. 라이브 브로커의 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

   b. 사용하는 경우:

      i. 공유 저장소를 제공하는 네트워크 파일 시스템은 라이브 브로커의 페이징, 바인딩, 저널 및 대규모 메시지 디렉터리가 백업 브로커가 액세스할 수 있는 공유 위치를 가리키는지 확인합니다.

```
<configuration>
  <core>
    ...
    <paging-directory>../sharedstore/data/paging</paging-directory>
    <bindings-directory>../sharedstore/data/bindings</bindings-directory>
    <journal-directory>../sharedstore/data/journal</journal-directory>
    <large-messages-directory>../sharedstore/data/large-messages</large-
messages-directory>
```

```
        ...
    </core>
</configuration>
```

ii.

*공유 저장소를 제공하는 데이터베이스는 마스터 및 백업 브로커가 모두 동일한 데이터베이스에 연결할 수 있고 **broker.xml** 구성 파일의 **database-store** 요소에 동일한 구성을 지정할 수 있는지 확인합니다. 구성 예는 다음과 같습니다.*

```
<configuration>
  <core>
    <store>
      <database-store>
        <jdbc-connection-url>jdbc:oracle:data/oracle/database-store;create=true</jdbc-connection-url>
        <jdbc-user>ENC(5493dd76567ee5ec269d11823973462f)</jdbc-user>
        <jdbc-password>ENC(56a0db3b71043054269d11823973462f)</jdbc-password>
        <bindings-table-name>BIND_TABLE</bindings-table-name>
        <message-table-name>MSG_TABLE</message-table-name>
        <large-message-table-name>LGE_TABLE</large-message-table-name>
        <page-store-table-name>PAGE_TABLE</page-store-table-name>
        <node-manager-store-table-name>NODE_TABLE<node-manager-store-table-name>
        <jdbc-driver-class-name>oracle.jdbc.driver.OracleDriver</jdbc-driver-class-name>
        <jdbc-network-timeout>10000</jdbc-network-timeout>
        <jdbc-lock-renew-period>2000</jdbc-lock-renew-period>
        <jdbc-lock-expiration>15000</jdbc-lock-expiration>
        <jdbc-journal-sync-period>5</jdbc-journal-sync-period>
      </database-store>
    </store>
  </core>
</configuration>
```

c.

HA 정책에 공유 저장소를 사용하도록 라이브 브로커를 구성합니다.

```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <master>
          <failover-on-shutdown>true</failover-on-shutdown>
        </master>
      </shared-store>
    </ha-policy>
    ...
  </core>
</configuration>
```

failover-on-shutdown

이 브로커가 정상적으로 중지되면 이 속성은 백업 브로커가 라이브 상태로 유지되어야 하는지 여부를 제어합니다.

d. *백업 브로커의 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

e. *사용하는 경우:*

i. *공유 저장소를 제공하는 네트워크 파일 시스템은 백업 브로커의 페이징, 바인딩, 저널 및 대규모 메시지 디렉터리가 라이브 브로커와 동일한 공유 위치를 가리키는지 확인합니다.*

```
<configuration>
  <core>
    ...
    <paging-directory>../sharedstore/data/paging</paging-directory>
    <bindings-directory>../sharedstore/data/bindings</bindings-directory>
    <journal-directory>../sharedstore/data/journal</journal-directory>
    <large-messages-directory>../sharedstore/data/large-messages</large-
messages-directory>
    ...
  </core>
</configuration>
```

ii. *공유 저장소를 제공하는 데이터베이스는 마스터 및 백업 브로커가 모두 동일한 데이터베이스에 연결할 수 있고 **broker.xml** 구성 파일의 **database-store** 요소에 동일한 구성을 지정할 수 있는지 확인합니다.*

f. ***HA** 정책에 공유 저장소를 사용하도록 백업 브로커를 구성합니다.*

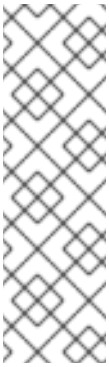
```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <slave>
          <failover-on-shutdown>true</failover-on-shutdown>
          <allow-failback>true</allow-failback>
          <restart-backup>true</restart-backup>
        </slave>
      </shared-store>
    </ha-policy>
    ...
  </core>
</configuration>
```

failover-on-shutdown

이 브로커가 활성화되어 있고 정상적으로 중지되면 이 속성은 백업 브로커(원본 라이브 브로커)가 활성화되어야 하는지 여부를 제어합니다.

allow-failback

페일오버가 발생하고 백업 브로커가 라이브 브로커에 대해 이 속성을 통해 백업 브로커가 원래 라이브 브로커로 장애 조치되고 클러스터에 다시 연결해야 하는지 여부를 제어합니다.



참고

***failback**은 라이브 백업 쌍(단일 백업 브로커와 페어링된 라이브 브로커)을 위한 것입니다. 라이브 브로커가 여러 백업으로 구성된 경우 **failback**이 발생하지 않습니다. 대신 장애 조치(**failover**) 이벤트가 발생하면 백업 브로커가 라이브 상태가 되고 다음 백업이 백업이 됩니다. 원래 라이브 브로커가 다시 온라인 상태가 되면 현재 사용 중인 브로커가 이미 백업되어 있으므로 장애 조치를 시작할 수 없습니다.*

restart-backup

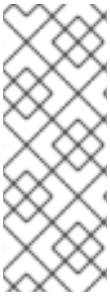
*이 속성은 라이브 브로커로 되돌아간 후 백업 브로커가 자동으로 다시 시작되는지 여부를 제어합니다. 이 속성의 기본값은 **true** 입니다.*

3. *클러스터의 나머지 라이브 백업 그룹에 대해 **2**단계를 반복합니다.*

14.3.3. 복제 고가용성 구성

***HA**(복제 고가용성) 정책을 사용하여 브로커 클러스터에서 **HA**를 구현할 수 있습니다. 복제를 사용하면 라이브 브로커와 백업 브로커 간에 영구 데이터가 동기화됩니다. 라이브 브로커가 오류가 발생하면 메시지 데이터가 백업 브로커에 동기화되고 실패한 라이브 브로커를 대신 수행합니다.*

공유 파일 시스템이 없는 경우 복제를 공유 저장소의 대안으로 사용해야 합니다. 그러나 복제는 라이브 브로커와 백업이 동시에 활성화되는 시나리오를 초래할 수 있습니다.



참고

라이브 및 백업 브로커는 네트워크를 통해 메시징 데이터를 동기화해야 하므로 복제에는 성능 오버헤드가 추가됩니다. 이 동기화 프로세스는 저널 작업을 차단하지만 클라이언트를 차단하지는 않습니다. 데이터 동기화에 대해 저널 작업을 차단할 수 있는 최대 시간을 구성할 수 있습니다.

라이브 백업 브로커 쌍 간의 복제 연결이 중단되면 브로커는 라이브 브로커가 여전히 활성화되어 있는지 또는 백업 브로커에 대한 장애 조치(failover)가 필요한지 여부를 결정하기 위해 조정할 방법이 필요합니다. 이러한 조정을 제공하기 위해 다음 조정 방법 중 하나를 사용하도록 브로커를 구성할 수 있습니다.

- **Apache Zoo Cryostat** 조정 서비스입니다.
- 클러스터의 다른 브로커를 사용하여 쿼럼 투표를 제공하는 임베디드 브로커 조정.

14.3.3.1. 조정 방법 선택

Apache Zoo Cryostat 조정 서비스를 사용하여 브로커 활성화를 조정하는 것이 좋습니다. 조정 방법을 선택할 때 인프라 요구 사항의 차이점과 두 조정 방법 간의 데이터 일관성 관리를 이해하는 것이 유용합니다.

인프라 요구사항

- **Zoo Cryostat** 조정 서비스를 사용하는 경우 단일 라이브 백업 브로커 쌍으로 작동할 수 있습니다. 그러나 한 노드에 연결이 끊어지면 브로커가 계속 작동 할 수 있도록 브로커를 **3** 개 이상의 **Apache Zoo Cryostat** 노드에 연결해야 합니다. 브로커에 조정 서비스를 제공하기 위해 다른 애플리케이션에서 사용하는 기존 **Zoo Cryostat** 노드를 공유할 수 있습니다. **Apache Zoo Cryostat** 설정에 대한 자세한 내용은 **Apache Zoo Cryostat** 설명서를 참조하십시오.
- 클러스터에서 사용 가능한 다른 브로커를 사용하여 쿼럼 투표를 제공하는 임베디드 브로커 조정을 사용하려면 **3**개 이상의 라이브 백업 브로커 쌍이 있어야 합니다. **3**개 이상의 라이브 백업 쌍을 사용하면 라이브 백업 브로커 쌍에 복제 중단이 발생할 때 발생하는 쿼럼 투표에서 대부분의 결과를 얻을 수 있습니다.

데이터 일관성

- **Apache Zoo Cryostat** 조정 서비스를 사용하는 경우 **Zoo Cryostat**는 각 브로커에서 데이터

의 버전을 추적하므로 가장 최신 저널 데이터를 가진 브로커만 브로커가 복제 목적으로 기본 또는 백업 브로커로 구성되어 있는지 여부에 관계없이 라이브 브로커로 활성화할 수 있습니다. 버전 추적은 브로커가 최신 저널로 활성화되고 클라이언트를 제공할 수 있는 가능성을 제거합니다.

- 포함된 브로커 조정을 사용하는 경우 각 브로커에서 데이터의 버전을 추적하여 가장 최신 저널이 있는 브로커만 라이브 브로커가 될 수 있도록 하는 메커니즘이 존재하지 않습니다. 따라서 최신 저널이 있는 브로커가 실시간 상태가 되고 고객에게 서비스를 제공할 수 있으므로 저널에서 다양한 상황이 발생할 수 있습니다.

14.3.3.2. 복제 중단 후 브로커가 조정하는 방법

이 섹션에서는 복제 연결이 중단된 후 두 가지 조정 방법이 어떻게 작동하는지 설명합니다.

Zoo Cryostat 조정 서비스 사용

복제 중단을 관리하기 위해 **Zoo Cryostat** 조정 서비스를 사용하는 경우 두 브로커 모두 여러 **Apache Zoo Cryostat** 노드에 연결되어 있어야 합니다.

- 언제든지 라이브 브로커가 대부분의 **Zoo Cryostat** 노드에 대한 연결을 끊으면 "스플릿 브레인"의 위험을 방지하기 위해 종료됩니다.

- 언제든지 백업 브로커가 대부분의 **Zoo Cryostat** 노드에 대한 연결을 끊으면 복제 데이터 수신을 중지하고 대부분의 **Zoo Cryostat** 노드에 다시 연결하기 전에 다시 연결할 수 있을 때까지 기다립니다. 연결이 대부분의 **Zoo Cryostat** 노드로 복원되면 백업 브로커는 데이터를 삭제하고 복제할 라이브 브로커를 검색하거나 현재 데이터로 라이브 브로커가 될 수 있는지 여부를 결정하는 데 사용합니다.

Zookeeper는 다음 제어 메커니즘을 사용하여 장애 조치 프로세스를 관리합니다.

- 언제든지 단일 라이브 브로커에서만 소유할 수 있는 공유 리스 잠금입니다.

- 브로커 데이터의 최신 버전을 추적하는 활성화 순서 카운터입니다. 각 브로커는 **NodeID**와 함께 서버 잠금 파일에 저장된 로컬 카운터에서 저널 데이터의 버전을 추적합니다. 또한 라이브 브로커는 **Zoo Cryostat**의 조정된 활성화 시퀀스 카운터로 버전을 공유합니다.

라이브 브로커와 백업 브로커 간의 복제 연결이 손실되면 라이브 브로커는 로컬 활성화 시퀀스 카운터

*값과 **Zoo Cryostat**의 조정된 활성화 시퀀스 카운터 값을 **1** 로 늘려 최신 데이터가 있음을 알립니다. 이제 백업 브로커의 데이터가 오래된 것으로 간주되며 복제 연결이 복원되고 최신 데이터가 동기화될 때까지 브로커가 라이브 브로커가 될 수 없습니다.*

*복제 연결이 손실된 후 백업 브로커는 라이브 브로커가 **Zoo Cryostat** 잠금을 소유하고 있고 **Zoo Cryostat**의 조정된 활성화 시퀀스 카운터가 로컬 카운터 값과 일치하는지 확인합니다.*

- *라이브 브로커가 잠금을 소유하는 경우 백업 브로커는 복제 연결이 손실되었을 때 **Zoo Cryostat**의 활성화 시퀀스 카운터가 라이브 브로커에 의해 업데이트되었음을 감지합니다. 이는 백업 브로커가 장애 조치를 시도하지 않도록 라이브 브로커가 실행 중임을 나타냅니다.*
- *라이브 브로커가 잠금을 소유하지 않은 경우 라이브 브로커는 활성 상태가 아닙니다. 백업 브로커의 활성화 시퀀스 카운터 값이 **Zoo Cryostat**의 조정된 활성화 순서 카운터 값과 동일하면 백업 브로커에 최신 데이터가 있음을 나타냅니다.*
- *라이브 브로커가 잠금을 소유하고 있지 않지만 백업 브로커의 활성화 순서 카운터 값이 **Zoo Cryostat**의 카운터 값보다 작으면 백업 브로커의 데이터는 최신 상태가 아니며 백업 브로커가 실패할 수 없습니다.*

포함된 브로커 조정 사용

라이브 백업 브로커 쌍이 포함된 브로커 조정을 사용하여 복제 중단을 조정하는 경우 다음 두 가지 유형의 쿼럼 투표를 시작할 수 있습니다.

*표 **14.1.** 쿼럼 투표*

투표 유형	설명	이니시에이터	필수 구성	참가자	투표 결과에 따른 조치

투표 유형	설명	이니시에이터	필수 구성	참가자	투표 결과에 따른 조치
백업 투표	백업 브로커가 라이브 브로커에 대한 복제 연결을 끊으면 백업 브로커는 이 투표 결과에 따라 시작할지 여부를 결정합니다.	백업 브로커	없음. 백업 브로커는 복제 파트너에 대한 연결이 끊어지면 백업 투표가 자동으로 수행됩니다. 그러나 다음 매개변수에 사용자 지정 값을 지정하여 백업 투표 속성을 제어할 수 있습니다. • **quorum-vote-wait** • **vote-retries** • **vote-retry-wait**	클러스터의 기타 라이브 브로커	백업 브로커는 클러스터의 다른 라이브 브로커에서 대다수(즉, *쿼럼*) 투표를 수신하여 복제 파트너를 더 이상 사용할 수 없음을 나타내는 경우 시작됩니다.
라이브 투표	라이브 브로커가 복제 파트너에 대한 연결이 끊어지면 라이브 브로커는 이 투표를 기반으로 계속 실행할지 여부를 결정합니다.	라이브 브로커	라이브 투표는 라이브 브로커가 복제 파트너와의 연결이 끊어지고 **vote-on-replication 실패가 true** 로 설정된 경우 발생합니다. 활성 상태가 된 백업 브로커는 라이브 브로커로 간주되며 실시간 투표를 시작할 수 있습니다.	클러스터의 기타 라이브 브로커	클러스터 **의 다른 라이브 브로커에서 투표를 받지 못하면** 라이브 브로커가 종료되며 클러스터 연결이 여전히 활성화되어 있음을 나타냅니다.

중요

다음은 브로커 클러스터의 구성에 쿼럼 투표 동작에 미치는 영향에 대해 알아야 할 몇 가지 중요한 사항입니다.

- *쿼럼 투표가 성공하려면 클러스터 크기가 대부분의 결과를 얻을 수 있어야 합니다. 따라서 클러스터에는 세 개 이상의 라이브 백업 브로커 쌍이 있어야 합니다.*

- *클러스터에 추가하는 라이브 백업 브로커 쌍이 많을수록 클러스터의 전반적인 내결함성을 높일 수 있습니다. 예를 들어, 세 개의 라이브 백업 쌍이 있다고 가정합니다. 전체 라이브 백업 쌍이 손실되면 나머지 두 개의 라이브 백업 쌍이 대부분의 경우 후속 쿼럼 투표를 수행할 수 없습니다. 이 경우 클러스터의 추가 복제 중단으로 인해 라이브 브로커가 종료되고 백업 브로커가 시작되지 않을 수 있습니다. 클러스터(예: 5개의 브로커 쌍)를 사용하여 클러스터를 구성하면 클러스터에 두 개 이상의 오류가 발생할 수 있으며 대부분의 경우 쿼럼 투표로 인한 결과가 발생합니다.*

- *클러스터의 라이브 백업 브로커 쌍 수를 의도적으로 줄이는 경우 대부분의 투표에 대해 이전에 설정한 임계값이 자동으로 감소되지 않습니다. 이 기간 동안 복제 연결 손실에 의해 트리거되는 쿼럼 투표는 성공할 수 없으므로 클러스터가 뇌를 분할하는 데 더 취약해집니다. 클러스터에서 쿼럼 투표의 최대 임계값을 다시 계산하도록 하려면 먼저 클러스터에서 제거 중인 라이브 백업 쌍을 종료합니다. 그런 다음 클러스터에서 나머지 라이브 백업 쌍을 다시 시작합니다. 나머지 브로커를 모두 다시 시작하면 클러스터에서 쿼럼 투표 임계값을 다시 계산합니다.*

14.3.3.3. Zoo Cryostat 조정 서비스를 사용하여 브로커 클러스터의 복제 구성

*두 브로커 모두에 대해 **Apache Zoo Cryostat** 조정 서비스를 사용하는 라이브 백업 쌍으로 동일한 복제 구성을 지정해야 합니다. 그런 다음 브로커는 기본 브로커이고 백업 브로커인 브로커를 결정하도록 조정합니다.*

사전 요구 사항

- *하나의 노드에 대한 연결이 끊어지면 브로커가 계속 작동 할 수 있도록 **3** 개 이상의 **Apache Zoo Cryostat** 노드.*

- *브로커 시스템에는 유사한 하드웨어 사양이 있습니다. 즉, 시스템이 라이브 브로커를 실행하고 언제든지 백업 브로커를 실행하는 기본 설정이 없습니다.*

- ***Zookeeper**에는 일시 중지 시간이 **Zoo Cryostat** 서버 틱 시간보다 훨씬 적도록 충분한 리소*

스가 있어야 합니다. 브로커의 예상 로드에 따라 브로커와 **Zoo Cryostat** 노드가 동일한 노드를 공유할 수 있는지 신중하게 고려하십시오. 자세한 내용은 **https://zookeeper.apache.org/** 을 참조하십시오.

프로세스

1. **live-backup** 쌍의 두 브로커 모두에 대해 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 쌍의 두 브로커에 대해 동일한 복제 구성을 구성합니다. 예를 들면 다음과 같습니다.

```
<configuration>
  <core>
    ...
    <ha-policy>
      <replication>
        <primary>
          <coordination-id>production-001</coordination-id>
          <manager>
            <properties>
              <property key="connect-string"
value="192.168.1.10:6666,192.168.2.10:6667,192.168.3.10:6668"/>
            </properties>
          </manager>
        </primary>
      </replication>
    </ha-policy>
    ...
  </core>
</configuration>
```

기본 설정

브로커 조정 결과에 따라 브로커 중 하나가 기본 브로커일 수 있음을 나타내도록 복제 유형을 **primary**로 구성합니다.

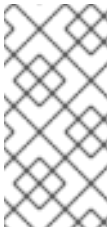
reconcile-id

라이브 백업 쌍의 두 브로커에 대한 공통 문자열 값을 지정합니다. 동일한 **Coordination-id** 문자열을 사용하는 브로커와 함께 활성화를 조정합니다. 조정 프로세스 중에 두 브로커는 **Coordination-id** 문자열을 노드 **Id**로 사용하고 **Zoo Cryostat**에서 잠금을 가져옵니다. 잠금을 확보하고 최신 데이터가 있는 첫 번째 브로커는 라이브 브로커로 시작되고 다른 브로커가 백업됩니다.

속성

Zoo Cryostat 노드에 대한 연결 세부 정보를 제공하기 위해 키-값 쌍 세트를 지정할 수 있는 속성 요소를 지정합니다.

표 14.2. Zookeeper 연결 세부 정보

키	현재의
연결 문자열	Zoo Cryostat 노드의 IP 주소 및 포트 번호의 쉼표로 구분된 목록을 지정합니다. 예를 들어 **value="192.168.1.10:6666,192.168.2.10:6667,192.168.3.10:6668"**.
session-ms	대다수의 Zoo Cryostat 노드에 대한 연결이 끊어진 후 브로커가 종료되기 전에 대기하는 기간입니다. 기본값은 **18000** ms입니다. 유효한 값은 2회에서 20배 사이입니다. Zoo Cryostat 서버의 틱 시간.  참고 가비지 컬렉션에 대한 Zoo Cryostat 일시 중지 시간은 Zoo Cryostat 하트비트가 안정적으로 작동할 수 있도록 **세션-ms** 속성 값의 0.33 미만이어야 합니다. 일시 중지 시간이 이 제한보다 작도록 할 수 없는 경우 각 브로커에 대한 **session-ms** 속성 값을 늘리고 느린 페일오버를 수락합니다.  중요 브로커 복제 파트너는 2초마다 자동으로 "핑" 패킷을 교환하여 파트너 브로커가 사용 가능한지 확인합니다. 백업 브로커가 라이브 브로커로부터 응답을 받지 못하면 백업은 브로커의 연결 시간-투-라이브(ttl)가 만료될 때까지 응답을 기다립니다. 기본 connection-ttl은 **60000** ms이므로 백업 브로커가 60초 후에 장애 조치를 시도합니다. 더 빠른 장애 조치를 허용하려면 connection-ttl 값을 **session-ms** 속성 값과 유사한 값으로 설정하는 것이 좋습니다. 새 connection-ttl을 설정하려면 **connection-ttl-override** 속성을 구성합니다.
네임스페이스(선택 사항)	브로커가 다른 애플리케이션과 Zoo Cryostat 노드를 공유하는 경우, 브로커에 조정 서비스를 제공하는 파일을 저장하기 위해 Zoo Cryostat 네임스페이스를 만들 수 있습니다. 두 브로커에 대해 라이브 백업 쌍에 동일한 네임스페이스를 지정해야 합니다.

3. ***브로커에 대한 추가 HA 속성을 구성합니다.***

이러한 추가 HA 속성에는 대부분의 일반적인 사용 사례에 적합한 기본값이 있습니다. 따라서 기본 동작을 원하지 않는 경우에만 이러한 속성을 구성해야 합니다. 자세한 내용은 부록 F. 추가 복제 고가용성 구성입니다.의 내용을 참조하십시오.

4. ***1~3단계를 반복하여 클러스터의 추가 라이브 백업 브로커 쌍을 구성합니다.***

추가 리소스

- HA에 복제를 사용하는 브로커 클러스터의 예는 HA 예제 를 참조하십시오.

- 노드 ID에 대한 자세한 내용은 노드 ID 이해를 참조하십시오.

14.3.3.4. 포함된 브로커 조정을 사용하여 복제 고가용성을 위해 브로커 클러스터 구성

임베디드 브로커 조정을 사용하는 복제에는 "스플릿 브레인"의 위험을 줄이지만 제거하지는 않는 최소 3개의 라이브 백업 쌍이 필요합니다.

다음 절차에서는 6브로커 클러스터에 대해 HA(복제 고가용성)를 구성하는 방법을 설명합니다. 이 토폴로지에서 6개의 브로커는 세 개의 라이브 백업 쌍으로 그룹화됩니다. 세 개의 라이브 브로커는 각각 전용 백업 브로커와 페어링됩니다.

사전 요구 사항

- 브로커가 6개 이상인 브로커 클러스터가 있어야 합니다.

  6개의 브로커는 세 개의 라이브 백업 쌍으로 구성됩니다. 클러스터에 브로커를 추가하는 방법에 대한 자세한 내용은 14장. 브로커 클러스터 설정 을 참조하십시오.

프로세스

1. 클러스터의 브로커를 라이브 백업 그룹으로 그룹화합니다.

   대부분의 경우 라이브 백업 그룹은 라이브 브로커와 백업 브로커의 두 브로커로 구성되어야 합니다. 클러스터에 6개의 브로커가 있는 경우 세 개의 라이브 백업 그룹이 필요합니다.

2. 하나의 라이브 브로커와 하나의 백업 브로커로 구성된 첫 번째 라이브 백업 그룹을 생성합니다.

   a. 라이브 브로커의 < broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.

   b. HA 정책에 복제를 사용하도록 라이브 브로커를 구성합니다.

```
<configuration>
  <core>
```

```
      ...
      <ha-policy>
         <replication>
            <master>
               <check-for-live-server>true</check-for-live-server>
               <group-name>my-group-1</group-name>
               <vote-on-replication-failure>true</vote-on-replication-failure>
               ...
            </master>
         </replication>
      </ha-policy>
      ...
   </core>
</configuration>
```

check-for-live-server

라이브 브로커가 실패하면 이 속성은 클라이언트가 재시작 시 실패해야 하는지 여부를 제어합니다.

이 속성을 **true** 로 설정하면 이전 장애 조치 후 라이브 브로커가 다시 시작되면 동일한 노드 **ID**가 있는 클러스터에서 다른 브로커를 검색합니다. 라이브 브로커가 동일한 노드 **ID**를 가진 다른 브로커를 발견하면 라이브 브로커가 실패할 때 백업 브로커가 성공적으로 시작되었음을 나타냅니다. 이 경우 라이브 브로커는 해당 데이터를 백업 브로커와 동기화합니다. 그런 다음 라이브 브로커는 백업 브로커를 종료하도록 요청합니다. 백업 브로커가 실패에 대해 구성된 경우 아래와 같이 백업 브로커가 종료됩니다. 그런 다음 라이브 브로커는 활성 역할을 재개하고 클라이언트는 다시 연결합니다.

주의

라이브 브로커에서 **check-for-live-server** 를 **true** 로 설정하지 않으면 이전 장애 조치 후 라이브 브로커를 다시 시작할 때 중복 메시징 처리가 발생할 수 있습니다. 특히 이 속성이 **false** 로 설정된 라이브 브로커를 다시 시작하면 라이브 브로커는 백업 브로커와 데이터를 동기화하지 않습니다. 이 경우 라이브 브로커는 백업 브로커가 이미 처리 한 것과 동일한 메시지를 처리하여 중복을 유발할 수 있습니다.

group-name

이 라이브 백업 그룹의 이름(선택 사항). 라이브 백업 그룹을 만들려면 라이브 및 백업 브로커를 동일한 그룹 이름으로 구성해야 합니다. 그룹 이름을 지정하지 않으면 라이브 브로커와 백업 브로커를 복제할 수 있습니다.

vote-on-replication-failure

이 속성은 라이브 브로커가 중단된 복제 연결이 발생할 경우 라이브 투표라는 쿼럼 투표를 시작하는지 여부를 제어합니다.

라이브 투표는 라이브 브로커가 중단된 복제 연결의 원인인지 여부를 결정하는 방법입니다. 투표 결과에 따라 라이브 브로커는 실행 중이거나 종료됩니다.

중요

*쿼럼 투표가 성공하려면 클러스터 크기가 대부분의 결과를 얻을 수 있어야 합니다. 따라서 복제 **HA** 정책을 사용할 때 클러스터에는 세 개 이상의 라이브 백업 브로커 쌍이 있어야 합니다.*

클러스터에서 더 많은 브로커 쌍을 구성할수록 클러스터의 전반적인 내결함성을 높일 수 있습니다. 예를 들어 라이브 백업 브로커 쌍이 세 개 있다고 가정합니다. 전체 라이브 백업 쌍에 대한 연결이 끊어지면 나머지 두 라이브 백업 쌍이 더 이상 쿼럼 투표를 통해 대부분의 결과를 얻을 수 없습니다. 이 경우 후속 복제 중단으로 인해 라이브 브로커가 종료될 수 있으며 백업 브로커가 시작되지 않을 수 있습니다. 클러스터(예: 5개의 브로커 쌍)를 사용하여 클러스터를 구성하면 클러스터에 두 개 이상의 오류가 발생할 수 있으며 대부분의 경우 쿼럼 투표로 인한 결과가 발생합니다.

c.

*라이브 브로커의 추가 **HA** 속성을 구성합니다.*

*이러한 추가 **HA** 속성에는 대부분의 일반적인 사용 사례에 적합한 기본값이 있습니다. 따라서 기본 동작을 원하지 않는 경우에만 이러한 속성을 구성해야 합니다. 자세한 내용은 부록 **F.** 추가 복제 고가용성 구성입니다.의 내용을 참조하십시오.*

d.

*백업 브로커의 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

e.

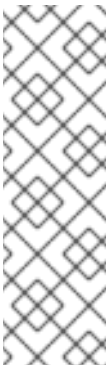
***HA** 정책에 복제를 사용하도록 백업 브로커를 구성합니다.*

```
<configuration>
  <core>
    ...
    <ha-policy>
      <replication>
        <slave>
          <allow-failback>true</allow-failback>
          <group-name>my-group-1</group-name>
          <vote-on-replication-failure>true</vote-on-replication-failure>
```

```
            ...
          </slave>
        </replication>
      </ha-policy>
      ...
    </core>
</configuration>
```

allow-failback

페일오버가 발생하고 백업 브로커가 라이브 브로커에 대해 이 속성을 통해 백업 브로커가 원래 라이브 브로커로 장애 조치되고 클러스터에 다시 연결해야 하는지 여부를 제어합니다.



참고

failback은 라이브 백업 쌍(단일 백업 브로커와 페어링된 라이브 브로커)을 위한 것입니다. 라이브 브로커가 여러 백업으로 구성된 경우 **failback**이 발생하지 않습니다. 대신 장애 조치(**failover**) 이벤트가 발생하면 백업 브로커가 라이브 상태가 되고 다음 백업이 백업이 됩니다. 원래 라이브 브로커가 다시 온라인 상태가 되면 현재 사용 중인 브로커가 이미 백업되어 있으므로 장애 조치를 시작할 수 없습니다.

group-name

이 라이브 백업 그룹의 이름(선택 사항). 라이브 백업 그룹을 만들려면 라이브 및 백업 브로커를 동일한 그룹 이름으로 구성해야 합니다. 그룹 이름을 지정하지 않으면 라이브 브로커와 백업 브로커를 복제할 수 있습니다.

vote-on-replication-failure

이 속성은 라이브 브로커가 중단된 복제 연결이 발생할 경우 라이브 투표라는 쿼럼 투표를 시작하는지 여부를 제어합니다. 활성 상태가 된 백업 브로커는 라이브 브로커로 간주되며 실시간 투표를 시작할 수 있습니다.

라이브 투표는 라이브 브로커가 중단된 복제 연결의 원인인지 여부를 결정하는 방법입니다. 투표 결과에 따라 라이브 브로커는 실행 중이거나 종료됩니다.

f.

(선택 사항) 백업 브로커가 시작하는 쿼럼 투표를 구성합니다.

```
<configuration>
  <core>
    ...
    <ha-policy>
      <replication>
```

```
            <slave>
            ...
               <vote-retries>12</vote-retries>
               <vote-retry-wait>5000</vote-retry-wait>
            ...
            </slave>
         </replication>
      </ha-policy>
      ...
   </core>
</configuration>
```

vote-retries

이 속성은 백업 브로커가 백업 브로커를 시작할 수 있는 대부분의 결과를 수신하기 위해 백업 브로커가 쿼럼 투표를 재시도하는 횟수를 제어합니다.

vote-retry-wait

이 속성은 쿼럼 투표의 각 재시도 사이에 백업 브로커가 대기하는 데 걸리는 시간(밀리초)을 제어합니다.

g. 백업 브로커에 대한 추가 **HA** 속성을 구성합니다.

이러한 추가 **HA** 속성에는 대부분의 일반적인 사용 사례에 적합한 기본값이 있습니다. 따라서 기본 동작을 원하지 않는 경우에만 이러한 속성을 구성해야 합니다. 자세한 내용은 부록 **F.** 추가 복제 고가용성 구성입니다.의 내용을 참조하십시오.

3. 클러스터의 추가 라이브 백업 그룹에 대해 **2**단계를 반복합니다.

클러스터에 **6**개의 브로커가 있는 경우 이 절차를 두 번 더 반복합니다. 나머지 라이브 백업 그룹에 대해 한 번 더 반복합니다.

추가 리소스

- **HA**에 복제를 사용하는 브로커 클러스터의 예는 **HA** 예제 를 참조하십시오.
- 노드 **ID**에 대한 자세한 내용은 노드 **ID** 이해를 참조하십시오.

14.3.4. 라이브 전용으로 제한된 고가용성 구성

라이브 전용 **HA** 정책을 사용하면 메시지를 손실하지 않고 클러스터에서 브로커를 종료할 수 있습니

다. 라이브 전용을 사용하면 라이브 브로커가 정상적으로 중지되면 메시지와 트랜잭션 상태를 다른 라이브 브로커에 복사한 다음 종료합니다. 그러면 클라이언트가 다른 브로커에 다시 연결하여 메시지를 계속 보내고 받을 수 있습니다.

라이브 전용 **HA** 정책은 브로커가 정상적으로 중지되는 경우에만 케이스를 처리합니다. 예기치 않은 브로커 오류를 처리하지 않습니다.

라이브 전용 **HA**는 메시지 손실을 방지하지만 메시지 순서를 유지하지 못할 수 있습니다. 라이브 전용 **HA**로 구성된 브로커가 중지되면 다른 브로커의 대기열 끝에 메시지가 추가됩니다.

참고

브로커가 축소할 준비가 되면 새 브로커가 메시지를 처리할 준비가 되어 있는지 알리는 연결을 해제하기 전에 클라이언트에 메시지를 보냅니다. 그러나 클라이언트는 초기 브로커가 축소된 후에만 새 브로커에 다시 연결해야 합니다. 이렇게 하면 클라이언트가 다시 연결할 때 큐 또는 트랜잭션과 같은 모든 상태를 다른 브로커에서 사용할 수 있습니다. 일반적인 재연결 설정은 클라이언트가 다시 연결할 때 적용되므로 축소하는 데 필요한 시간을 처리할 수 있을 만큼 이러한 높은 설정을 설정해야 합니다.

다음 절차에서는 축소를 위해 클러스터의 각 브로커를 구성하는 방법을 설명합니다. 이 절차를 완료하면 브로커가 정상적으로 중지될 때마다 메시지와 트랜잭션 상태를 클러스터의 다른 브로커에 복사합니다.

프로세스

1. 첫 번째 브로커의 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. 라이브 전용 **HA** 정책을 사용하도록 브로커를 구성합니다.

```
<configuration>
  <core>
    ...
    <ha-policy>
      <live-only>
      </live-only>
    </ha-policy>
    ...
  </core>
</configuration>
```

3. 브로커 클러스터를 축소하는 방법을 구성합니다.

이 브로커를 축소해야 하는 브로커 또는 브로커 그룹을 지정합니다.

표 14.3. 브로커 클러스터를 축소하는 방법

축소를...로 축소하려면 다음을 수행합니다.	이 작업을 수행...

클러스터의 특정 브로커	축소할 브로커의 커넥터를 지정합니다.
	
	`<live-only>`
	`  <scale-down>`
	`  <connectors>`
	`  <connector-ref>broker1-connector</connector-ref>`
	`  </connectors>`
	`  </scale-down>`
	`</live-only>`

클러스터의 브로커	브로커 클러스터의 검색 그룹을 지정합니다.
	
	`<live-only>`
	`  <scale-down>`
	`  <discovery-group-ref discovery-group-name="my-discovery-group"/>`
	`  </scale-down>`
	`</live-only>`

특정 브로커 그룹의 브로커	브로커 그룹을 지정합니다.
	
	`<live-only>`
	`  <scale-down>`
	`  <group-name>my-group-name</group-name>`
	`  </scale-down>`
	`</live-only>`

4. *클러스터의 나머지 브로커에 대해 이 절차를 반복합니다.*

추가 리소스

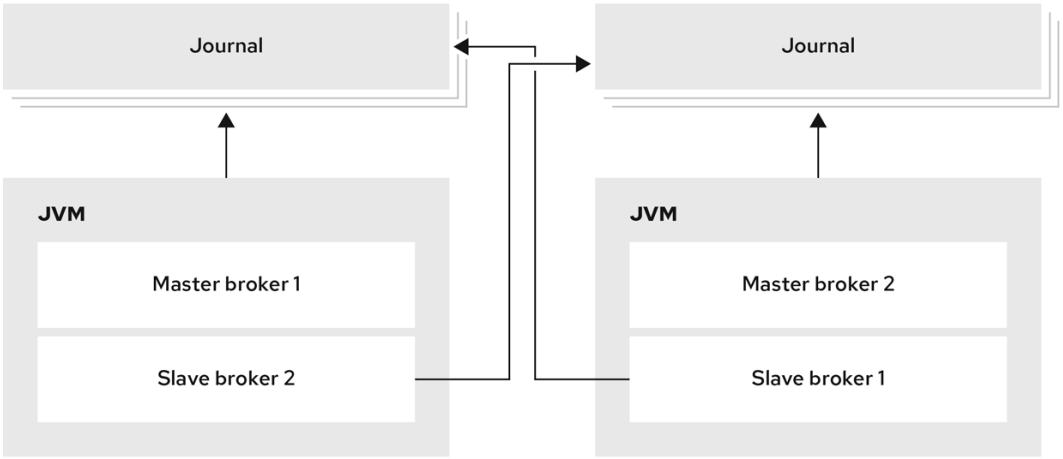
- *실시간 전용을 사용하여 클러스터를 확장하는 브로커 클러스터의 예는 스케일 다운 예제 를 참조하십시오.*

14.3.5. 배치된 백업을 사용하여 고가용성 구성

실시간 백업 그룹을 구성하는 대신 다른 라이브 브로커와 동일한 JVM에 백업 브로커를 배치할 수 있습니다. 이 구성에서 각 라이브 브로커는 다른 라이브 브로커를 요청하여 JVM에서 백업 브로커를 생성하고

시작하도록 구성됩니다.

그림 14.7. 배치된 라이브 및 백업 브로커



120_AMQ_1120

*공유 저장소 또는 복제와 함께 **HA**(고가용성) 정책으로 **colocation**을 사용할 수 있습니다. 새 백업 브로커는 이를 생성하는 라이브 브로커의 구성을 상속합니다. 백업 이름은 **colocated_backup_n** 으로 설정됩니다. 여기서 **n** 은 라이브 브로커가 생성한 백업 수입니다.*

*또한 백업 브로커는 이를 생성하는 라이브 브로커의 커넥터 및 어셉터에 대한 구성을 상속합니다. 기본적으로 포트 오프셋 **100**은 각각에 적용됩니다. 예를 들어 라이브 브로커에 포트 **616**에 대한 수락자가 있는 경우 생성된 첫 번째 백업 브로커는 포트 **61716**을 사용하면 두 번째 백업은 **61816**을 사용합니다.*

*저널, 대용량 메시지 및 페이징의 디렉터리는 선택한 **HA** 정책에 따라 설정됩니다. 공유 저장소를 선택하는 경우 요청 브로커는 대상 브로커에 사용할 디렉터리를 알립니다. 복제를 선택하면 디렉터리는 생성 브로커에서 상속되고 새 백업의 이름이 추가됩니다.*

*이 절차에서는 클러스터의 각 브로커가 공유 저장소 **HA**를 사용하고 클러스터에 있는 다른 브로커와 생성 및 배치되도록 백업을 요청합니다.*

프로세스

1. *첫 번째 브로커의 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

2. ***HA** 정책 및 공동 배치를 사용하도록 브로커를 구성합니다.*

   *이 예에서 브로커는 공유 저장소 **HA** 및 **colocation**으로 구성됩니다.*

```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <colocated>
          <request-backup>true</request-backup>
          <max-backups>1</max-backups>
          <backup-request-retries>-1</backup-request-retries>
          <backup-request-retry-interval>5000</backup-request-retry-interval/>
          <backup-port-offset>150</backup-port-offset>
          <excludes>
            <connector-ref>remote-connector</connector-ref>
          </excludes>
          <master>
            <failover-on-shutdown>true</failover-on-shutdown>
          </master>
          <slave>
            <failover-on-shutdown>true</failover-on-shutdown>
            <allow-failback>true</allow-failback>
            <restart-backup>true</restart-backup>
          </slave>
        </colocated>
      </shared-store>
    </ha-policy>
    ...
  </core>
</configuration>
```

request-backup

*이 속성을 **true** 로 설정하면 이 브로커는 클러스터의 다른 라이브 브로커가 생성하도록 백업 브로커를 요청합니다.*

max-backups

*이 브로커가 생성할 수 있는 백업 브로커 수입니다. 이 속성을 **0** 으로 설정하면 이 브로커는 클러스터의 다른 브로커의 백업 요청을 허용하지 않습니다.*

backup-request-retries

*이 브로커가 백업 브로커를 생성하도록 요청해야 하는 횟수입니다. 기본값은 **-1** 이며, 이는 무제한 연결을 의미합니다.*

backup-request-retry-interval

*브로커가 백업 브로커를 생성하기 위해 요청을 재시도하기 전에 기다려야 하는 시간(밀리초)입니다. 기본값은 **5000** 초 또는 **5**초입니다.*

backup-port-offset

새 백업 브로커에 대한 어셉터 및 커넥터에 사용할 포트 오프셋입니다. 이 브로커가 클

*러스터의 다른 브로커에 대한 백업을 생성하라는 요청을 수신하는 경우 이 양에 따라 포트 오프셋을 사용하여 백업 브로커를 생성합니다. 기본값은 **100**입니다.*

***excludes** (선택 사항)*

*백업 포트 오프셋에서 커넥터를 제외합니다. 백업 포트 오프셋에서 제외해야 하는 외부 브로커에 대한 커넥터를 구성한 경우 각 커넥터에 대해 **< connector-ref >**를 추가합니다.*

master

이 브로커의 공유 저장소 또는 복제 페일오버 구성입니다.

슬레이브

이 브로커의 백업에 대한 공유 저장소 또는 복제 페일오버 구성입니다.

3. *클러스터의 나머지 브로커에 대해 이 절차를 반복합니다.*

추가 리소스

- *배치된 백업을 사용하는 브로커 클러스터의 예는 **HA 예제** 를 참조하십시오.*

14.3.6. 장애 조치(failover)를 위해 클라이언트 구성

브로커 클러스터에서 고가용성을 구성한 후 오류가 발생하도록 클라이언트를 구성합니다. 클라이언트 페일오버를 사용하면 브로커가 실패할 경우 다운타임을 최소화하여 클러스터의 다른 브로커에 다시 연결할 수 있습니다.

참고

*일시적인 네트워크 문제가 발생하는 경우 **AMQ Broker**는 동일한 브로커에 연결을 자동으로 다시 연결합니다. 클라이언트가 동일한 브로커에 다시 연결된다는 점을 제외하고 장애 조치와 유사합니다.*

두 가지 유형의 클라이언트 장애 조치를 구성할 수 있습니다.

자동 클라이언트 장애 조치

클라이언트는 처음 연결할 때 브로커 클러스터에 대한 정보를 받습니다. 연결된 브로커가 실패하면 클라이언트가 브로커의 백업에 자동으로 다시 연결되고 백업 브로커는 장애 조치 전에 각 연결에 존

재하는 모든 세션과 소비자를 다시 생성합니다.

애플리케이션 수준 클라이언트 장애 조치

자동 클라이언트 장애 조치 대신 실패 처리기에서 고유한 사용자 지정 다시 연결 논리를 사용하여 클라이언트 애플리케이션을 코딩할 수 있습니다.**As an alternative to automatic client failover, you can instead code your client applications with your own custom reconnection logic in a failure handler.**

프로세스

- **AMQ Core Protocol JMS**를 사용하여 자동 또는 애플리케이션 수준 페일오버로 클라이언트 애플리케이션을 구성합니다.

  자세한 내용은 **AMQ Core Protocol JMS Client** 사용을 참조하십시오.

14.4. 메시지 재배포 활성화

ON_DEMAND 또는 **OFF_WITH_REDISTRIBUTION** 으로 설정된 메시지 로드 밸런싱 을 사용하여 브로커 클러스터가 구성된 경우 메시지를 사용할 소비자가 없는 큐에서 메시지가 **"stuck"**되지 않도록 메시지 재배포 를 구성할 수 있습니다.

이 섹션에는 다음에 대한 정보가 포함되어 있습니다.

- 메시지 배포 이해
- 메시지 재배포 구성

14.4.1. 메시지 재배포 이해

브로커 클러스터는 로드 밸런싱을 사용하여 메시지 로드를 클러스터에 배포합니다. 클러스터 연결에서 로드 밸런싱을 구성할 때 다음 메시지 로드 밸런싱 설정을 사용하여 재배포를 활성화할 수 있습니다.

- **ON_DEMAND -** 로드 밸런싱을 활성화하고 재배포 허용
- **OFF_WITH_REDISTRIBUTION -** 부하 분산을 비활성화하지만 재배포를 허용합니다.

두 경우 모두 브로커는 일치하는 소비자가 있는 다른 브로커에만 메시지를 전달합니다. 이 동작을 사용하면 메시지를 사용할 소비자가 없는 대기열로 메시지가 이동되지 않습니다. 그러나 메시지가 브로커로 전달된 후 큐에 연결된 소비자가 대기열에 연결되면 해당 메시지가 큐에서 "스트리크"되고 사용되지 않습니다. 이 문제를 종종 별점이라고 합니다.

메시지 재배포는 소비자가 일치하는 클러스터의 브로커에 없는 대기열에서 메시지를 자동으로 재배포하는 것을 방지합니다.

OFF_WITH_REDISTRIBUTION 을 사용하면 브로커는 활성 로컬 소비자가 없는 경우에만 소비자와 일치하는 다른 브로커에 메시지를 전달하여 소비자를 사용할 수 없을 때 대안을 제공하는 동안 브로커의 우선 순위를 지정할 수 있습니다.

메시지 재배포는 사용 가능한 로컬 소비자의 선택기와 일치하지 않을 때 메시지가 재배포되는 필터 **(selector** 라고도 함) 사용을 지원합니다.

추가 리소스

- 클러스터 로드 밸런싱에 대한 자세한 내용은 **14.1.1**절. “브로커 클러스터의 메시지 부하를 분산하는 방법” 을 참조하십시오.

14.4.2. 메시지 재배포 구성

다음 절차에서는 로드 밸런싱을 사용하여 메시지 재배포를 구성하는 방법을 보여줍니다. 로드 밸런싱 없이 메시지 재배포를 사용하려면 **< message-load-balancing >**을 **OFF_WITH_REDISTRIBUTION** 으로 설정합니다.

프로세스

1. **&lt ;broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

2. **< cluster-connection >** 요소에서 **< message-load-balancing >**이 **ON_DEMAND** 로 설정되어 있는지 확인합니다.

```
<configuration>
  <core>
    ...
    <cluster-connections>
      <cluster-connection name="my-cluster">
        ...
```

```
            <message-load-balancing>ON_DEMAND</message-load-balancing>
            ...
        </cluster-connection>
    </cluster-connections>
  </core>
</configuration>
```

3.

&lt ;address-settings& gt; 요소 내에서 큐 또는 큐 집합에 대한 재배포 지연을 설정합니다.

이 예에서는 마지막 소비자가 종료한 후 **my.queue** 에 분산된 메시지가 **5000**밀리초 후에 재분배됩니다.

```
<configuration>
  <core>
    ...
    <address-settings>
      <address-setting match="my.queue">
        <redistribution-delay>5000</redistribution-delay>
      </address-setting>
    </address-settings>
    ...
  </core>
</configuration>
```

address-setting

match 속성을 메시지를 재배포할 큐의 이름으로 설정합니다. 브로커 와일드카드 구문을 사용하여 대기열 범위를 지정할 수 있습니다. 자세한 내용은 **4.2**절. “주소 세트에 주소 설정 적용”의 내용을 참조하십시오.

redistribution-delay

이 대기열의 최종 소비자가 클러스터의 다른 브로커에 다시 배포되기 전에 브로커가 기다려야 하는 시간(밀리초)입니다. 이 값을 **0** 으로 설정하면 메시지가 즉시 재배포됩니다. 그러나 일반적으로 재배포하기 전에 지연을 설정해야 합니다. 소비자는 닫는 것이 일반적이지만 다른 하나는 동일한 큐에 빠르게 생성될 수 있습니다.

4.

클러스터의 추가 브로커에 대해 이 절차를 반복합니다.

추가 리소스

- 메시지를 재배포하는 브로커 클러스터 구성의 예는 **queue-message-redistribution** 예제 를 참조하십시오.

14.5. 클러스터형 메시지 그룹화 구성

메시지 그룹화를 사용하면 클라이언트가 특정 유형의 메시지 그룹을 동일한 소비자가 직렬로 처리할 수 있습니다. 클러스터의 각 브로커에 그룹화 처리기를 추가하면 클라이언트가 클러스터의 모든 브로커에 그룹화된 메시지를 보낼 수 있고 동일한 소비자가 올바른 순서로 이러한 메시지를 계속 사용할 수 있습니다.

참고

그룹화 및 클러스터링 기술은 다음과 같이 요약할 수 있습니다.

- *메시지 그룹화는 메시지 사용에 순서를 적용합니다. 그룹에서는 다음 메시지를 진행하기 전에 각 메시지를 완전히 사용하고 승인해야 합니다. 이 방법론은 동시성이 옵션이 아닌 직렬 메시지 처리로 이어집니다.*

- *클러스터링은 메시지 처리량을 높이기 위해 브로커를 수평으로 확장하는 것을 목표로 합니다. 수평 확장은 메시지를 동시에 처리할 수 있는 추가 소비자를 추가하여 달성됩니다.*

이러한 기술은 서로 일치하지 않기 때문에 클러스터링 및 그룹화 사용을 피하십시오.

로컬 처리기와 원격 처리기의 두 가지 유형의 그룹화 처리기 가 있습니다. 브로커 클러스터를 사용하면 특정 그룹의 모든 메시지를 적절한 큐로 라우팅하여 의도한 소비자가 올바른 순서로 사용할 수 있도록 합니다.

사전 요구 사항

- *클러스터의 각 브로커에 하나 이상의 소비자가 있어야 합니다.*

메시지가 대기열의 소비자에 고정되면 그룹 ID가 동일한 모든 메시지가 해당 큐로 라우팅됩니다. 소비자가 제거되면 소비자가 없는 경우에도 대기열에서 메시지를 계속 수신합니다.

프로세스

1. *클러스터의 한 브로커에 로컬 처리기를 구성합니다.*

고가용성을 사용하는 경우 마스터 브로커여야 합니다.

a.

*브로커의 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

b.

***< core>** 요소 내에서 로컬 처리기를 추가합니다.*

로컬 처리기는 원격 처리기의 중재자 역할을 합니다. 라우팅 정보를 저장하고 다른 브로커와 통신합니다.

```
<configuration>
  <core>
    ...
    <grouping-handler name="my-grouping-handler">
      <type>LOCAL</type>
      <timeout>10000</timeout>
    </grouping-handler>
    ...
  </core>
</configuration>
```

grouping-handler

***name** 속성을 사용하여 그룹화 처리기의 고유한 이름을 지정합니다.*

type

*이를 **LOCAL** 로 설정합니다.*

timeout

*메시지를 라우팅할 위치에 대한 결정을 내릴 때까지 대기하는 시간(밀리초)입니다. 기본값은 **5000**밀리초입니다. 라우팅 결정이 이루어지기 전에 시간 초과에 도달하면 예외가 발생하여 엄격한 메시지 순서가 보장됩니다.*

*브로커가 그룹 **ID**가 있는 메시지를 수신하면 소비자가 연결된 큐의 경로를 제안합니다. 클러스터의 다른 브로커에서 경로를 그룹화하여 경로를 수락하면 경로가 설정됩니다. 클러스터의 모든 브로커는 이 그룹 **ID**가 있는 메시지를 해당 큐로 전달합니다. 브로커의 경로 제안이 거부되면 경로가 수락될 때까지 프로세스를 반복하여 대체 경로를 제안합니다.*

2.

고가용성을 사용하는 경우 로컬 처리기 구성을 마스터 브로커의 슬레이브 브로커에 복사합니다.

로컬 처리기 구성을 슬레이브 브로커에 복사하면 로컬 처리기의 단일 장애 지점을 방지할 수 있습니다.

3. 클러스터의 나머지 각 브로커에서 원격 처리기를 구성합니다.

a. 브로커의 **< broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

b. **< core>** 요소 내에서 원격 처리기를 추가합니다.

```
<configuration>
  <core>
    ...
    <grouping-handler name="my-grouping-handler">
      <type>REMOTE</type>
      <timeout>5000</timeout>
    </grouping-handler>
    ...
  </core>
</configuration>
```

grouping-handler

name 속성을 사용하여 그룹화 처리기의 고유한 이름을 지정합니다.

type

이를 **REMOTE** 로 설정합니다.

timeout

메시지를 라우팅할 위치에 대한 결정을 내릴 때까지 대기하는 시간(밀리초)입니다. 기본값은 **5000**밀리초입니다. 이 값을 로컬 처리기 값의 절반 이상으로 설정합니다.

추가 리소스

- 메시지 그룹화용으로 구성된 브로커 클러스터의 예는 **JMS** 클러스터형 그룹화 예제 를 참조하십시오.

14.6. 브로커 클러스터에 클라이언트 연결

Red Hat build of Apache Cryostat JMS 클라이언트를 사용하여 클러스터에 연결할 수 있습니다. **JMS**를 사용하면 메시징 클라이언트를 구성하여 브로커 목록을 동적으로 또는 정적으로 검색할 수 있습니다. 또한 클러스터 전체에서 연결에서 생성된 클라이언트 세션을 배포하도록 클라이언트 측 로드 밸런싱을 구성할 수도 있습니다.

프로세스

프로세스

- **AMQ Core Protocol JMS**를 사용하여 브로커 클러스터에 연결하도록 클라이언트 애플리케이션을 구성합니다.

  자세한 내용은 **AMQ Core Protocol JMS Client** 사용을 참조하십시오.

14.7. 클라이언트 연결 파티셔닝

클라이언트 연결을 분할하려면 클라이언트가 연결을 시작할 때마다 개별 클라이언트의 라우팅 연결이 동일한 브로커로 연결됩니다.

클라이언트 연결을 분할하는 두 가지 사용 사례는 다음과 같습니다.

- **subscriber**가 항상 도달할 수 있는 구독자 큐가 있는 브로커에 연결할 수 있도록 서브스크립션의 클라이언트를 분할합니다.
- 클라이언트를 데이터 전송이라고도 하는 위치로 끌어들이는 방식으로 데이터를 이동할 필요성을 최소화합니다.

Cryostat 서브스크립션

Cryostat 서브스크립션은 브로커의 큐로 표시되며, 지속성 구독자가 브로커에 처음 연결할 때 생성됩니다. 이 대기열은 브로커에 남아 있으며 클라이언트가 구독을 취소할 때까지 메시지를 수신합니다. 따라서 클라이언트가 구독자 큐에 있는 메시지를 사용하기 위해 동일한 브로커에 반복적으로 연결하려고 합니다.

ScanSetting 서브스크립션 큐를 위해 클라이언트를 파티셔닝하려면 클라이언트 연결에서 클라이언트 **ID**를 필터링할 수 있습니다.

데이터 가시성

데이터 심각도를 고려하지 않고 사용자 환경의 브로커 수를 확장하면 브로커 간에 메시지를 이동해야 하므로 성능상의 이점이 손실됩니다. 날짜 분대성을 지원하려면 클라이언트 소비자가 소비해야 하는 메시지가 생성되는 브로커에 연결되도록 클라이언트 연결을 분할해야 합니다.

클라이언트 연결을 파티셔닝하여 데이터 저하를 지원하려면 클라이언트 연결의 다음 특성을 필터링할 수 있습니다.

- 연결 사용자에게 할당된 역할(**ROLE_NAME**)
- 사용자 사용자 이름(**USER_NAME**)
- 클라이언트의 호스트 이름(**SNI_HOST**)
- 클라이언트의 **IP** 주소(**SOURCE_IP**)

14.7.1. ScanSetting 서브스크립션을 지원하기 위해 클라이언트 연결 파티셔닝

ScanSetting 서브스크립션을 위해 클라이언트를 파티셔닝하려면 일관된 해시 알고리즘 또는 정규식을 사용하여 들어오는 연결에서 클라이언트 **ID**를 필터링할 수 있습니다.

사전 요구 사항

클라이언트는 예를 들어 로드 밸런서를 사용하거나 연결 **URL**에 모든 브로커 인스턴스를 구성하여 클러스터의 모든 브로커에 연결할 수 있도록 구성됩니다. 클라이언트 세부 정보가 해당 브로커의 파티션 구성과 일치하지 않기 때문에 브로커가 연결을 거부하는 경우 클라이언트는 클러스터의 다른 브로커에 연결하여 연결을 수락하는 브로커를 찾을 수 있어야합니다.

14.7.1.1. 일관된 해시 알고리즘을 사용하여 클라이언트 ID 필터링

일관된 해시 알고리즘을 사용하여 각 클라이언트 연결에서 클라이언트 **ID**를 해시하도록 클러스터의 각 브로커를 구성할 수 있습니다. 브로커가 클라이언트 **ID**를 해시한 후 해시된 값에 대한 **modulo** 작업을 수행하여 클라이언트 연결에 대한 대상 브로커를 식별하는 정수 값을 반환합니다. 브로커는 반환 된 정수 값을 브로커에 구성된 고유 값과 비교합니다. 일치하는 항목이 있는 경우 브로커는 연결을 수락합니다. 값이 일치하지 않으면 브로커가 연결을 거부합니다. 이 프로세스는 일치 항목이 발견되고 브로커가 연결을 수락할 때까지 클러스터의 각 브로커에서 반복됩니다.

프로세스

1. 첫 번째 브로커의 **<broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.
2. **connection-routers** 요소를 만들고 일관된 해시 알고리즘을 사용하여 클라이언트 **ID**를 필

*터링하는 **connection-route** 를 생성합니다. 예를 들면 다음과 같습니다.*

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name=”consistent-hash-routing”>
        <key>CLIENT_ID</target-key>
        <local-target-filter>NULL|0</local-target-filter>
        <policy name="CONSISTENT_HASH_MODULO">
          <property key="modulo" value="<number_of_brokers_in_cluster>">
          </property>
        </policy>
      </connection-route>
    </connection-routers>
    ...
  </core>
</configuration>
```

connection-route

***connection-route** 이름 의 경우 이 연결 라우팅 구성에 대해 식별 문자열을 지정합니다. 일관된 해시 필터를 적용하려면 이 이름을 각 브로커 수락자에 추가해야 합니다.*

key

*필터를 적용할 키 유형입니다. 클라이언트 **ID**를 필터링하려면 키 필드에 **CLIENT_ID** 를 지정합니다.*

local-target-filter

*브로커가 **modulo** 작업에서 반환된 정수 값과 비교하여 일치하는 항목이 있는지 확인하고 브로커가 연결을 수락할 수 있는지 확인합니다. 예제의 **NULL|0** 값은 클라이언트 **ID(NULL)**가 없는 연결에 대해 일치하고 **modulo** 작업에서 반환된 숫자가 **0** 인 연결을 제공합니다.*

policy

*해시된 클라이언트 **ID**에서 **modulo** 작업을 수행하여 대상 브로커를 식별하는 **modulo** 속성 키를 허용합니다. **modulo** 속성 키의 값은 클러스터의 브로커 수와 같아야 합니다.*

중요

*정책 이름은 **CONSISTENT_HASH_MODULO** 여야 합니다.*

3. *두 번째 브로커의 **<broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.*

4. *connection-routers 요소를 만들고 일관된 해시 알고리즘을 사용하여 클라이언트 ID를 필터링하는 연결 경로를 만듭니다.*

   다음 예에서 NULL|1 의 local-target-filter 값은 NULL(클라이언트 ID)이 없는 연결과 modulo 작업에서 반환된 값이 1 인 연결에 대해 일치를 제공합니다.

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name=”consistent-hash-routing”>
        <key>CLIENT_ID</target-key>
        <local-target-filter>NULL|1</local-target-filter>
        <policy name="CONSISTENT_HASH_MODULO">
          <property key="modulo" value="<number_of_brokers_in_cluster>">
          </property>
        </policy>
      </connection-route>
    </connection-routers>
    ...
  </core>
</configuration>
```

5. *클러스터의 각 추가 브로커에 대해 일관된 해시 필터를 생성하려면 이 절차를 반복합니다.*

14.7.1.2. 정규식을 사용하여 클라이언트 ID 필터링

클라이언트 연결에서 클라이언트 ID 부분에 정규식 필터를 적용하도록 브로커를 구성하여 클라이언트 연결을 분할할 수 있습니다. 브로커는 정규식 필터의 결과가 브로커에 대해 구성된 로컬 대상 필터와 일치하는 경우에만 연결을 허용합니다. 일치 항목이 없으면 브로커가 연결을 거부합니다. 이 프로세스는 일치 항목이 발견되고 브로커가 연결을 수락할 때까지 클러스터의 각 브로커에서 반복됩니다.

사전 요구 사항

- *정규식으로 필터링할 수 있는 각 클라이언트 ID의 일반 문자열입니다.*

프로세스

1. *첫 번째 브로커의 <broker_instance_dir> /etc/broker.xml 구성 파일을 엽니다.*

2. *connection-routers 요소를 생성하고 클라이언트 ID의 일부를 필터링하는 connection-route 를 생성합니다. 예를 들면 다음과 같습니다.*

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name="regex-routing">
        <key>CLIENT_ID</target-key>
        <key-filter>^.{3}</key-filter>
        <local-target-filter>NULL|CL1</local-target-filter>
      </connection-route>
  </connection-routers>
    ...
  </core>
</configuration>
```

connection-route

connection-route 이름 의 경우 이 라우팅 구성에 대한 식별 문자열을 지정합니다. 정규식 필터를 적용하려면 이 이름을 각 브로커 수락자에 추가해야 합니다.

key

필터를 적용할 키입니다. 클라이언트 **ID**를 필터링하려면 키 필드에 **CLIENT_ID** 를 지정합니다.

key-filter

키 값을 추출하기 위해 정규식이 적용되는 클라이언트 **ID** 문자열의 일부입니다. 위의 첫 번째 브로커의 예에서 브로커는 클라이언트 **ID**의 처음 **3**자인 키 값을 추출합니다. 예를 들어 클라이언트 **ID** 문자열이 **CL100.consumer** 인 경우 브로커는 **CL1** 키 값을 추출합니다. 브로커가 키 값을 추출한 후 **local-target-filter** 의 값과 비교합니다.

들어오는 연결에 클라이언트 **ID**가 없거나 브로커가 키 필터링 에 지정된 정규식을 사용하여 키 값을 추출할 수 없는 경우 키 값이 **NULL**로 설정됩니다.

local-target-filter

브로커가 키 값과 비교하여 일치하는 항목이 있는지 확인하고 브로커가 해당 연결을 수락할 수 있는지 확인합니다. 위의 첫 번째 브로커에 대한 예에 표시된 것처럼 **NULL|CL1** 값은 클라이언트 **ID(NULL)**가 없거나 클라이언트 **ID**에 **CL1** 의 **3**자 접두사를 갖는 연결과 일치합니다.

3. 두 번째 브로커의 **<broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

4. **connection-routers** 요소를 만들고 클라이언트 **ID**의 일부를 기반으로 연결을 필터링하는 연결 경로를 만듭니다.

다음 필터 예에서 브로커는 정규식을 사용하여 클라이언트 ID의 처음 3자인 키 값을 추출합니다. 브로커는 NULL 및 CL2 의 값을 키 값과 비교하여 일치하는 항목이 있는지 확인하고 브로커가 연결을 수락할 수 있는지 확인합니다.

```
<configuration>
   <core>
      ...
      <connection-routers>
         <connection-route name="regex-routing">
            <key>CLIENT_ID</target-key>
            <key-filter>^.{3}</key-filter>
            <local-target-filter>NULL|CL2</local-target-filter>
         </connection-route>
      </connection-routers>
      ...
   </core>
</configuration>
```

5. 이 절차를 반복하고 클러스터의 추가 브로커마다 적절한 연결 라우팅 필터를 생성합니다.

14.7.2. 데이터 가시성을 지원하기 위해 클라이언트 연결 파티셔닝

클라이언트 소비자가 소비해야 하는 메시지가 생성되는 브로커에 연결되도록 클라이언트 연결을 분할할 수 있습니다. 예를 들어 생산자 및 소비자 애플리케이션에서 사용하는 주소 세트가 있는 경우 특정 브로커에서 주소를 구성할 수 있습니다. 그런 다음 해당 주소를 사용하는 생산자와 소비자 모두에 대한 클라이언트 연결을 분할하여 해당 브로커에만 연결할 수 있습니다.

연결된 사용자에게 할당된 역할, 사용자의 사용자 이름 또는 클라이언트의 호스트 이름 또는 IP 주소와 같은 특성을 기반으로 클라이언트 연결을 분할할 수 있습니다. 이 섹션에서는 클라이언트 사용자에게 할당된 사용자 역할을 필터링하여 클라이언트 연결을 분할하는 방법의 예를 보여줍니다. 클라이언트가 브로커에 연결하기 위해 인증해야 하는 경우 역할 기준과 일치하는 사용자만 브로커에 연결할 수 있도록 클라이언트 사용자에게 역할을 할당하고 연결을 필터링할 수 있습니다.

사전 요구 사항

- 클라이언트는 예를 들어 로드 밸런서를 사용하거나 연결 URL에 모든 브로커 인스턴스를 구성하여 클러스터의 모든 브로커에 연결할 수 있도록 구성됩니다. 클라이언트가 해당 브로커에 대해 구성된 파티션 필터 기준과 일치하지 않기 때문에 연결을 거부하는 경우 클라이언트는 클러스터의 다른 브로커에 연결하여 연결을 수락하는 브로커를 찾을 수 있어야합니다.

프로세스

1. 첫 번째 브로커의 **<broker_instance_dir> /etc/artemis-roles.properties** 파일을 엽니다. **broker1users** 역할을 추가하고 사용자를 역할에 추가합니다.

2. 첫 번째 브로커의 **<broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

3. **connection-routers** 요소를 생성하고 사용자에게 할당된 역할에 따라 연결을 필터링하는 **connection-route** 를 생성합니다. 예를 들면 다음과 같습니다.

```
<configuration>
  <core>
    ...
    <connection-routers>
      <connection-route name="role-based-routing">
        <key>ROLE_NAME</target-key>
        <key-filter>broker1users</key-filter>
        <local-target-filter>broker1users</local-target-filter>
      </connection-route>
    </connection-routers>
    ...
  </core>
</configuration>
```

connection-route

connection-route 이름 의 경우 이 라우팅 구성에 대한 식별 문자열을 지정합니다. 역할 필터를 적용하려면 이 이름을 각 브로커 수락자에 추가해야 합니다.

key

필터를 적용할 키입니다. 역할 기반 필터링을 구성하려면 키 필드에 **ROLE_NAME** 을 지정합니다.

key-filter

브로커가 사용자 역할을 필터링하고 키 값을 추출하는 데 사용하는 문자열 또는 정규식입니다. 브로커가 일치하는 역할을 발견하면 키 값을 해당 역할로 설정합니다. 일치하는 역할을 찾지 못하면 브로커는 키 값을 **NULL**로 설정합니다. 위의 예에서 브로커는 **broker1users** 필터를 클라이언트 사용자의 역할에 적용합니다. 브로커가 키 값을 추출한 후 **local-target-filter** 의 값과 비교합니다.

local-target-filter

브로커가 키 값과 비교하여 일치하는 항목이 있는지 확인하고 브로커가 해당 연결을 수락할 수 있는지 확인합니다. 이 예에서 브로커는 **broker1users** 값을 키 값과 비교합니다. 일치하는 항목이 있습니다. 즉, 사용자에게 **broker1users** 역할이 있고 브로커가 연결을 수락합니다.

4. 이 절차를 반복하고 필터에서 적절한 역할을 지정하여 클러스터의 다른 브로커의 클라이언트를 분할합니다.

14.7.3. 어셉터에 연결 경로 추가

브로커에 연결 경로를 구성한 후 클라이언트 연결을 파티셔닝하기 위해 브로커의 승인자 중 하나에 경로를 추가해야 합니다. 수락자에 연결 경로를 추가한 후 브로커는 연결 경로에 구성된 필터를 수락자가 수신한 연결에 적용합니다.

프로세스

1. 첫 번째 브로커의 **<broker_instance_dir> /etc/broker.xml** 구성 파일을 엽니다.

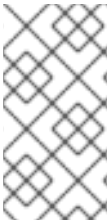
2. 파티셔닝을 활성화할 각 수락자에 대해 라우터 키를 추가하고 **connection-route** 이름을 지정합니다. 다음 예에서 **consistent-hash -routing** 의 연결 경로 이름이 **Artemis acceptor**에 추가됩니다.

```
<configuration>
  <core>
    ...
    <acceptors>
    ...
    <!-- Acceptor for every supported protocol -->
    <acceptor name="artemis">tcp://0.0.0.0:61616?
tcpSendBufferSize=1048576;tcpReceiveBufferSize=1048576;protocols=CORE,AMQP,S
TOMP,HORNETQ,MQTT,OPENWIRE;useEpoll=true;amqpCredits=1000;amqpLowCredit
s=300;router="consistent-hash-routing" </acceptor>
    </acceptors>
    ...
  </core>
</configuration>
```

3. 이 절차를 반복하여 클러스터의 각 브로커에 적절한 연결 경로 필터를 지정합니다.

15장. CEPH를 사용하여 다중 사이트 내결함성 메시징 시스템 구성

대규모 엔터프라이즈 메시징 시스템에는 일반적으로 지리적으로 분산된 데이터 센터에 있는 별도의 브로커 클러스터가 있습니다. 데이터 센터 중단 시 시스템 관리자는 기존 메시징 데이터를 보존하고 클라이언트 애플리케이션이 메시지를 계속 생성하고 사용할 수 있도록 해야 할 수 있습니다. 특정 브로커 토폴로지 및 소프트웨어 정의 스토리지 플랫폼인 **Red Hat Ceph Storage**를 사용하여 데이터 센터 중단 중에 메시징 시스템의 연속성을 보장할 수 있습니다. 이러한 유형의 솔루션을 다중 사이트 내결함성 아키텍처 라고 합니다.

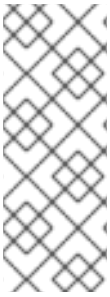


참고

AMQP 프로토콜 지원만 필요한 경우 16장. 브로커 연결을 사용하여 다중 사이트 내결함성 메시징 시스템 구성 을/를 고려하십시오.

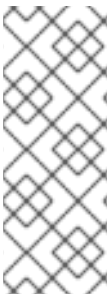
다음 섹션에서는 **Red Hat Ceph Storage**를 사용하여 데이터 센터 중단으로부터 메시징 시스템을 보호하는 방법을 설명합니다.

- **Red Hat Ceph Storage** 클러스터 작동 방식
- **Red Hat Ceph Storage** 클러스터 설치 및 구성
- 데이터 센터 중단 시 라이브 브로커에서 인수할 백업 브로커 추가
- **Ceph** 클라이언트 역할을 사용하여 브로커 서버 구성
- 공유 저장소 **HA**(고가용성) 정책을 사용하도록 각 브로커 구성, 각 브로커가 메시징 데이터를 저장하는 위치를 지정
- 데이터 센터 중단 시 새 브로커에 연결하도록 클라이언트 애플리케이션 구성
- 중단 후 데이터 센터 다시 시작



참고

다중 사이트 내결함성은 데이터 센터 내에서 **HA**(고가용성) 브로커 중복을 대체하지 않습니다. 라이브 백업 그룹을 기반으로 하는 브로커 중복성은 단일 클러스터 내의 단일 브로커 오류에 대한 자동 보호 기능을 제공합니다. 반면 다중 사이트 내결함성은 대규모 데이터 센터 중단을 방지합니다.



참고

메시징 시스템의 연속성을 보장하기 위해 **Red Hat Ceph Storage**를 사용하려면 공유 저장소 **HA**(고가용성) 정책을 사용하도록 브로커를 구성해야 합니다. 복제 **HA** 정책을 사용하도록 브로커를 구성할 수 없습니다. 이러한 정책에 대한 자세한 내용은 고가용성 구현을 참조하십시오.

15.1. RED HAT CEPH STORAGE 클러스터 작동 방식

Red Hat Ceph Storage는 클러스터형 오브젝트 스토리지 시스템입니다. **Red Hat Ceph Storage**는 오브젝트 및 정책 기반 복제의 데이터 분할을 사용하여 데이터 무결성 및 시스템 가용성을 보장합니다.

Red Hat Ceph Storage는 **CRUSH**(**Controlled Replication Under Scalable Hashing**)라는 알고리즘을 사용하여 자동으로 데이터 스토리지 위치를 저장하고 검색하는 방법을 결정합니다. **CRUSH** 맵 이라는 **Ceph** 항목을 구성하여 클러스터 토폴로지를 자세히 설명하고 스토리지 클러스터 간에 데이터를 복제하는 방법을 지정합니다.

CRUSH 맵에는 **OSD**(오브젝트 스토리지 장치) 목록, 장치를 장애 도메인 계층 구조로 집계하는 **'buckets'** 목록, **CRUSH**에 **Ceph** 클러스터의 풀에서 데이터를 복제하는 방법을 알려주는 규칙이 포함되어 있습니다.

설치의 기본 물리적 조직을 반영하여 **CRUSH** 맵은 모델링할 수 있으므로 물리적 근접성, 공유 전원 소스 및 공유 네트워크와 같은 관련 장치 오류의 잠재적 소스를 모델링할 수 있습니다. 이 정보를 클러스터 맵에 인코딩하여 **CRUSH**는 스토리지 클러스터에서 데이터의 의사 무작위 배포를 유지 관리하면서 다양한 장애 도메인(예: 데이터 센터) 간에 오브젝트 복제본을 분리할 수 있습니다. 이렇게 하면 데이터 손실을 방지하고 클러스터가 성능 저하된 상태로 작동할 수 있습니다.

Red Hat Ceph Storage 클러스터에는 다양한 노드(실제 또는 가상)가 필요합니다. 클러스터에는 다음 유형의 노드가 포함되어야 합니다.

노드 모니터링

각 모니터(**MON**) 노드는 클러스터 맵의 마스터 복사본을 유지 관리하는 모니터 데몬(**ceph-mon**)을 실행합니다. 클러스터 맵에는 클러스터 토폴로지가 포함됩니다. **Ceph** 클러스터에 연결하는 클라이언트는

Monitor에서 클러스터 맵의 현재 사본을 검색하여 클라이언트가 클러스터에서 데이터를 읽고 쓸 수 있습니다.

중요

Red Hat Ceph Storage 클러스터는 하나의 모니터 노드로 실행할 수 있지만 프로덕션 클러스터에서 고가용성을 보장하기 위해 Red Hat은 모니터 노드가 3개 이상인 배포만 지원합니다. 최소 3개의 모니터 노드는 하나의 Monitor가 실패하거나 사용할 수 없는 경우 클러스터의 나머지 모니터 노드에서 새 리더를 선택할 수 있도록 쿼럼이 존재합니다.

관리자 노드

각 Manager(MGR) 노드는 Ceph Manager 데몬(ceph-mgr)을 실행합니다. 이 데몬은 런타임 지표 및 스토리지 사용률, 현재 성능 지표, 시스템 로드를 포함하여 Ceph 클러스터의 현재 상태를 추적합니다. 일반적으로 Manager 노드는 모니터 노드와 함께 배치됩니다(즉, 동일한 호스트 시스템에서).

Object Storage 장치 노드

각 OSD(오브젝트 스토리지 장치) 노드는 Ceph OSD 데몬(ceph-osd)을 실행하여 노드에 연결된 논리 디스크와 상호 작용합니다. Ceph는 OSD 노드에 데이터를 저장합니다. Ceph는 매우 적은 수의 OSD 노드로 실행할 수 있지만(기본값은 3개) 프로덕션 클러스터는 모드 확장 시 성능이 향상됩니다(예: 스토리지 클러스터에 50개의 OSD 사용). 스토리지 클러스터에 여러 OSD가 있으면 시스템 관리자가 CRUSH 맵 내에서 격리된 장애 도메인을 정의할 수 있습니다.

메타데이터 서버 노드

각 메타데이터 서버(MDS) 노드는 MDS 데몬(ceph-mds)을 실행하여 Ceph 파일 시스템(CephFS)에 저장된 파일과 관련된 메타데이터를 관리합니다. MDS 데몬은 공유 클러스터에 대한 액세스도 조정합니다.

추가 리소스

Red Hat Ceph Storage에 대한 자세한 내용은 Red Hat Ceph Storage를 참조하십시오.

15.2. RED HAT CEPH STORAGE 설치

AMQ Broker 다중 사이트 내결함성 아키텍처에서는 Red Hat Ceph Storage 3을 사용합니다. Red Hat Ceph Storage 클러스터는 데이터 센터에 데이터를 복제하면 별도의 데이터 센터의 브로커가 사용할 수 있는 공유 저장소를 효과적으로 생성합니다. 공유 저장소 HA(고가용성) 정책을 사용하고 메시징 데이터를 Red Hat Ceph Storage 클러스터에 저장하도록 브로커를 구성합니다.

프로덕션 용도로 사용되는 **Red Hat Ceph Storage** 클러스터에는 다음이 포함되어야 합니다.

- **3**개의 모니터**(MON)** 노드
- **3**개의 **MGR (MGR)** 노드
- 여러 **OSD** 데몬이 포함된 **Object Storage Device(OSD)** 노드 **3**개
- 메타데이터 서버**(MDS)** 노드 세 개

중요

동일하거나 분리된 물리적 또는 가상 머신에서 **OSD, MON, MGR** 및 **MDS** 노드를 실행할 수 있습니다. 그러나 **Red Hat Ceph Storage** 클러스터 내에서 내결함성을 보장하기 위해서는 각 유형의 노드를 별도의 데이터 센터에 배포하는 것이 좋습니다. 특히 단일 데이터 센터 중단 시 스토리지 클러스터에 최소 두 개의 사용 가능한 **MON** 노드가 있는지 확인해야 합니다. 따라서 클러스터에 **3**개의 **MON** 노드가 있는 경우 각 노드는 별도의 데이터 센터의 별도의 호스트 시스템에서 실행해야 합니다. 이 데이터 센터가 실패하면 스토리지 클러스터에서 남아 있는 **MON** 노드만 있으므로 단일 데이터 센터에서 두 개의 **MON** 노드를 실행하지 마십시오. 이 경우 스토리지 클러스터가 더 이상 작동하지 않을 수 있습니다.

이 섹션의 절차에서는 **MON, MGR, OSD** 및 **MDS** 노드가 포함된 **Red Hat Ceph Storage 3** 클러스터를 설치하는 방법을 보여줍니다.

사전 요구 사항

- **Red Hat Ceph Storage** 설치 준비에 대한 자세한 내용은 다음을 참조하십시오.
  - 사전 요구 사항
  - **Red Hat Ceph Storage** 설치를 위한 요구 사항 체크리스트

프로세스

- **MON, MGR, OSD** 및 **MDS** 노드가 포함된 **Red Hat Ceph 3** 스토리지 클러스터를 설치하는

방법을 보여주는 절차는 다음을 참조하십시오.

- Red Hat Ceph Storage 클러스터 설치
- 메타데이터 서버 설치

15.3. RED HAT CEPH STORAGE 클러스터 구성

다음 예제 절차에서는 내결함성을 위해 Red Hat Ceph 스토리지 클러스터를 구성하는 방법을 보여줍니다. CRUSH 버킷을 생성하여 OSD(오브젝트 스토리지 장치) 노드를 실제 물리적 설치를 반영하는 데이터 센터로 집계합니다. 또한 CRUSH에 스토리지 풀에서 데이터를 복제하는 방법을 알려주는 규칙을 생성합니다. 이 단계에서는 Ceph 설치로 생성된 기본 CRUSH 맵을 업데이트합니다.

사전 요구 사항

- Red Hat Ceph Storage 클러스터가 이미 설치되어 있습니다. 자세한 내용은 Red Hat Ceph Storage 설치를 참조하십시오.
- Red Hat Ceph Storage에서 PG(배치 그룹)를 사용하여 풀에서 많은 수의 데이터 오브젝트를 구성하는 방법과 풀에서 사용할 PG 수를 계산하는 방법을 이해해야 합니다. 자세한 내용은 PG(배치 그룹)를 참조하십시오.
- 풀에서 오브젝트 복제본 수를 설정하는 방법을 이해해야 합니다. 자세한 내용은 오브젝트 복제본 수를 설정합니다.

프로세스

1. OSD 노드를 구성하기 위해 CRUSH 버킷을 생성합니다. 버킷은 데이터 센터와 같은 물리적 위치를 기반으로 하는 OSD 목록입니다. Ceph에서 이러한 물리적 위치를 장애 도메인 이라고 합니다.

```
ceph osd crush add-bucket dc1 datacenter
ceph osd crush add-bucket dc2 datacenter
```

2. OSD 노드의 호스트 시스템을 생성한 데이터 센터 CRUSH 버킷으로 이동합니다. 호스트 이름 host1-host4 를 호스트 시스템의 이름으로 교체합니다.

```
ceph osd crush move host1 datacenter=dc1
```

```
ceph osd crush move host2 datacenter=dc1
ceph osd crush move host3 datacenter=dc2
ceph osd crush move host4 datacenter=dc2
```

3. *생성한 **CRUSH** 버킷이 기본 **CRUSH** 트리의 일부인지 확인합니다.*

```
ceph osd crush move dc1 root=default
ceph osd crush move dc2 root=default
```

4. *데이터 센터 전체에서 스토리지 오브젝트 복제본을 매핑하는 규칙을 생성합니다. 이렇게 하면 데이터 손실을 방지하고 단일 데이터 센터 중단 시 클러스터가 계속 실행될 수 있습니다.*

*규칙을 생성하는 명령은 다음 구문을 사용합니다. **ceph osd crush** 규칙 **create-replicated <rule-name> <root> <failure-domain> <class>**. 예를 들면 다음과 같습니다.*

```
ceph osd crush rule create-replicated multi-dc default datacenter hdd
```

참고

*이전 명령에서 스토리지 클러스터가 **SSD(Solid-State Drive)**를 사용하는 경우 **hdd** (하드 디스크 드라이브) 대신 **ssd** 를 지정합니다.*

5. *생성한 규칙을 사용하도록 **Ceph** 데이터 및 메타데이터 풀을 구성합니다. 처음에 이로 인해 **CRUSH** 알고리즘에 의해 결정된 스토리지 대상으로 데이터가 다시 입력될 수 있습니다.*

```
ceph osd pool set cephfs_data crush_rule multi-dc
ceph osd pool set cephfs_metadata crush_rule multi-dc
```

6. *메타데이터 및 데이터 풀에 대한 **PG**(배치 그룹) 및 배치 그룹(**PGP**) 수를 지정합니다. **PGP** 값은 **PG** 값과 같아야 합니다.*

```
ceph osd pool set cephfs_metadata pg_num 128
ceph osd pool set cephfs_metadata pgp_num 128

ceph osd pool set cephfs_data pg_num 128
ceph osd pool set cephfs_data pgp_num 128
```

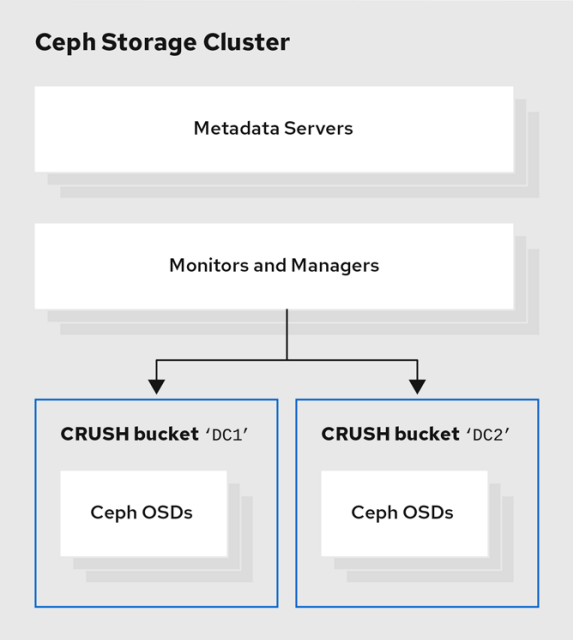
7. *데이터 및 메타데이터 풀에서 사용할 복제본 수를 지정합니다.*

```
ceph osd pool set cephfs_data min_size 1
```

```
ceph osd pool set cephfs_metadata min_size 1

ceph osd pool set cephfs_data size 2
ceph osd pool set cephfs_metadata size 2
```

*다음 그림은 이전 예제 절차에 의해 생성된 **Red Hat Ceph Storage** 클러스터를 보여줍니다. 스토리지 클러스터에는 데이터 센터에 해당하는 **CRUSH** 버킷으로 구성된 **OSD**가 있습니다.*

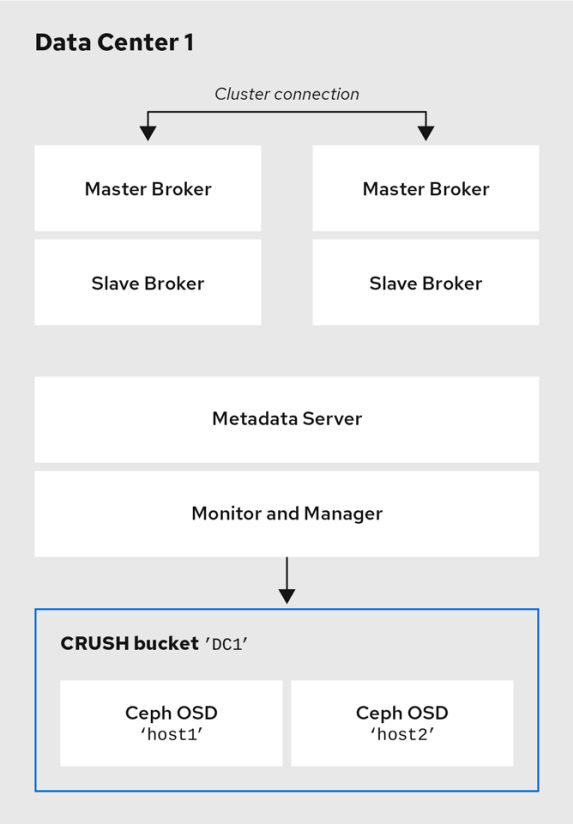
*그림 **15.1. Red Hat Ceph Storage Curster** 예*



다음 그림은 브로커 서버를 포함하여 첫 번째 데이터 센터의 가능한 레이아웃을 보여줍니다. 특히 데이터 센터 호스트:

- *두 개의 라이브 백업 브로커 쌍을 위한 서버*
- *이전 절차의 첫 번째 데이터 센터에 할당한 **OSD** 노드*
- *단일 메타데이터 서버, 모니터 및 관리자 노드. 모니터 및 관리자 노드는 일반적으로 동일한 시스템에 공동 배치됩니다.*

그림 15.2. Ceph Storage 클러스터를 사용하는 단일 데이터 센터 및 두 개의 라이브 백업 브로커 쌍



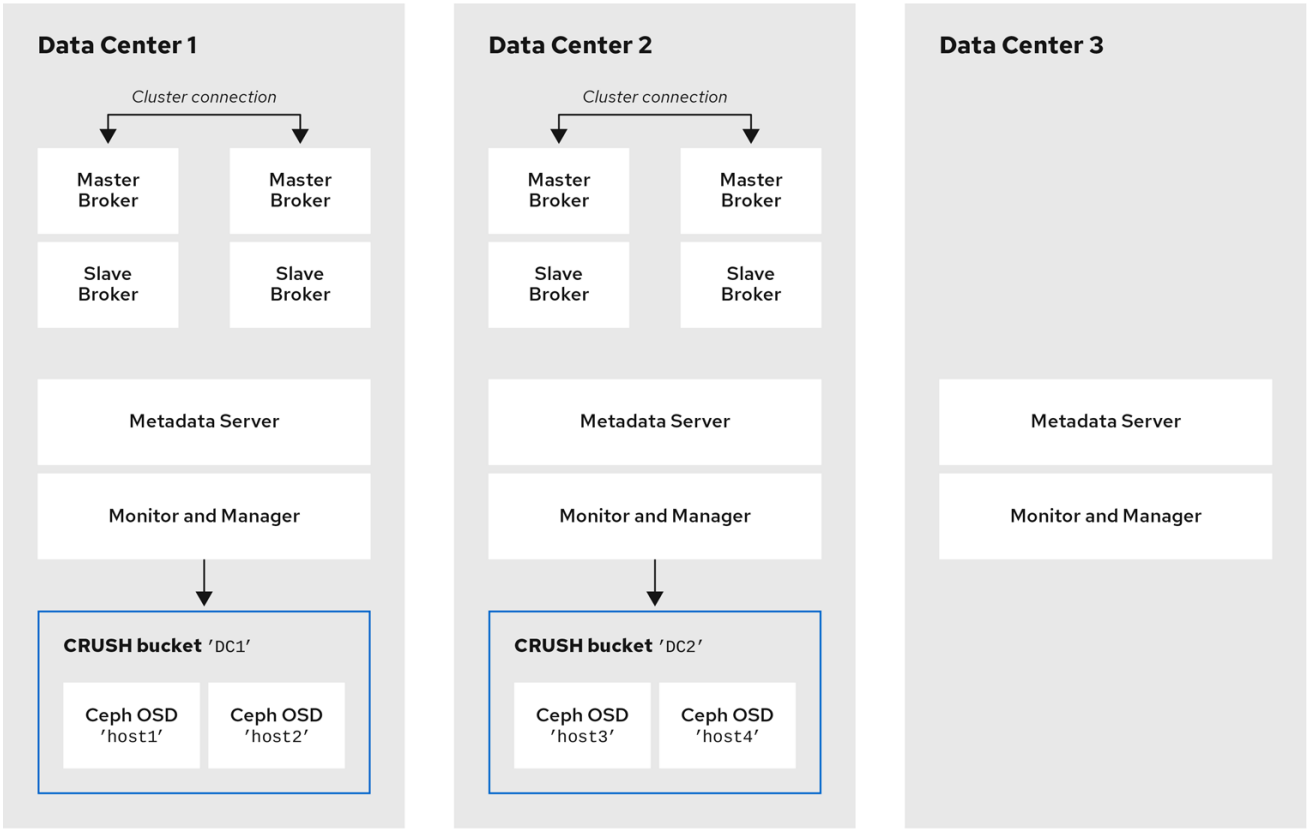
Ceph_41_0919

중요

*동일하거나 분리된 물리적 또는 가상 머신에서 **OSD, MON, MGR** 및 **MDS** 노드를 실행할 수 있습니다. 그러나 **Red Hat Ceph Storage** 클러스터 내에서 내결함성을 보장하기 위해서는 각 유형의 노드를 별도의 데이터 센터에 배포하는 것이 좋습니다. 특히 단일 데이터 센터 중단 시 스토리지 클러스터에 최소 두 개의 사용 가능한 **MON** 노드가 있는지 확인해야 합니다. 따라서 클러스터에 **3**개의 **MON** 노드가 있는 경우 각 노드는 별도의 데이터 센터의 별도의 호스트 시스템에서 실행해야 합니다.*

*다음 그림은 전체 예제 토폴로지를 보여줍니다. 스토리지 클러스터의 내결함성을 보장하기 위해 **MON, MGR** 및 **MDS** 노드가 **3**개의 개별 데이터 센터에 배포됩니다.*

그림 15.3. Ceph 스토리지 클러스터가 있는 여러 데이터 센터와 두 개의 라이브 백업 브로커 쌍

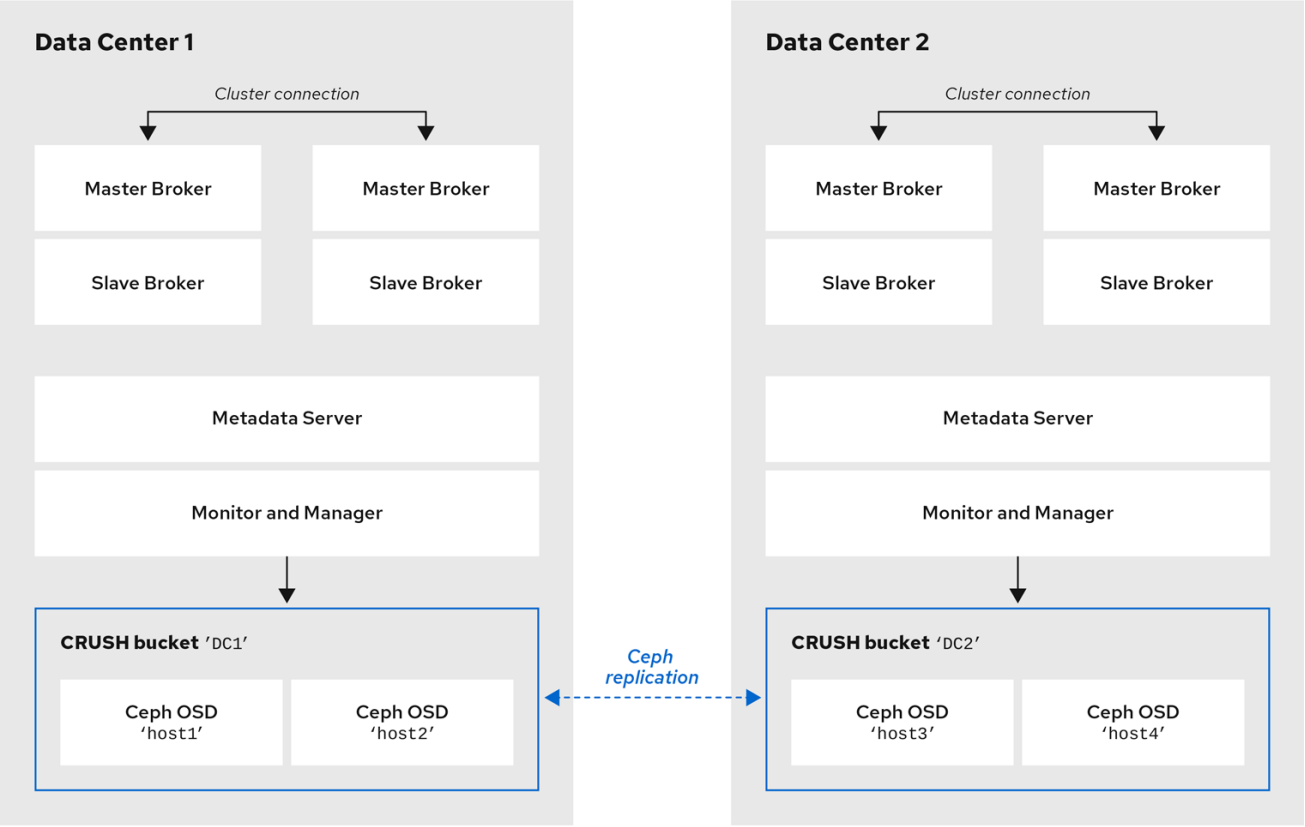


참고

*브로커 서버와 동일한 데이터 센터에 있는 특정 **OSD** 노드의 호스트 시스템을 찾는 것이 해당 특정 **OSD** 노드에 메시징 데이터를 저장하는 것을 의미하지는 않습니다. **Ceph** 파일 시스템의 지정된 디렉터리에 메시징 데이터를 저장하도록 브로커를 구성합니다. 그런 다음 클러스터의 메타데이터 서버 노드는 데이터 센터에서 사용 가능한 모든 **OSD**에 저장된 데이터를 배포하고 데이터 센터 간에 이 데이터의 복제를 처리하는 방법을 결정합니다. 다음 섹션에서는 **Ceph** 파일 시스템에 메시징 데이터를 저장하도록 브로커를 구성하는 방법을 보여줍니다.*

아래 그림은 브로커 서버가 있는 두 데이터 센터 간 데이터 복제를 보여줍니다.

그림 15.4. 브로커 서버가 있는 두 데이터 센터 간 Ceph 교체



추가 리소스

다음에 대한 자세한 내용은 다음을 수행합니다.

- Red Hat Ceph Storage 클러스터의 CRUSH 관리 에서는 CRUSH 관리를 참조하십시오.
- 스토리지 풀에 설정할 수 있는 전체 속성 세트는 풀 값을 참조하십시오.

15.4. 브로커 서버에 CEPH 파일 시스템 마운트

메시징 시스템에서 메시징 데이터를 저장하도록 Red Hat Ceph Storage 클러스터에 브로커를 구성하려면 먼저 Ceph 파일 시스템(CephFS)을 마운트해야 합니다.

이 섹션의 절차에서는 브로커 서버에 CephFS를 마운트하는 방법을 보여줍니다.

사전 요구 사항

- 다음이 있습니다.
  - **Red Hat Ceph Storage** 클러스터를 설치 및 구성. 자세한 내용은 **Red Hat Ceph Storage** 설치 및 **Red Hat Ceph Storage** 클러스터 구성을 참조하십시오.
  - **3**개 이상의 **Ceph Metadata Server** 데몬**(ceph-mds)**을 설치 및 구성합니다. 자세한 내용은 메타데이터 서버 설치 및 메타데이터 서버 데몬 구성을 참조하십시오.
  - 모니터 노드에서 **Ceph** 파일 시스템을 생성했습니다. 자세한 내용은 **Ceph** 파일 시스템 생성을 참조하십시오.
  - 브로커 서버가 권한 있는 액세스에 사용할 수 있는 키를 사용하여 **Ceph File System** 클라이언트 사용자를 생성했습니다. 자세한 내용은 **Ceph** 파일 시스템 클라이언트 사용자 생성을 참조하십시오.

프로세스

브로커 서버에 **Ceph** 파일 시스템을 마운트하는 방법에 대한 자세한 내용은 **Ceph** 파일 시스템 마운트를 커널 클라이언트로 참조하십시오.

15.5. 다중 사이트 내결함성 메시징 시스템에서 브로커 구성

브로커를 다중 사이트 내결함성 메시징 시스템의 일부로 구성하려면 다음을 수행해야 합니다.

- 데이터 센터 실패 시 라이브 브로커에서 인수할 유휴 백업 브로커 추가
- **Ceph** 클라이언트 역할을 사용하여 모든 브로커 서버를 구성
- **Ceph File System**에서 메시징 데이터를 저장하는 위치를 지정하여 공유 저장소 **HA(**고가용성**)** 정책을 사용하도록 각 브로커를 구성합니다.

15.5.1. 백업 브로커 추가

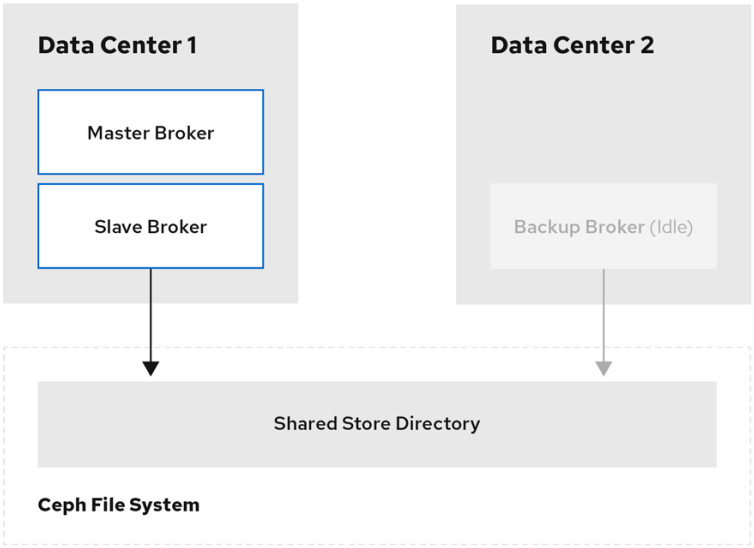
각 데이터 센터 내에서 데이터 센터 중단 시 종료되는 실시간 마스터 슬레이브 브로커 그룹에서 인수할 수 있는 유휴 백업 브로커를 추가해야 합니다. 유휴 백업 브로커에서 라이브 마스터 브로커의 구성을 복제

해야 합니다. 또한 기존 브로커와 동일한 방식으로 클라이언트 연결을 수락하도록 백업 브로커를 구성해야 합니다.

*이후 절차에서는 기존 **master-slave** 브로커 그룹에 가입하도록 유휴 백업 브로커를 구성하는 방법을 참조하십시오. 별도의 데이터 센터에서 라이브 마스터 종속 브로커 그룹의 유휴 백업 브로커를 찾아야 합니다. 또한 데이터 센터 오류 발생 시에만 유휴 백업 브로커를 수동으로 시작하는 것이 좋습니다.*

다음 그림은 예제 토폴로지를 보여줍니다.

*그림 **15.5.** 다중 사이트 내결함성 메시징 시스템의 유휴 백업 브로커*



추가 리소스

- *추가 브로커 인스턴스를 생성하는 방법을 알아보려면 독립 실행형 브로커 생성을 참조하십시오.*
- *브로커 네트워크 연결 구성에 대한 자세한 내용은 **2**장. 네트워크 연결에서 어셉터 및 커넥터 구성 을 참조하십시오.*

15.5.2. 브로커를 Ceph 클라이언트로 구성

*내결함성 시스템에 필요한 백업 브로커를 추가한 경우 **Ceph** 클라이언트 역할을 사용하여 모든 브로커 서버를 구성해야 합니다. 클라이언트 역할을 사용하면 브로커가 **Red Hat Ceph Storage** 클러스터에 데이터를 저장할 수 있습니다.*

Ceph 클라이언트 구성 방법을 알아보려면 Ceph 클라이언트 역할 설치를 참조하십시오.

15.5.3. 공유 저장소 고가용성 구성

Red Hat Ceph Storage 클러스터는 다른 데이터 센터의 브로커가 사용할 수 있는 공유 저장소를 효과적으로 생성합니다. 오류가 발생할 경우 브로커 클라이언트에서 메시지를 계속 사용할 수 있도록 live-backup 그룹의 각 브로커를 사용하도록 구성합니다.

- *공유 저장소 HA(고가용성) 정책*
- *Ceph 파일 시스템의 동일한 저널, 페이징 및 대용량 메시지 디렉터리*

다음 절차에서는 live-backup 그룹의 마스터, 슬레이브 및 유휴 백업 브로커에 공유 저장소 HA 정책을 구성하는 방법을 보여줍니다.

프로세스

1. *live-backup 그룹에 있는 각 브로커의 broker.xml 구성 파일을 편집합니다. Ceph 파일 시스템에서 동일한 페이징, 바인딩, 저널 및 대규모 메시지 디렉터리를 사용하도록 각 브로커를 구성합니다.*

```
# Master Broker - DC1
<paging-directory>mnt/cephfs/broker1/paging</paging-directory>
<bindings-directory>/mnt/cephfs/data/broker1/bindings</bindings-directory>
<journal-directory>/mnt/cephfs/data/broker1/journal</journal-directory>
<large-messages-directory>mnt/cephfs/data/broker1/large-messages</large-messages-directory>

# Slave Broker - DC1
<paging-directory>mnt/cephfs/broker1/paging</paging-directory>
<bindings-directory>/mnt/cephfs/data/broker1/bindings</bindings-directory>
<journal-directory>/mnt/cephfs/data/broker1/journal</journal-directory>
<large-messages-directory>mnt/cephfs/data/broker1/large-messages</large-messages-directory>

# Backup Broker (Idle) - DC2
<paging-directory>mnt/cephfs/broker1/paging</paging-directory>
<bindings-directory>/mnt/cephfs/data/broker1/bindings</bindings-directory>
<journal-directory>/mnt/cephfs/data/broker1/journal</journal-directory>
<large-messages-directory>mnt/cephfs/data/broker1/large-messages</large-messages-directory>
```

2.

다음과 같이 백업 브로커를 **HA** 정책 내에서 마스터로 구성합니다. 이 구성 설정을 사용하면 수동으로 시작할 때 백업 브로커가 마스터가 됩니다. 브로커가 유휴 백업이므로 활성 마스터 브로커에 지정할 수 있는 **failover-on-shutdown** 매개변수가 이 경우 적용되지 않습니다.

```
<configuration>
  <core>
    ...
    <ha-policy>
      <shared-store>
        <master>
        </master>
      </shared-store>
    </ha-policy>
    ...
  </core>
</configuration>
```

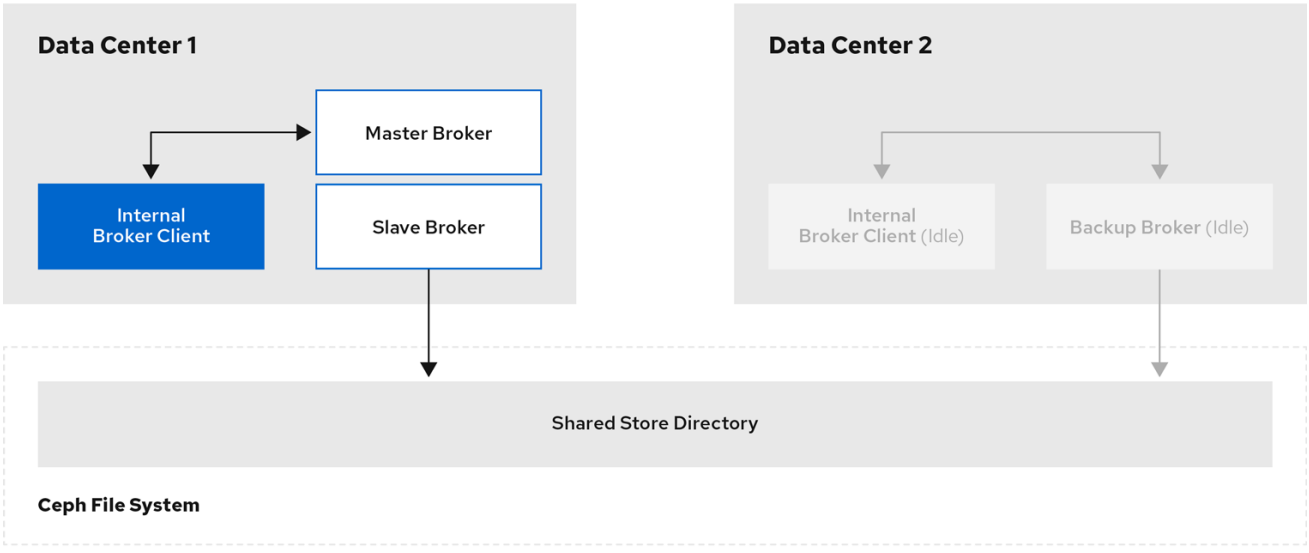
추가 리소스

- 라이브 백업 브로커 그룹에 대한 공유 저장소 고가용성 정책을 구성하는 방법에 대한 자세한 내용은 공유 저장소 고가용성 구성을 참조하십시오.

15.6. 다중 사이트 내결함성 메시징 시스템에서 클라이언트 구성

내부 클라이언트 애플리케이션은 브로커 서버와 동일한 데이터 센터에 있는 시스템에서 실행 중인 애플리케이션입니다. 다음 그림은 이 토폴로지를 보여줍니다.

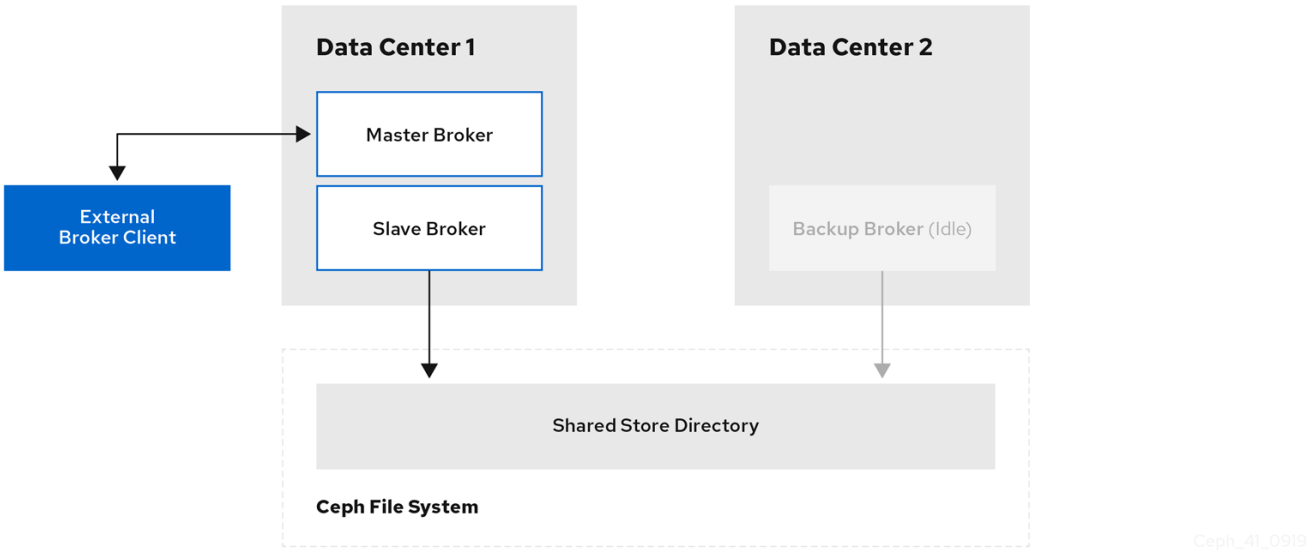
그림 **15.6.** 다중 사이트 내결함성 메시징 시스템의 내부 클라이언트



외부 클라이언트 애플리케이션은 브로커 데이터 센터 외부에 있는 시스템에서 실행 중인 애플리케이션

입니다. 다음 그림은 이 토폴로지를 보여줍니다.

그림 15.7. 다중 사이트 내결함성 메시징 시스템의 외부 클라이언트



다음 하위 섹션에서는 데이터 센터 중단 시 다른 데이터 센터의 백업 브로커에 연결하도록 내부 및 외부 클라이언트 애플리케이션을 구성하는 예를 보여줍니다.

15.6.1. 내부 클라이언트 구성

데이터 센터 중단이 발생하면 브로커와 함께 내부 클라이언트 애플리케이션이 종료됩니다. 이 상황을 완화하려면 별도의 데이터 센터에서 사용 가능한 클라이언트 애플리케이션의 다른 인스턴스가 있어야 합니다. 데이터 센터 중단 시 백업 클라이언트를 수동으로 시작하여 수동으로 시작한 백업 브로커에 연결합니다.

백업 클라이언트가 백업 브로커에 연결할 수 있도록 하려면 기본 데이터 센터의 클라이언트와 유사하게 클라이언트 연결을 구성해야 합니다.

예

master-slave 브로커 그룹에 대한 **AMQ Core Protocol JMS** 클라이언트에 대한 기본 연결 구성은 다음과 같습니다. 이 예에서 **host1** 및 **host2** 는 마스터 및 슬레이브 브로커의 호스트 서버입니다.

```
<ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("(tcp://host1:port,tcp://host2:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

데이터 센터 중단 시 백업 브로커에 연결하도록 백업 클라이언트를 구성하려면 유사한 연결 구성을 사용하지만 백업 브로커 서버의 호스트 이름만 지정합니다. 이 예에서 백업 브로커 서버는 **host3**입니다.

```
<ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("(tcp://host3:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

추가 리소스

- 브로커 네트워크 연결 구성에 대한 자세한 내용은 **2장. 네트워크 연결에서 어셉터 및 커넥터 구성** 을 참조하십시오.

15.6.2. 외부 클라이언트 구성

외부 브로커 클라이언트가 데이터 센터 중단 시 메시징 데이터를 계속 생성하거나 사용하려면 다른 데이터 센터의 브로커로 장애 조치하도록 클라이언트를 구성해야 합니다. 다중 사이트 내결함성 시스템의 경우 중단 시 수동으로 시작하는 백업 브로커로 장애 조치하도록 클라이언트를 구성합니다.

예

다음은 기본 **master-slave** 그룹을 사용할 수 없는 경우 **AMQ Core Protocol JMS** 및 **Apache Cryostat JMS** 클라이언트의 **Red Hat** 빌드가 백업 브로커로 장애 조치되도록 구성하는 예입니다. 이 예제에서 **host1** 및 **host2** 는 기본 마스터 및 슬레이브 브로커의 호스트 서버이며 **host3** 은 데이터 센터 중단 시 수동으로 시작하는 백업 브로커의 호스트 서버입니다.

- **AMQ Core Protocol JMS** 클라이언트를 구성하려면 클라이언트가 연결하려고 하는 정렬된 브로커 목록에 백업 브로커를 포함합니다.

```
<ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("(tcp://host1:port,tcp://host2:port,tcp://host3:port)?
ha=true&retryInterval=100&retryIntervalMultiplier=1.0&reconnectAttempts=-1");
```

- **Red Hat build of Apache Cryostat JMS** 클라이언트를 구성하려면 클라이언트에서 구성하는 장애 조치 **URI**에 백업 브로커를 포함합니다.

```
failover:(amqp://host1:port,amqp://host2:port,amqp://host3:port)?
jms.clientID=myclient&failover.maxReconnectAttempts=20
```

추가 리소스

- 다음의 장애 조치 구성에 대한 자세한 내용은 다음을 참조하십시오.

- **AMQ Core Protocol JMS** 클라이언트의 경우 **Cryostat** 옵션을 참조하십시오.
- **Red Hat build of Apache Cryostat JMS** 클라이언트의 경우 **Cryostat** 옵션을 참조하십시오.
- 기타 지원되는 클라이언트는 **Red Hat AMQ Clients** 제품 설명서의 클라이언트별 문서를 참조하십시오.

15.7. 데이터 센터 중단 중 스토리지 클러스터 상태 확인

내결함성을 위해 **Red Hat Ceph Storage** 클러스터를 구성하면 데이터 센터 중 하나가 실패한 경우에도 클러스터가 성능이 저하된 상태로 계속 실행됩니다.

다음 절차에서는 성능이 저하된 상태로 실행되는 동안 클러스터의 상태를 확인하는 방법을 보여줍니다.

프로세스

1. **Ceph** 스토리지 클러스터의 상태를 확인하려면 **health** 또는 **status** 명령을 사용합니다.

```
# ceph health
# ceph status
```

2. 명령줄에서 클러스터의 지속적인 이벤트를 보려면 새 터미널을 엽니다. 그런 다음 다음을 입력합니다.

```
# ceph -w
```

이전 명령을 실행하면 스토리지 클러스터가 여전히 실행 중이지만 성능이 저하된 상태에 있음을 나타내는 출력이 표시됩니다. 특히 다음과 유사한 경고가 표시됩니다.

```
health: HEALTH_WARN
        2 osds down
        Degraded data redundancy: 42/84 objects degraded (50.0%), 16 pgs unclean, 16 pgs
degraded
```

추가 리소스

- **Red Hat Ceph Storage** 클러스터의 상태를 모니터링하는 방법에 대한 자세한 내용은 **Monitoring** 을 참조하십시오.

15.8. 데이터 센터 중단 중 메시징 연속성 유지

다음 절차에서는 데이터 센터 중단 중에 클라이언트가 브로커 및 관련 메시징 데이터를 사용할 수 있도록 유지하는 방법을 보여줍니다. 특히 데이터 센터가 실패하면 다음을 수행해야 합니다.

- 실패한 데이터 센터의 브로커에서 인수하기 위해 생성한 유휴 백업 브로커를 수동으로 시작합니다.
- 내부 또는 외부 클라이언트를 새 활성 브로커에 연결합니다.

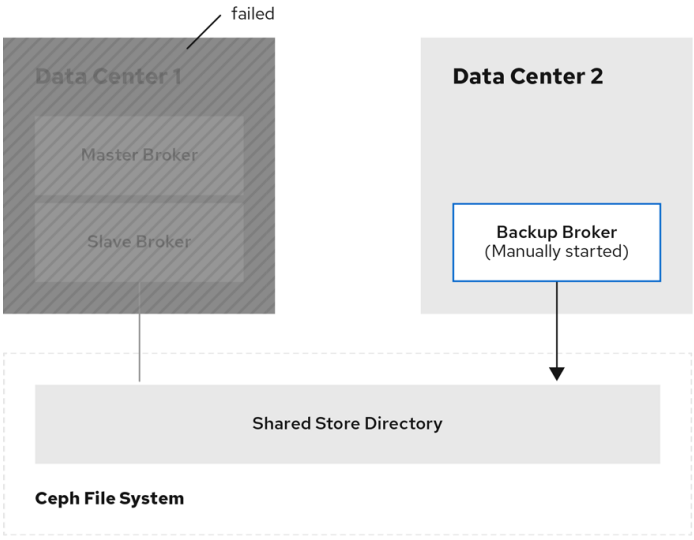
사전 요구 사항

- 다음이 있어야 합니다.
  - **Red Hat Ceph Storage** 클러스터를 설치 및 구성. 자세한 내용은 **Red Hat Ceph Storage** *설치* 및 **Red Hat Ceph Storage** *클러스터 구성을* 참조하십시오.
  - **Ceph** 파일 시스템을 마운트합니다. 자세한 내용은 *브로커 서버에* **Ceph** *파일 시스템 마운트를* 참조하십시오.
  - 데이터 센터 오류가 발생할 경우 라이브 브로커에서 인수하기 위해 유휴 백업 브로커를 추가했습니다. 자세한 내용은 *백업 브로커 추가* 를 참조하십시오.
  - **Ceph** 클라이언트 역할을 사용하여 브로커 서버를 구성합니다. 자세한 내용은 **Ceph** *클라이언트로 브로커 구성을* 참조하십시오.
  - 각 브로커가 공유 저장소 **HA(**고가용성) 정책을 사용하도록 구성하여 각 브로커가 메시징 데이터를 저장할 위치를 지정합니다. 자세한 내용은 *공유 저장소 고가용성 구성을* 참조하십시오.
  - 데이터 센터 중단 시 백업 브로커에 연결하도록 클라이언트를 구성합니다. 자세한 내용은 *다중 사이트 내결함성 메시징 시스템에서 클라이언트 구성* 을 참조하십시오.

프로세스

1. *실패한 데이터 센터의 각 마스터-슬레이브 브로커 쌍에 대해 추가한 유휴 백업 브로커를 수동으로 시작합니다.*

그림 15.8. 데이터 센터 중단 후 백업 브로커 시작

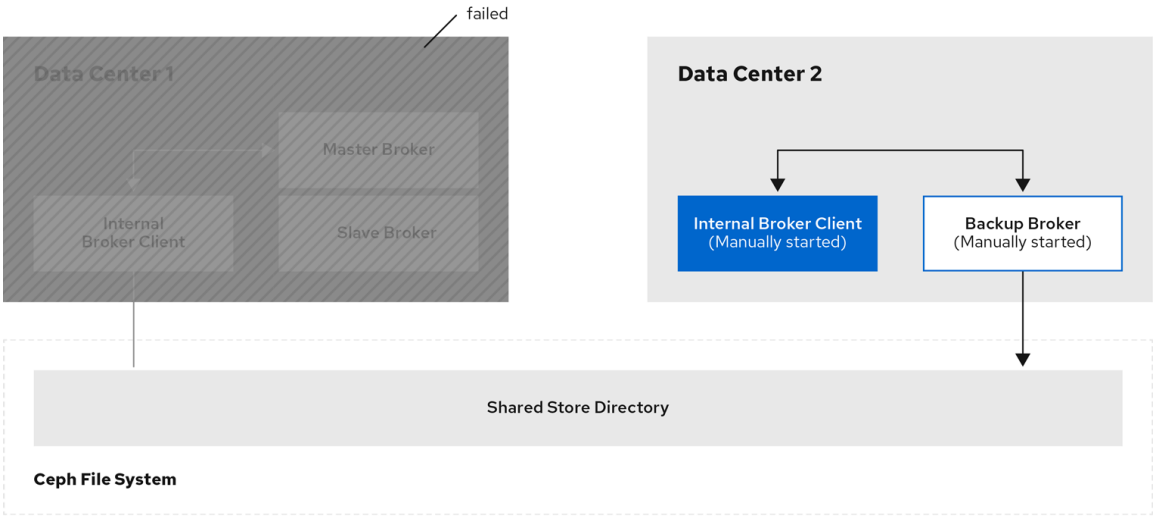


2. *클라이언트 연결을 다시 설정합니다.*

a. *실패한 데이터 센터에서 내부 클라이언트를 사용하는 경우 생성한 백업 클라이언트를 수동으로 시작합니다. 다중 사이트 내결함성 메시징 시스템에서 클라이언트 구성에 설명된 대로 수동으로 시작한 백업 브로커에 연결하도록 클라이언트를 구성해야 합니다.*

다음 그림은 새 토폴로지를 보여줍니다.

그림 15.9. 데이터 센터 중단 후 백업 브로커에 연결된 내부 클라이언트

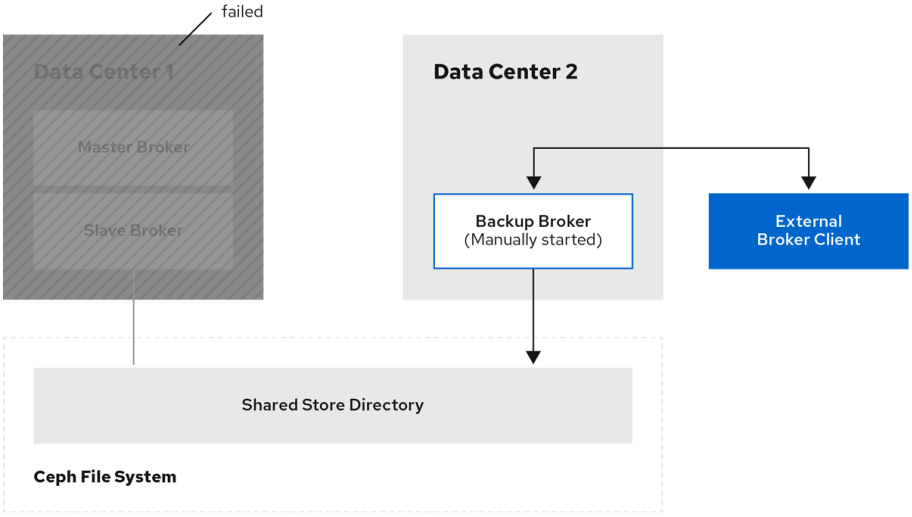


b.

외부 클라이언트가 있는 경우 외부 클라이언트를 수동으로 새 활성 브로커에 연결하거나 클라이언트가 구성을 기반으로 새 활성 브로커에 자동으로 실패하는지 확인합니다. 자세한 내용은 외부 클라이언트 구성을 참조하십시오.

다음 그림은 새 토폴로지를 보여줍니다.

그림 15.10. 데이터 센터 중단 후 백업 브로커에 연결된 외부 클라이언트



15.9. 이전에 실패한 데이터 센터 다시 시작

이전에 실패한 데이터 센터가 다시 온라인 상태가 되면 다음 단계를 수행하여 메시징 시스템의 원래 상태를 복원합니다.

- **Red Hat Ceph Storage** 클러스터의 노드를 호스팅하는 서버를 재시작
- 메시징 시스템에서 브로커 다시 시작
- 클라이언트 애플리케이션에서 복원된 브로커로의 연결 복원

다음 하위 섹션에서는 이러한 단계를 수행하는 방법을 보여줍니다.

15.9.1. 스토리지 클러스터 서버 다시 시작

이전에 실패한 데이터 센터에서 **Monitor, Metadata Server, Manager, Object Storage Device(OSD)** 노드를 다시 시작할 때 **Red Hat Ceph Storage** 클러스터 자체 복구로 전체 데이터 중복성을 복원합니다. 이 프로세스 중에 **Red Hat Ceph Storage**는 필요에 따라 데이터를 복원된 **OSD** 노드로 자동으로 백필합니다.

스토리지 클러스터가 자동으로 자동 복구되고 전체 데이터 중복성을 복원했는지 확인하려면 *데이터 센터 중단 중에 스토리지 클러스터 상태 확인에* 이전에 표시된 명령을 사용합니다. 이러한 명령을 다시 실행하면 이전 **HEALTH_WARN** 메시지에 표시된 백분율이 **100%**로 돌아올 때까지 개선되기 시작합니다.

15.9.2. 브로커 서버 다시 시작

다음 절차에서는 스토리지 클러스터가 더 이상 성능이 저하된 상태에서 작동하지 않는 경우 브로커 서버를 다시 시작하는 방법을 보여줍니다.

프로세스

1. 데이터 센터 중단이 발생할 때 수동으로 시작한 백업 브로커에 연결된 클라이언트 애플리케이션을 중지합니다.
2. 수동으로 시작한 백업 브로커를 중지합니다.
   a. **Linux**의 경우:

```
<broker_instance_dir>/bin/artemis stop
```

b.

Windows에서:

```
<broker_instance_dir>\bin\artemis-service.exe stop
```

3.

이전에 실패한 데이터 센터에서 원래 마스터 및 슬레이브 브로커를 다시 시작합니다.

a.

Linux의 경우:

```
<broker_instance_dir>/bin/artemis run
```

b.

Windows에서:

```
<broker_instance_dir>\bin\artemis-service.exe start
```

원래 마스터 브로커는 다시 시작할 때 마스터로 해당 역할을 자동으로 다시 시작합니다.

15.9.3. 클라이언트 연결 다시 설정

브로커 서버를 다시 시작하면 클라이언트 애플리케이션을 해당 브로커에 다시 연결합니다. 다음 하위 섹션에서는 내부 및 외부 클라이언트 애플리케이션을 다시 연결하는 방법을 설명합니다.

15.9.3.1. 내부 클라이언트 다시 연결

내부 클라이언트는 복원된 브로커와 이전에 실패한 데이터 센터에서 동일한 실행 중인 클라이언트입니다. 내부 클라이언트를 다시 연결하려면 다시 시작하십시오. 각 클라이언트 애플리케이션은 연결 구성에 지정된 복원된 마스터 브로커에 다시 연결됩니다.

브로커 네트워크 연결 구성에 대한 자세한 내용은 2장. 네트워크 연결에서 어셉터 및 커넥터 구성 을 참조하십시오.

15.9.3.2. 외부 클라이언트 다시 연결

외부 클라이언트는 이전에 실패한 데이터 센터 외부에서 실행 중인 클라이언트입니다. 클라이언트 유형 및 외부 브로커 클라이언트 구성 의 정보에 따라 클라이언트가 백업 브로커로 자동으로 실패하도록 구성하거나 이 연결을 수동으로 설정했습니다. 이전에 실패한 데이터 센터를 복원할 때 아래에 설명된 대로 유사한 방식으로 클라이언트에서 복원된 마스터 브로커로 연결을 다시 설정합니다.

- *백업 브로커로 자동으로 장애 조치되도록 외부 클라이언트를 구성한 경우 백업 브로커를 종료하고 원래 마스터 브로커를 다시 시작할 때 클라이언트가 원래 마스터 브로커로 자동으로 실패합니다.*

- *데이터 센터 중단이 발생했을 때 외부 클라이언트를 백업 브로커에 수동으로 연결하는 경우 다시 시작한 원래 마스터 브로커에 클라이언트를 수동으로 다시 연결해야 합니다.*

16장. 브로커 연결을 사용하여 다중 사이트 내결함성 메시징 시스템 구성

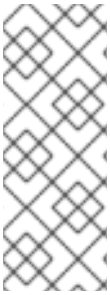
대규모 엔터프라이즈 메시징 시스템에는 일반적으로 지리적으로 분산된 데이터 센터에 있는 별도의 브로커 클러스터가 있습니다. 데이터 센터 중단 시 시스템 관리자는 기존 메시징 데이터를 보존하고 클라이언트 애플리케이션이 메시지를 계속 생성하고 사용할 수 있도록 해야 할 수 있습니다. 브로커 연결을 사용하여 데이터 센터 중단 중에 메시징 시스템의 연속성을 보장할 수 있습니다. 이러한 유형의 솔루션을 다중 사이트 내결함성 아키텍처 라고 합니다.

참고

브로커 연결 브로커 간 통신에는 **AMQP** 프로토콜만 지원됩니다. 클라이언트는 지원되는 모든 프로토콜을 사용할 수 있습니다. 현재 메시지는 미러링 프로세스를 통해 **AMQP**로 변환됩니다.

다음 섹션에서는 브로커 연결을 사용하여 데이터 센터 중단으로부터 메시징 시스템을 보호하는 방법을 설명합니다.

- 16.1절. “브로커 연결 정보”
- 16.2절. “브로커 미러링 구성”



참고

다중 사이트 내결함성은 데이터 센터 내에서 **HA**(고가용성) 브로커 중복을 대체하지 않습니다. 라이브 백업 그룹을 기반으로 하는 브로커 중복성은 단일 클러스터 내의 단일 브로커 오류에 대한 자동 보호 기능을 제공합니다. 반면 다중 사이트 내결함성은 대규모 데이터 센터 중단을 방지합니다.

16.1. 브로커 연결 정보

브로커 연결을 사용하면 브로커가 다른 브로커에 대한 연결을 설정하고 해당 브로커와의 메시지를 미러링 할 수 있습니다.

AMQP 서버 연결

브로커는 브로커 연결을 사용하여 **AMQP** 프로토콜을 사용하여 다른 엔드포인트에 대한 연결을 시작할 수 있습니다. 예를 들어 브로커가 다른 **AMQP** 서버에 연결하고 해당 연결에 요소를 생성할 수 있습니다.

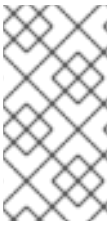
AMQP 서버 연결에서 다음과 같은 유형의 작업이 지원됩니다.

- *미러 - 브로커는 다른 브로커에 대한 AMQP 연결을 사용하고 메시지를 복제하고 전선을 통해 승인을 보냅니다.*
- *보낸 사람 - 특정 큐에서 수신된 메시지는 다른 브로커로 전송됩니다.*
- *수신자 - 브로커가 다른 브로커에서 메시지를 가져옵니다.*
- *피어 - 브로커는 AMQ 상호 연결 끝점에서 발신자와 수신자를 모두 생성합니다.*

이 장에서는 브로커 연결을 사용하여 내결함성 시스템을 생성하는 방법을 설명합니다. 보낸 사람, 수신자 및 피어 옵션에 대한 자세한 내용은 17장. 브로커 브리징 을 참조하십시오.

미러링을 통해 다음 이벤트가 전송됩니다.

- *메시지 전송 - 하나의 브로커로 전송되는 메시지는 대상 브로커에 "복제"됩니다.*
- *메시지 확인 - 한 브로커에서 메시지를 제거하는 확인 사항이 대상 브로커로 전송됩니다.*
- *큐 및 주소 생성.*
- *큐 및 주소 삭제*



참고

메시지가 대상 미러에서 소비자를 위해 보류 중이면 승인이 성공하지 못하고 두 브로커 모두에 의해 메시지가 전달될 수 있습니다.

미러링은 작업을 차단하지 않으며 브로커의 성능에 영향을 미치지 않습니다.

브로커는 미러가 구성된 시점부터 도착한 메시지만 미러링합니다. 기존 메시지는 다른 브로커에 전달되지 않습니다.

16.2. 브로커 미러링 구성

브로커 연결을 사용하여 한 쌍의 브로커 간에 메시지를 미러링할 수 있습니다. 브로커는 언제든지 하나만 활성화할 수 있습니다.

사전 요구 사항

- 작업 브로커가 두 개 있습니다.

프로세스

1. 첫 번째 브로커의 **broker.xml** 파일에 **broker-connections** 요소를 생성합니다. 예를 들면 다음과 같습니다.

```
<broker-connections>
 <amqp-connection uri="tcp://<hostname>:<port>" name="DC1">
  <mirror/>
 </amqp-connection>
</broker-connections>
```

<hostname>

다른 브로커 인스턴스의 호스트 이름입니다.

<port>

다른 호스트의 브로커가 사용하는 포트입니다.

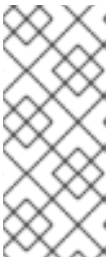
첫 번째 브로커의 모든 메시지는 두 번째 브로커에 미러링되지만 미러가 생성되기 전에 존재하는 메시지는 미러링되지 않습니다.

2. 첫 번째 브로커가 재해 복구에 대해 미러링된 브로커가 최신 상태임을 확인하기 위해 다음 예와 같이 첫 번째 브로커가 메시지를 동기적으로 미러링하려면 브로커의 **amqp-connection** 요소에 **sync=true** 속성을 설정합니다.

동기 미러링을 수행하려면 브로커가 미러링된 브로커로 전송한 메시지를 두 브로커의 볼륨에 동시에 기록해야 합니다. 두 브로커 모두에서 쓰기 작업이 완료되면 소스 브로커는 쓰기 요청이

완료되고 제어가 클라이언트에 반환됨을 확인합니다.

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror sync="true"/>
  </amqp-connection>
</broker-connections>
```



참고

예를 들어 브로커를 사용할 수 없는 경우와 같이 미러링된 브로커에서 쓰기 요청을 완료할 수 없는 경우 최신 쓰기 요청을 완료하기 위해 미러를 사용할 수 있을 때까지 클라이언트 연결이 차단됩니다.

참고

*예제의 브로커 연결 이름은 **$ACTIVEMQ_ARTEMIS_MIRROR_mirror** 라는 큐를 생성하는 데 사용됩니다. 큐가 해당 브로커에 표시되지 않더라도 해당 브로커가 해당 메시지를 수락하도록 구성되어 있는지 확인합니다.*

3. *두 번째 브로커의 **broker.xml** 파일에 **broker-connections** 요소를 생성합니다. 예를 들면 다음과 같습니다.*

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror/>
  </amqp-connection>
</broker-connections>
```

4. *두 번째 브로커가 메시지를 동기적으로 미러링하려면 브로커의 **amqp-connection** 요소에서 **sync=true** 속성을 설정합니다. 예를 들면 다음과 같습니다.*

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror sync="true"/>
  </amqp-connection>
</broker-connections>
```

5. *(선택 사항) 필요에 따라 미러에 대해 다음 매개변수를 구성합니다.*

queue-removal

*큐 또는 주소 제거 이벤트가 전송되는지 여부를 지정합니다. 기본값은 **true**입니다.*

메시지 확인

*메시지 승인이 전송되는지 여부를 지정합니다. 기본값은 **true**입니다.*

queue-creation

*큐 또는 주소 생성 이벤트가 전송되는지 여부를 지정합니다. 기본값은 **true**입니다.*

예를 들면 다음과 같습니다.

```
<broker-connections>
  <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
    <mirror sync="true" queue-removal="false" message-acknowledgments ="false" queue-
creation="false"/>
  </amqp-connection>
</broker-connections>
```

6. *(선택 사항) 브로커 재시도 시도를 대상 미러의 메시지 승인 시도를 사용자 지정합니다.*

큐 메모리에 없는 메시지의 대상 미러에서 승인이 수신될 수 있습니다. 브로커에게 대상 미러에서 메시지를 승인할 충분한 시간을 제공하기 위해 환경에 대해 다음 매개변수를 사용자 지정할 수 있습니다.

mirrorAckManagerQueueAttempts

*브로커가 메모리에서 메시지를 찾는 시도 횟수입니다. 기본값은 **5** 입니다. 지정된 횟수의 시도 후 브로커가 메모리에 있는 메시지를 찾지 못하면 브로커는 페이지 파일에서 메시지를 검색합니다.*

mirrorAckManagerPageAttempts

*메모리에서 메시지를 찾을 수 없는 경우 브로커가 페이지 파일에서 메시지를 찾는 시도 횟수입니다. 기본값은 **2**입니다.*

mirrorAckManagerRetryDelay

브로커가 메모리에서 승인할 메시지를 찾은 다음 페이지 파일에서 시도하는 간격(밀리초)입니다.

***broker-connections** 요소 외부에서 이러한 매개변수 중 하나를 지정합니다. 예를 들면 다음과 같습니다.*

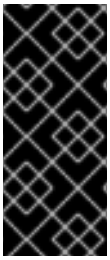
```
<mirrorAckManagerQueueAttempts>8</mirrorAckManagerQueueAttempts>

<broker-connections>
 <amqp-connection uri="tcp://<hostname>:<port>" name="DC2">
  <mirror/>
 </amqp-connection>
</broker-connections>
```

7. *(선택 사항) 메시지가 대상 미러에서 호출되는 경우 브로커가 메시지 파일에 메시지를 작성하여 중복 탐지 정보 쓰기를 조정하려면 **mirrorPageTransaction** 을 **true** 로 설정합니다.*

   ***mirrorPageTransaction** 속성이 기본값인 **false** 로 설정되고 브로커 간에 통신 오류가 발생하면 드문 경우 대상 미러에 중복 메시지를 쓸 수 있습니다.*

   *이 매개변수를 **true** 로 설정하면 브로커의 메모리 사용량이 증가합니다.*

8. *15.6절. “다중 사이트 내결함성 메시징 시스템에서 클라이언트 구성” 에 설명된 지침을 사용하여 브로커 연결과 함께 이를 통해 공유 스토리지가 없습니다.*



중요

***Red Hat**은 미러 구성에서 두 브로커의 메시지를 사용하는 클라이언트 애플리케이션을 지원하지 않습니다. 클라이언트가 두 브로커 모두에서 메시지를 사용하지 않도록 하려면 브로커 중 하나에서 클라이언트 수락을 비활성화합니다.*

17장. 브로커 브리징

브리지는 두 브로커를 연결하고 메시지 간에 메시지를 전달하는 방법을 제공합니다.

다음 브릿지를 사용할 수 있습니다.

코어

core-bridge 예제에서는 하나의 브로커에 배포된 코어 브릿지를 보여줍니다. 이 브릿지는 로컬 대기열의 메시지를 사용하고 두 번째 브로커의 주소로 전달합니다.

mirror

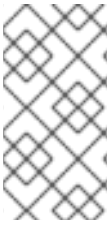
참조 16장. 브로커 연결을 사용하여 다중 사이트 내결함성 메시징 시스템 구성

보낸 사람 및 수신자

참조 17.1절. “브로커 연결에 대한 보낸 사람 및 수신자 구성”

피어

참조 17.2절. “브로커 연결에 대한 피어 구성”



참고

Core 브리지의 **broker.xml** 요소는 브리지 입니다. 다른 브리징 기술은 **< broker-connection>** 요소를 사용합니다.

17.1. 브로커 연결에 대한 보낸 사람 및 수신자 구성

broker.xml 의 **<broker -connections>** 섹션에 보낸 사람 또는 수신자 브로커 연결 요소를 생성하여 브로커를 다른 브로커에 연결할 수 있습니다.

보낸 사람의 경우 브로커는 다른 브로커에 메시지를 보내는 큐에 메시지 소비자를 생성합니다.

수신자 의 경우 브로커는 다른 브로커에서 메시지를 수신하는 주소에 메시지 생산자를 생성합니다.

두 요소는 모두 메시지 브리지로 작동합니다. 그러나 메시지를 처리하는 데 필요한 추가 오버헤드는 없

습니다. 발신자와 수신자는 브로커의 다른 소비자 또는 생산자와 마찬가지로 작동합니다.

특정 대기열은 발신자 또는 수신자가 구성할 수 있습니다. 와일드카드 표현식은 발신자와 수신자를 특정 주소 또는 주소 집합에 일치시키는 데 사용할 수 있습니다. 발신자 또는 수신자를 구성할 때 다음 속성을 설정할 수 있습니다.

- ***address-match:** 와일드카드 표현식을 사용하여 발신자 또는 수신자를 특정 주소 또는 주소 집합과 일치시킵니다.*

- ***queue-name:** 특정 큐에 대한 발신자 또는 수신자를 구성합니다.*

- *주소 표현식 사용:*

```
<broker-connections>
  <amqp-connection uri="tcp://HOST:PORT" name="other-server">
    <sender address-match="queues.#"/>
    <!-- notice the local queues for remotequeues.# need to be created on this broker -->
    <receiver address-match="remotequeues.#"/>
  </amqp-connection>
</broker-connections>

<addresses>
  <address name="remotequeues.A">
    <anycast>
      <queue name="remoteQueueA"/>
    </anycast>
  </address>
  <address name="queues.B">
    <anycast>
      <queue name="localQueueB"/>
    </anycast>
  </address>
</addresses>
```

- *큐 이름 사용:*

```
<broker-connections>
  <amqp-connection uri="tcp://HOST:PORT" name="other-server">
    <receiver queue-name="remoteQueueA"/>
    <sender queue-name="localQueueB"/>
  </amqp-connection>
</broker-connections>

<addresses>
  <address name="remotequeues.A">
```

```
    <anycast>
      <queue name="remoteQueueA"/>
    </anycast>
  </address>
  <address name="queues.B">
    <anycast>
      <queue name="localQueueB"/>
    </anycast>
  </address>
</addresses>
```

참고

수신자는 이미 존재하는 로컬 큐와만 일치할 수 있습니다. 따라서 수신자를 사용하는 경우 대기열이 로컬에 미리 생성되어 있는지 확인합니다. 그러지 않으면 브로커가 원격 대기열 및 주소와 일치할 수 없습니다.

참고

전송 및 수신에 대한 무한 루프를 생성하므로 동일한 대상이 있는 발신자와 수신자를 생성하지 마십시오.

17.2. 브로커 연결에 대한 피어 구성

브로커는 **AMQ Interconnect** 인스턴스에 연결하는 피어로 구성하고 브로커가 해당 라우터에 구성된 지정된 **AMQP** 방법 포인트 주소에 대한 저장소 및 전달 대기열 역할을 수행하도록 지시할 수 있습니다. 이 시나리오에서 클라이언트는 라우터에 연결하여 방법 주소를 사용하여 메시지를 보내고 수신하며 라우터는 브로커의 대기열에서 또는 브로커의 큐에서 이러한 메시지를 라우팅합니다.

이 피어 구성은 브로커의 브로커 연결 구성에 일치하는 각 대상에 대해 발신자 및 수신자 쌍을 생성합니다. 이러한 쌍에는 라우터가 브로커와 협업할 수 있는 구성이 포함됩니다. 이 기능은 라우터가 연결을 시작하고 자동 링크를 생성하는 데 필요한 요구 사항을 방지합니다.

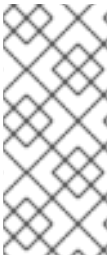
피어 구성을 사용하면 발신자 및 수신자가 있는 경우와 동일한 속성이 표시됩니다. 예를 들어, 이름이 있는 큐가 있는 구성은 큐 를 시작합니다. 일치하는 라우터 방식 주소에 대한 스토리지 역할을 하는 구성은 다음과 같습니다.

```
<broker-connections>
  <amqp-connection uri="tcp://HOST:PORT" name="router">
    <peer address-match="queues.#"/>
  </amqp-connection>
</broker-connections>

<addresses>
```

```
    <address name="queues.A">
      <anycast>
        <queue name="queues.A"/>
      </anycast>
    </address>
    <address name="queues.B">
      <anycast>
        <queue name="queues.B"/>
      </anycast>
    </address>
  </addresses>
```

라우터에 일치하는 주소 방법 구성이 있어야 합니다. 이렇게 하면 브로커가 연결된 특정 라우터 주소를 웨이포인트로 처리하도록 지시합니다. 예를 들어 다음 접두사 기반 라우터 주소 구성을 참조하십시오.

```
address {
    prefix: queue
    waypoint: yes
}
```



참고

다른 브로커에 직접 연결하는 데 피어 옵션을 사용하지 마십시오. 이 옵션을 사용하여 다른 브로커에 연결하면 모든 메시지가 즉시 사용할 준비가 되어 전송 및 수신에 대한 무한 에코를 생성합니다.

18장. 로깅

AMQ Broker는 Apache Log4j 2 로깅 유틸리티를 사용하여 메시지 로깅을 제공합니다. 브로커를 설치할 때 < broker_instance_dir> /etc/log4j2.properties 파일에 기본 Log4j 2 구성이 있습니다. 기본 구성을 사용하면 로거는 콘솔과 파일에 모두 씁니다.

AMQ Broker에서 사용할 수 있는 로거는 다음 표에 표시되어 있습니다.

표 18.1. AMQ Broker 로거

로거	설명
org.apache.activemq.artemis.core.server	브로커 코어 로그
org.apache.activemq.artemis.journal	로그 저널 호출
org.apache.activemq.artemis.utils	유틸리티 호출 로그
org.apache.activemq.artemis.jms	JMS 호출 로그
org.apache.activemq.artemis.integration.bootstrap	부트스트랩 호출 로그
org.apache.activemq.audit.base	모든 Cryostat 개체 메서드에 대한 액세스 로그
org.apache.activemq.audit.message	프로덕션, 사용 및 메시지 검색과 같은 메시지 작업을 기록합니다.
org.apache.activemq.audit.resource	인증 이벤트를 기록, Cryostat 또는 AMQ Broker 관리 콘솔에서 브로커 리소스 생성 또는 삭제, 관리 콘솔에서 메시지 검색

18.1. 로깅 수준 변경

apache.activemq.artemis.core .server 로거에 대한 다음 예와 같이 로거 이름 뒤에 <logger name>.level 줄에 있는 각 로거의 로깅 수준을 구성할 수 있습니다.

```
logger.artemis_server.name=org.apache.activemq.artemis.core.server
logger.artemis_server.level=INFO
```

감사 로거의 기본 로깅 수준은 OFF 입니다. 즉, 로깅이 비활성화됩니다. AMQ Broker에서 사용할 수 있는 다른 로거의 기본 로깅 수준은 INFO 입니다. Log4j 2에서 사용할 수 있는 로깅 수준에 대한 자세한 내용은 Log4j 2 설명서를 참조하십시오.

18.2. 감사 로깅 활성화

기본 감사 로거, 메시지 감사 로거, 리소스 감사 로거 등 세 가지 감사 로거를 활성화할 수 있습니다.

기본 감사 로거(**org.apache.activemq.audit.base**)

주소 및 큐 생성 및 삭제와 같은 모든 **Cryostat** 개체 메서드에 대한 액세스를 기록합니다. 로그 는 이러한 작업이 성공했는지 또는 실패했는지 여부를 나타내지 않습니다.

메시지 감사 로거(**org.apache.activemq.audit.message**)

메시지 관련 브로커 작업(예: 프로덕션, 사용 또는 메시지 검색)을 기록합니다.

리소스 감사 로거(**org.apache.activemq.audit.resource**)

클라이언트, 경로 및 **AMQ Broker** 관리 콘솔에서 인증 성공 또는 실패를 기록합니다. 또한 **Cryostat** 또는 관리 콘솔에서 큐 생성, 업데이트 또는 삭제를 로깅하고 관리 콘솔에서 메시지 검색도 기록합니다.

다른 사용자와 독립적으로 각 감사 로거를 활성화할 수 있습니다. 기본적으로 로깅 수준은 **OFF** 로 설정됩니다. 즉, 각 감사 로거에 대해 로깅이 비활성화됩니다. 감사 로거 중 하나를 활성화하려면 로깅 수준을 **OFF** 에서 **INFO** 로 변경합니다. 예를 들면 다음과 같습니다.

```
logger.audit_base = INFO, audit_log_file
```

참고

INFO 는 **logger.org.apache.activemq.audit.base,logger.org.apache.activemq.audit.message** 및 **logger.org.apache.activemq.audit.resource** 감사 로거에 사용할 수 있는 유일한 로깅 수준입니다.

중요

메시지 감사 로거는 브로커의 성능 집약적인 경로에서 실행됩니다. 특히 브로커가 높은 메시징 부하에서 실행되는 경우 로거를 활성화하면 브로커의 성능에 부정적인 영향을 미칠 수 있습니다. 처리량이 높은 메시징 시스템에서 감사 로깅을 활성화하지 않는 것이 좋습니다.

18.3. 클라이언트 또는 임베디드 서버 로깅

클라이언트에서 로깅을 활성화하려면 **SLF4J facade**를 지원하는 애플리케이션에 로깅 구현을 포함해

야 합니다. Maven을 사용하는 경우 Log4j 2에 대해 다음 종속성을 추가합니다.

```
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>artemis-jms-client</artifactId>
  <version>2.28.0</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-slf4j-impl</artifactId>
  <version>2.19.0</version>
</dependency>
```

Log4j 2 구성을 classpath의 log4j2.properties 파일에 제공할 수 있습니다. 또는 log4j2.configurationFile 시스템 속성을 사용하여 사용자 지정 구성 파일을 지정할 수 있습니다. 예를 들면 다음과 같습니다.

```
-Dlog4j2.configurationFile=file:///path/to/custom-log4j2-config.properties
```

다음은 클라이언트의 log4j2.properties 파일의 예입니다.

```
# Log4J 2 configuration

# Monitor config file every X seconds for updates
monitorInterval = 5

rootLogger = INFO, console, log_file

logger.activemq.name=org.apache.activemq
logger.activemq.level=INFO

# Console appender
appender.console.type=Console
appender.console.name=console
appender.console.layout.type=PatternLayout
appender.console.layout.pattern=%d %-5level [%logger] %msg%n

# Log file appender
appender.log_file.type = RollingFile
appender.log_file.name = log_file
appender.log_file.fileName = log/application.log
appender.log_file.filePattern = log/application.log.%d{yyyy-MM-dd}
appender.log_file.layout.type = PatternLayout
appender.log_file.layout.pattern = %d %-5level [%logger] %msg%n
appender.log_file.policies.type = Policies
appender.log_file.policies.cron.type = CronTriggeringPolicy
appender.log_file.policies.cron.schedule = 0 0 0 * * ?
appender.log_file.policies.cron.evaluateOnStartup = true
```

18.4. AMQ BROKER 플러그인 지원

AMQ는 사용자 지정 플러그인을 지원합니다. 플러그인을 사용하여 디버그 로그를 통해서만 사용할 수 있는 다양한 이벤트 유형에 대한 정보를 기록할 수 있습니다. 여러 플러그인을 함께 등록, 연결 및 실행할 수 있습니다. 플러그인은 등록 순서에 따라 실행됩니다. 즉, 등록된 첫 번째 플러그인이 항상 먼저 실행됩니다.

사용자 지정 플러그인을 생성하고 ActiveMQServerPlugin 인터페이스를 사용하여 구현할 수 있습니다. 이 인터페이스는 플러그인이 classpath에 있고 브로커에 등록되어 있는지 확인합니다. 기본적으로 모든 인터페이스 메서드를 구현하므로 구현해야 하는 필수 동작만 추가해야 합니다.

18.4.1. 클래스 경로에 플러그인 추가

관련 .jar 파일을 <broker _instance_dir> /lib 디렉터리에 추가하여 사용자 정의 생성된 브로커 플러그인을 브로커 런타임에 추가합니다.

임베디드 시스템을 사용하는 경우 포함된 애플리케이션의 일반 클래스 경로에 .jar 파일을 배치합니다.

18.4.2. 플러그인 등록

broker.xml 구성 파일에 broker-plugins 요소를 추가하여 플러그인을 등록해야 합니다. 속성 하위 요소를 사용하여 플러그인 구성 값을 지정할 수 있습니다. 이러한 속성은 플러그인이 인스턴스화된 후 플러그인의 init(Map<String, String>) 작업에 읽고 전달됩니다.

```
<broker-plugins>
  <broker-plugin class-name="some.plugin.UserPlugin">
    <property key="property1" value="val_1" />
    <property key="property2" value="val_2" />
  </broker-plugin>
 </broker-plugins>
```

18.4.3. 프로그래밍 방식으로 플러그인 등록

플러그인을 프로그래밍 방식으로 등록하려면 registerBrokerPlugin() 메서드를 사용하고 플러그인의 새 인스턴스를 전달합니다. 아래 예제에서는 UserPlugin 플러그인의 등록을 보여줍니다.

```
Configuration config = new ConfigurationImpl();

config.registerBrokerPlugin(new UserPlugin());
```

18.4.4. 특정 이벤트 로깅

*기본적으로 **AMQ** 브로커는 특정 브로커 이벤트를 로깅하기 위해 **LoggingActiveMQServerPlugin** 플러그인을 제공합니다. **LoggingActiveMQServerplugin** 플러그인은 기본적으로 주석 처리되며 정보를 기록하지 않습니다.*

*다음 표에서는 각 플러그인 속성에 대해 설명합니다. 이벤트를 기록하려면 구성 속성 값을 **true** 로 설정합니다.*

*표 **18.2. LoggingActiveMQServerPlugin** 플러그인 속성*

속성	설명
LOG_CONNECTION_EVENTS	연결이 생성되거나 삭제될 때 정보를 기록합니다.
LOG_SESSION_EVENTS	세션이 생성되거나 닫힐 때 정보를 기록합니다.
LOG_CONSUMER_EVENTS	소비자가 생성되거나 닫힐 때 정보를 기록합니다.
LOG_DELIVERING_EVENTS	메시지가 소비자에게 전달될 때 그리고 소비자가 메시지를 승인할 때 정보를 기록합니다.
LOG_SENDING_EVENTS	메시지가 주소로 전송되고 브로커 내에서 메시지가 라우팅될 때 정보를 기록합니다.
LOG_INTERNAL_EVENTS	큐 생성 또는 삭제 시, 메시지가 만료되면 브리지가 배포되고, 심각한 오류가 발생할 때 정보를 기록합니다.
LOG_ALL_EVENTS	위의 모든 이벤트에 대한 정보를 기록합니다.

*연결 이벤트를 기록하도록 **LoggingActiveMQServerPlugin** 플러그인을 구성하려면 **broker.xml** 구성 파일의 **< broker-plugins >** 섹션의 주석을 제거합니다. 주석으로 처리된 기본 예제에서 모든 이벤트의 값이 **true** 로 설정됩니다.*

```
<configuration ...>
...
<!-- Uncomment the following if you want to use the Standard LoggingActiveMQServerPlugin plugin
to log in events -->
         <broker-plugins>
        <broker-plugin class-
name="org.apache.activemq.artemis.core.server.plugin.impl.LoggingActiveMQServerPlugin">
          <property key="LOG_ALL_EVENTS" value="true"/>
          <property key="LOG_CONNECTION_EVENTS" value="true"/>
          <property key="LOG_SESSION_EVENTS" value="true"/>
          <property key="LOG_CONSUMER_EVENTS" value="true"/>
          <property key="LOG_DELIVERING_EVENTS" value="true"/>
          <property key="LOG_SENDING_EVENTS" value="true"/>
          <property key="LOG_INTERNAL_EVENTS" value="true"/>
```

```
        </broker-plugin>
      </broker-plugins>
...
</configuration>
```

< broker-plugins > 섹션 내에서 구성 매개변수를 변경하는 경우 브로커를 다시 시작하여 구성 업데이트를 다시 로드해야 합니다. 이러한 구성 변경 사항은 **configuration-file-refresh-period** 설정에 따라 다시 로드되지 않습니다.

로그 수준이 **INFO** 로 설정되면 이벤트가 발생한 후 항목이 기록됩니다. 로그 수준이 **DEBUG** 로 설정된 경우 이벤트 전과 후에 로그 항목이 모두 생성됩니다(예: **CreateConsumer()** 및 **afterCreateConsumer())**. 로그 수준이 **DEBUG** 로 설정된 경우 로거는 사용 가능한 경우 알림에 대한 자세한 정보를 기록합니다.

19장. 튜닝 지침

AMQ 브로커를 튜닝하려면 다음 지침을 검토하십시오.

19.1. 지속성 튜닝

지속성 성능 개선에 대한 팁은 다음 정보를 검토하십시오.

- 파일 기반 저널에 메시지를 저장합니다.

  메시지 지속성을 위해 파일 기반 저널을 사용합니다. AMQ Broker는 JDBC(Java Database Connectivity) 데이터베이스에도 메시지를 유지할 수 있지만 파일 기반 저널 사용과 비교하여 성능 비용이 발생합니다.

- 메시지 저널을 자체 물리 볼륨에 배치합니다.

  추가 전용 저널의 이점 중 하나는 디스크 헤드 이동이 최소화된다는 것입니다. 디스크가 공유되면 이 이점이 손실됩니다. 트랜잭션 코디네이터, 데이터베이스 및 기타 저널과 같은 여러 프로세스가 동일한 디스크에서 읽고 쓰는 경우 디스크 헤드가 다른 파일 간에 건너뛰어야 하므로 성능에 영향을 미칩니다. 페이징 또는 큰 메시지를 사용하는 경우 별도의 볼륨에도 배치되었는지 확인합니다.

- journal-min-files 매개변수 값을 조정합니다.

  journal-min-files 매개변수를 평균 지속 가능한 비율에 맞는 파일 수로 설정합니다. 저널 데이터 디렉터리에 새 파일이 자주 생성되는 경우 많은 데이터가 유지되므로 저널에서 유지 관리하는 최소 파일 수를 늘려야 합니다. 이를 통해 저널은 새 데이터 파일을 생성하지 않고 재사용할 수 있습니다.

- 저널 파일 크기를 최적화합니다.

  journal-file-size 매개변수의 값을 디스크의 용량에 맞춥니다. 10MB의 기본값은 대부분의 시스템에서 충분해야 합니다.

- 비동기 IO(AIO) 저널 유형을 사용합니다.

  Linux 운영 체제의 경우 저널 유형을 AIO로 유지합니다. AIO는 Java new I/O(NIO)보다 확장

성이 향상됩니다.

- journal-buffer-timeout 매개변수 값을 조정합니다.

  journal-buffer-timeout 매개변수의 값을 늘리면 대기 시간이 지남에 따라 처리량이 증가합니다.

- journal-max-io 매개변수 값을 조정합니다.

  AIO를 사용하는 경우 journal-max-io 매개변수 값을 늘려 성능을 향상시킬 수 있습니다. NIO를 사용하는 경우 이 값을 변경하지 마십시오.

- journal-pool-files 매개변수를 조정합니다.

  저널 파일 풀의 상위 임계값인 journal-pool-files 매개변수를 예상 최대 로드에 가까운 숫자로 설정합니다. 필요한 경우 저널은 상위 임계값을 초과하지만 가능한 경우 임계값으로 축소됩니다. 이렇게 하면 필요한 것보다 많은 디스크 공간을 소비하지 않고 파일을 재사용할 수 있습니다. 저널 데이터 디렉토리에 새 파일이 너무 자주 생성되는 경우 journal-pool-size 매개변수를 늘립니다. 이 매개변수를 늘리면 저널이 새 파일을 생성하는 대신 더 많은 기존 파일을 재사용할 수 있으므로 성능이 향상됩니다.

- 저널 쓰기 시 지속성 보장이 필요하지 않은 경우 journal-data-sync 매개변수를 비활성화합니다.

  전원 장애가 발생하는 경우 저널 쓰기에 보장된 지속성이 필요하지 않은 경우 journal-data-sync 매개변수를 비활성화하고 성능을 개선하기 위해 journal 유형 NIO 또는 MAPPED 를 사용합니다.

19.2. JMS(JAVA MESSAGE SERVICE) 조정

JMS API를 사용하는 경우 다음 정보를 검토하여 성능을 개선하는 방법에 대한 팁을 검토하십시오.

- 메시지 ID를 비활성화합니다.

  메시지 ID가 필요하지 않은 경우 MessageProducer 클래스에서 setDisableMessageID() 메서드를 사용하여 해당 ID를 비활성화합니다. 값을 true 로 설정하면 고유한 ID를 생성할 필요가 없으며 메시지 크기가 줄어듭니다.

- 메시지 타임스탬프를 비활성화합니다.

  메시지 타임스탬프가 필요하지 않은 경우 **MessageProducer** 클래스에서 **setDisableMessageTimeStamp()** 메서드를 사용하여 비활성화합니다. 값을 **true** 로 설정하면 타임스탬프를 생성하는 오버헤드가 제거되고 메시지 크기가 줄어듭니다.

- **ObjectMessage** 를 사용하지 마십시오.

  ObjectMessage 는 직렬화된 개체가 있는 메시지를 보내는 데 사용됩니다. 즉 메시지 본문 또는 페이로드가 바이트 스트림으로 유선을 통해 전송됩니다. **Java** 직렬화된 형태의 작은 오브젝트도 매우 크며, 유선에서 상당한 공간을 차지합니다. 또한 사용자 지정 마샬링 기술과 비교하면 속도가 느려집니다. 런타임까지 페이로드 유형을 모르는 경우와 같이 다른 메시지 유형 중 하나를 사용할 수 없는 경우에만 **ObjectMessage** 를 사용하십시오.

- **AUTO_ACKNOWLEDGE** 를 피하십시오.

  소비자에 대한 승인 모드를 선택하면 네트워크를 통해 승인 메시지를 전송하여 발생하는 추가 오버헤드 및 트래픽으로 인해 성능에 영향을 미칩니다. **AUTO_ACKNOWLEDGE** 는 클라이언트에 수신된 각 메시지에 대해 서버에서 승인이 전송되어야 하므로 이러한 오버헤드가 발생합니다. 가능한 경우 지연된 방식으로 메시지를 승인하는 **DUPS_OK_ACKNOWLEDGE** 를 사용하거나 **CLIENT_ACKNOWLEDGE** 를 사용합니다. 즉, 클라이언트 코드가 메시지를 확인하는 메서드를 호출합니다. 또는 트랜잭션된 세션에서 하나의 승인 또는 커밋을 사용하여 많은 승인을 일괄 처리합니다.

- **Cryostat** 메시지를 방지합니다.

  기본적으로 **JMS** 메시지는 **Cryostat**입니다. **Cryostat** 메시지가 필요하지 않은 경우 해당 메시지를 확인할 수 없음으로 설정합니다. **Cryostat** 메시지는 스토리지에 유지되므로 추가 오버헤드가 발생합니다.

- **TRANSACTED_SESSION** 모드를 사용하여 단일 트랜잭션으로 메시지를 보내고 받습니다.

  AMQ Broker는 단일 트랜잭션에 메시지를 일괄 처리하여 커밋에 하나의 네트워크 라운드트립만 필요하며, 모든 전송 또는 수신 시에는 필요하지 않습니다.

19.3. 전송 설정 조정

전송 설정 튜닝에 대한 팁은 다음 정보를 검토하십시오.

- 운영 체제가 TCP 자동 튜닝을 지원하는 경우 이후 버전의 Linux에서와 마찬가지로 TCP 전송 및 수신 버퍼 크기를 늘리지 마십시오. 성능 향상을 시도합니다. 자동 튜닝이 있는 시스템에서 버퍼 크기를 수동으로 설정하면 자동 처리가 작동하지 않고 실제로 브로커 성능이 저하될 수 있습니다. 운영 체제가 TCP 자동 튜닝을 지원하지 않고 브로커가 빠른 머신과 네트워크에서 실행 중인 경우 TCP 전송 및 수신 버퍼 크기를 늘려 브로커 성능을 향상시킬 수 있습니다. 자세한 내용은 부록 A. acceptor 및 Connector 구성 매개변수의 내용을 참조하십시오.

- 브로커에서 많은 동시 연결을 예상하거나 클라이언트가 연결을 빠르게 열고 닫는 경우 브로커를 실행하는 사용자에게 충분한 파일 처리를 생성할 수 있는 권한이 있는지 확인합니다. 이 작업을 수행하는 방법은 운영 체제마다 다릅니다. Linux 시스템에서 /etc/security/limits.conf 파일에서 허용되는 오픈 파일 처리 수를 늘릴 수 있습니다. 예를 들어 다음 행을 추가합니다.

```
serveruser  soft  nofile  20000
serveruser  hard  nofile  20000
```

  이 예제에서는 serveruser 사용자가 최대 20000개의 파일 처리를 열 수 있도록 허용합니다.

- batchDelay netty TCP 매개변수 값을 설정하고 directDeliver netty TCP 매개변수를 false로 설정하여 매우 작은 메시지의 처리량을 최대화합니다.

19.4. 브로커 가상 머신 튜닝

다양한 가상 머신 설정을 조정하는 방법에 대한 팁은 다음 정보를 검토하십시오.

- 최상의 성능을 위해 최신 Java 가상 머신을 사용하십시오.

- 서버에 최대한 많은 메모리를 할당합니다.

  AMQ Broker는 페이징을 사용하여 메모리 부족으로 실행할 수 있습니다. 그러나 AMQ Broker에서 모든 큐를 메모리에 유지할 수 있는 경우 성능이 향상됩니다. 필요한 메모리 양은 큐의 크기 및 수와 메시지의 크기와 수에 따라 달라집니다. -Xms 및 -Xmx JVM 인수를 사용하여 사용 가능한 메모리를 설정합니다.

- 힙 크기를 조정합니다.

  로드 기간이 길면 AMQ Broker에서 많은 수의 오브젝트를 생성하고 제거할 수 있으므로 오래된 오브젝트를 빌드할 수 있습니다. 이렇게 하면 메모리 부족 브로커의 위험이 증가하고 전체 가비지 수집이 발생하여 일시 중지 및 의도하지 않은 동작이 발생할 수 있습니다. 이 위험을 줄이려

면 JVM의 최대 힙 크기(-Xmx)가 global-max-size 매개변수 값의 5배 이상으로 설정되어 있는지 확인합니다. 예를 들어 브로커가 부하가 높고 global-max-size 가 1GB인 경우 최대 힙 크기를 5GB로 설정합니다.

19.5. 다른 설정 조정

성능 개선에 대한 추가 팁은 다음 정보를 검토하십시오.

- 비동기 send acknowledgements를 사용합니다.

  비-고속적인 메시지를 보내야 하고 send() 반환 시 서버에 도달했다는 보장이 필요하지 않은 경우, 차단을 전송하도록 설정하지 마십시오. 대신 비동기 send acknowledgements를 사용하여 별도의 스트림에서 반환된 전송 승인을 받습니다. 그러나 서버가 충돌하는 경우 일부 메시지가 손실될 수 있습니다.

- 사전 인증 모드를 사용합니다.

  사전 확인 모드를 사용하면 메시지가 클라이언트에 전송되기 전에 확인됩니다. 이렇게 하면 유선의 승인 트래픽 양이 줄어듭니다. 그러나 클라이언트가 충돌하는 경우 클라이언트가 다시 연결되면 메시지가 다시 전달되지 않습니다.

- 보안을 비활성화합니다.

  보안 사용 매개 변수를 false 로 설정하여 성능이 약간 향상됩니다.

- 지속성을 비활성화합니다.

  persistence-enabled 매개변수를 false 로 설정하여 메시지 지속성을 끌 수 있습니다.

- 트랜잭션을 즉시 동기화합니다.

  journal-sync- Cryostatal 매개변수를 false 로 설정하면 트랜잭션 을 유지할 때 실패 시 트랜잭션 손실이 발생할 때 성능이 향상됩니다.

- 즉시 동기화되지 않습니다.

journal-sync-non- Cryostatal 매개변수를 false 로 설정하면 실패 시의 message loss가 손실될 가능성이 낮을 때 더 나은 성능을 제공합니다.

- *메시지가 차단되지 않도록 보냅니다.*

  전송된 모든 메시지의 네트워크 라운드 트립을 기다리지 않으려면 JMS(Java Messaging Service) 및 JNDI(Java Naming and Directory Interface)를 사용하는 경우 block-on-durable-send 및 block-on-durable-send 매개변수를 false 로 설정합니다. 또는 setBlockOnDurableSend() 및 setBlockOnNonDurableSend() 메서드를 호출하여 ServerLocator 에서 직접 설정합니다.

- *consumer-window-size 를 최적화합니다.*

  매우 빠른 소비자인 경우 consumer-window-size 매개변수의 값을 증가시켜 소비자 흐름 제어를 효과적으로 비활성화할 수 있습니다.

- *JMS API 대신 코어 API를 사용합니다.*

  JMS 작업은 서버가 코어 API를 사용할 때보다 성능이 저하되기 전에 핵심 작업으로 변환되어야 합니다. 코어 API를 사용하는 경우 SimpleString 을 최대한 많이 사용하는 방법을 사용하십시오. simpleString.lang.String과 달리 java.lang.String에는 복사가 필요하지 않습니다. 따라서 호출 간에 SimpleString 인스턴스를 재사용하면 불필요한 복사를 방지할 수 있습니다. 핵심 API는 다른 브로커에 이식할 수 없습니다.

19.6. 안티 패턴 방지

- *연결, 세션, 소비자 및 생산자를 가능한 경우 재사용합니다.*

  가장 일반적인 메시징 안티 패턴은 전송 또는 소비되는 모든 메시지에 대해 새로운 연결, 세션 및 생산자를 생성하는 것입니다. 이러한 오브젝트는 생성하는 데 시간이 걸리며 여러 네트워크 라운드트립이 포함될 수 있으며, 이는 리소스를 제대로 사용하지 못할 수 있습니다.

참고

Spring JMS 템플릿과 같은 일부 널리 사용되는 라이브러리는 이러한 패턴을 사용합니다. Spring JMS 템플릿을 사용하는 경우 성능이 저하될 수 있습니다. Spring JMS 템플릿은 JMS 세션을 캐시하는 애플리케이션 서버(예: Java Connector Architecture 사용)에서만 안전하게 사용할 수 있습니다. 애플리케이션 서버에서도 메시지를 동기적으로 사용하는 데 안전하게 사용할 수 없습니다.

- ***fat message*를 피하십시오.**

  ***XML*과 같은 상세 형식은 유선에서 상당한 공간을 차지하며 결과적으로 성능이 저하됩니다. 가능한 경우 메시지 본문에서 *XML*을 방지할 수 있습니다.**

- **각 요청에 대해 임시 큐를 생성하지 마십시오.**

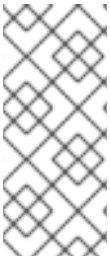
  이러한 일반적인 안티 패턴에는 임시 큐 요청 응답 패턴이 포함됩니다. 임시 큐 요청-응답 패턴을 사용하면 메시지가 대상으로 전송되고 응답 헤더가 로컬 임시 큐의 주소로 설정됩니다. 수신자가 메시지를 수신하면 처리한 다음 *reply-to* 헤더에 지정된 주소로 응답을 보냅니다. 이 패턴으로 인해 발생하는 일반적인 실수는 전송되는 각 메시지에 새 임시 큐를 생성하여 성능을 크게 줄이는 것입니다. 대신 여러 요청에 임시 큐를 재사용해야 합니다.

- **필요한 경우가 아니면 메시지 기반 빈을 사용하지 마십시오.**

  메시지 중심 빈을 사용하여 메시지를 사용하는 것은 간단한 *JMS* 메시지 소비자를 사용하여 메시지를 사용하는 것보다 느립니다.

부록 A. ACCEPTOR 및 CONNECTOR 구성 매개변수

아래 표에는 **Netty** 네트워크 연결을 구성하는 데 사용되는 몇 가지 사용 가능한 매개변수가 자세히 설명되어 있습니다. 매개 변수 및 해당 값은 연결 문자열의 **URI**에 추가됩니다. 자세한 내용은 *네트워크 연결에서 어셉터 및 커넥터 구성* 을 참조하십시오. 각 테이블에는 허용자 또는 커넥터와 함께 사용할 수 있는지 또는 둘 다와 함께 사용할 수 있는지의 이름과 노트로 매개 변수가 나열되어 있습니다. 일부 매개변수(예: 허용자)만 사용할 수 있습니다.



참고

모든 **Netty** 매개변수는 **org.apache.activemq.artemis.core.remoting.impl.netty.TransportConstants** 클래스에 정의됩니다. *고객 포털에서* 소스 코드를 다운로드할 수 있습니다.

표 **A.1. Netty TCP** 매개변수

매개변수	와 함께 사용...	설명
batchDelay	둘 다	수락자 또는 커넥터에 패킷을 작성하기 전에 브로커는 최대 **batchDelay** 밀리초 동안 쓰기를 배치하도록 구성할 수 있습니다. 이렇게 하면 매우 작은 메시지의 전체 처리량이 증가할 수 있습니다. 따라서 메시지 전송에 대한 평균 대기 시간이 늘어남에 따라 발생합니다. 기본값은 **0** ms입니다.
connectionsAllowed	어셉터	수락자가 허용하는 연결 수를 제한합니다. 이 제한에 도달하면 로그에 DEBUG 수준 메시지가 표시되고 연결이 거부됩니다. 사용 중인 클라이언트 유형에 따라 연결이 거부될 때 발생하는 작업이 결정됩니다. 기본값은 **-1** 입니다. 즉, 수락자가 허용하는 연결 수에 제한이 없음을 의미합니다.
directDeliver	둘 다	메시지가 서버에 도착하고 대기 고객에게 전달되면 기본적으로 메시지가 도착한 것과 동일한 스레드에서 전송이 수행됩니다. 이로 인해 비교적 적은 메시지와 소비자 수가 적은 환경에서 대기 시간이 양호하지만, 특히 멀티 코어 시스템에서 전체 처리량과 확장성의 비용으로 대기 시간을 단축할 수 있습니다. 가장 짧은 대기 시간과 처리량을 줄이려면 **directDeliver** 에 기본값을 사용할 수 있습니다. 이는 **true** 입니다. 대기 시간에 약간의 추가 히트를 원하지만 가장 높은 처리량 세트를 **false** 로 설정 **하려는** 경우 .

매개변수	와 함께 사용...	설명
handshake-timeout	어셉터	권한이 없는 클라이언트가 많은 수의 연결을 열고 이를 열어 두지 않도록 합니다. 각 연결에 파일 처리가 필요하므로 다른 클라이언트에서 사용할 수 없는 리소스를 사용합니다. 이 시간 초과는 연결이 인증되지 않고 리소스를 사용할 수 있는 시간을 제한합니다. 연결이 인증되면 리소스 제한 설정을 사용하여 리소스 사용을 제한할 수 있습니다. 기본값은 **10** 초로 설정됩니다. 다른 정수 값으로 설정할 수 있습니다. 이 옵션을 0 또는 음수 정수로 설정하여 해제할 수 있습니다. 시간 초과 값을 편집한 후 브로커를 다시 시작해야 합니다.
localAddress	커넥터	클라이언트가 원격 주소에 연결할 때 사용할 로컬 주소를 지정합니다. 일반적으로 Application Server에서 사용되거나 Cryostat를 실행하여 아웃바운드 연결에 사용되는 주소를 제어합니다. local-address가 설정되지 않은 경우 커넥터는 사용 가능한 로컬 주소를 사용합니다.
localPort	커넥터	클라이언트가 원격 주소에 연결할 때 사용할 로컬 포트를 지정합니다. 일반적으로 Application Server에서 사용되거나 Cryostat를 실행하여 아웃바운드 연결에 사용되는 포트를 제어합니다. 기본값(0인)이 사용되는 경우 커넥터는 시스템이 임시 포트를 선택하도록 합니다. 유효한 포트는 0에서 65535입니다.
nioRemotingThreads	둘 다	NIO를 사용하도록 구성된 경우 브로커는 기본적으로 수신 패킷을 처리하기 위해 **Runtime.getRuntime().availableProcessors()** 에서 보고하는 코어 수(또는 하이퍼 스레드) 수의 3배와 동일한 스레드를 사용합니다. 이 값을 재정의하려면 이 매개변수를 지정하여 스레드 수를 설정할 수 있습니다. 이 매개변수의 기본값은 **-1** 입니다. 즉, **Runtime.getRuntime().availableProcessors()** * 3에서 파생된 값을 사용합니다.
tcpNoDelay	둘 다	이 값이 **true** 이면 나글의 알고리즘이 비활성화됩니다. Java(클라이언트) 소켓 옵션입니다. 기본값은 **true**입니다.
tcpReceiveBufferSize	둘 다	TCP 수신 버퍼의 크기를 바이트 단위로 결정합니다. 기본값은 **32768** 입니다.

매개변수	와 함께 사용...	설명
tcpSendBufferSize	둘 다	TCP 전송 버퍼의 크기를 바이트 단위로 결정합니다. 기본값은 **32768** 입니다. TCP 버퍼 크기는 네트워크의 대역폭 및 대기 시간에 따라 조정해야 합니다. 요약하면 TCP send/receive buffer size는 다음과 같이 계산되어야 합니다. buffer_size = bandwidth * RTT. 여기서 대역폭은 초당 바이트 단위이고 네트워크 라운드 트립 시간(RTT)은 초 단위입니다. RTT는 **ping** 유틸리티를 사용하여 쉽게 측정할 수 있습니다. 빠른 네트워크의 경우 기본값에서 버퍼 크기를 늘릴 수 있습니다.

표 A.2. Netty HTTP 매개변수

매개변수	와 함께 사용...	설명
httpClientIdleTime	어셉터	연결을 활성 상태로 유지하기 위해 빈 HTTP 요청을 보내기 전에 클라이언트를 유휴 상태로 유지할 수 있는 시간입니다.
httpClientIdleScanPeriod	어셉터	유휴 클라이언트를 스캔하는 빈도(밀리초)입니다.
httpEnabled	어셉터	더 이상 필요하지 않습니다. 단일 포트가 지원되면 브로커는 HTTP가 사용 중인지를 자동으로 감지하고 자체적으로 구성합니다.
httpRequiresSessionId	둘 다	**true** 인 경우 클라이언트는 첫 번째 호출 후 세션 ID를 수신합니다. HTTP 커넥터가 서블릿 수락자에 연결할 때 사용됩니다. 이 구성은 권장되지 않습니다.
httpResponseTime	어셉터	연결을 활성 상태로 유지하기 위해 빈 HTTP 응답을 보내기 전에 서버가 기다릴 수 있는 시간입니다.
httpServerScanPeriod	어셉터	응답이 필요한 클라이언트를 스캔하는 빈도(밀리초)입니다.

표 A.3. Netty TLS/SSL 매개변수

매개변수	와 함께 사용...	설명

매개변수	와 함께 사용...	설명
enabledCipherSuites	둘 다	SSL 통신에 사용되는 쉼표로 구분된 암호화 제품군 목록입니다. 클라이언트 애플리케이션에서 지원하는 가장 안전한 암호화 제품군을 지정합니다. 브로커와 클라이언트에 공통되는 암호화 제품군의 쉼표로 구분된 목록을 지정하거나 암호화 모음을 지정하지 않는 경우 브로커와 클라이언트는 사용할 암호화 제품군을 서로 협상합니다. 지정할 암호화 모음을 모르는 경우 먼저 디버그 모드에서 실행 중인 클라이언트와 broker-client 연결을 설정하여 브로커와 클라이언트 모두에 공통된 암호화 제품군을 확인할 수 있습니다. 그런 다음 브로커에서 **enabledCipherSuites** 를 구성합니다. 사용 가능한 암호화 제품군은 브로커 및 클라이언트에서 사용하는 TLS 프로토콜 버전에 따라 다릅니다. 브로커를 업그레이드한 후 기본 TLS 프로토콜 버전이 변경되면 브로커와 클라이언트가 공통 암호화 제품군을 사용할 수 있도록 이전 TLS 프로토콜 버전을 선택해야 할 수 있습니다. 자세한 내용은 **enabledProtocols** 를 참조하십시오.
enabledProtocols	둘 다	SSL 통신에 사용되는 TLS 프로토콜 버전의 쉼표로 구분된 목록입니다. TLS 프로토콜 버전을 지정하지 않으면 브로커는 JVM의 기본 버전을 사용합니다. 브로커가 JVM의 기본 TLS 프로토콜 버전을 사용하고 브로커를 업그레이드한 후 해당 버전이 변경되면 브로커 및 클라이언트에서 사용하는 TLS 프로토콜 버전이 호환되지 않을 수 있습니다. 이후 TLS 프로토콜 버전을 사용하는 것이 좋지만, **enabledProtocols** 에서 이전 버전을 지정하여 최신 TLS 프로토콜 버전을 지원하지 않는 클라이언트와 상호 운용할 수 있습니다.
forceSSLParameters	커넥터	이 커넥터에 대한 SSL 컨텍스트를 구성하기 위해 JVM 시스템 속성(**javax.net.ssl** 및 AMQ Broker 시스템 속성 모두 포함) 대신 커넥터의 매개변수로 설정된 SSL 설정을 제어합니다. 유효한 값은 **true** 또는 **false** 입니다. 기본값은 **false** 입니다.
keyStorePassword	둘 다	어셉터에 사용되는 경우 서버 측 키 저장소의 암호입니다. 커넥터에 사용하면 클라이언트 측 키 저장소의 암호입니다. 이는 양방향 SSL(즉, 상호 인증)을 사용하는 경우에만 커넥터와 관련이 있습니다. 이 값은 서버에서 구성할 수 있지만 클라이언트가 다운로드하여 사용합니다. 클라이언트가 서버에 설정된 것과 다른 암호를 사용해야 하는 경우 customary **javax.net.ssl.keyStorePassword** 시스템 속성 또는 ActiveMQ별 **org.apache.activemq.ssl.keyStorePassword** 시스템 속성을 사용하여 서버 측 설정을 재정의할 수 있습니다. ActiveMQ별 시스템 속성은 클라이언트의 다른 구성 요소가 이미 표준 Java 시스템 속성을 사용하고 있는 경우 유용합니다.

매개변수	와 함께 사용...	설명
keyStorePath	둘 다	어셉터에서 사용하는 경우 이 경로는 서버의 인증서를 보유하고 있는 서버의 SSL 키 저장소 경로입니다(권한에서 자체 서명 또는 서명). 커넥터에 사용되는 경우 클라이언트 인증서를 보유하는 클라이언트 측 SSL 키 저장소의 경로입니다. 이는 양방향 SSL(즉, 상호 인증)을 사용하는 경우에만 커넥터와 관련이 있습니다. 이 값은 서버에서 구성되지만 클라이언트가 다운로드하여 사용합니다. 클라이언트가 서버에 설정된 것과 다른 경로를 사용해야 하는 경우 customary **javax.net.ssl.keyStore** 시스템 속성 또는 ActiveMQ 별 **org.apache.activemq.ssl.keyStore** 시스템 속성을 사용하여 서버 측 설정을 덮어쓸 수 있습니다. ActiveMQ별 시스템 속성은 클라이언트의 다른 구성 요소가 이미 표준 Java 시스템 속성을 사용하고 있는 경우 유용합니다.
keyStoreAlias	둘 다	어셉터에 사용되는 경우 이 별칭은 키 저장소에서 연결할 때 클라이언트에 제공할 키의 별칭입니다. 커넥터에 사용되는 경우 클라이언트가 연결할 때 키 저장소의 키 별칭입니다. 이는 2방향 SSL, 즉 상호 인증을 사용하는 경우 커넥터에만 관련이 있습니다.
needClientAuth	어셉터	이 어셉터에 연결하는 클라이언트에는 양방향 SSL이 필요하다는 것을 알려줍니다. 유효한 값은 **true** 또는 **false**입니다. 기본값은 **false**입니다.
sslEnabled	둘 다	SSL을 활성화하려면 **true** 여야 합니다. 기본값은 **false**입니다.
sslAutoReload	어셉터	**true** 로 설정하면 브로커는 키 저장소 및 신뢰 저장소 경로와 같은 SSL 구성의 변경 사항을 확인하고 변경 사항이 감지되면 구성을 자동으로 다시 로드합니다. 기본값은 **false**입니다. 브로커가 변경 사항을 확인하는 간격은 **configuration-file-refresh-period** 요소의 값에 따라 결정됩니다. 자세한 내용은 구성 업데이트 다시 로드를 참조하십시오.
trustManagerFactoryPlugin	둘 다	**org.apache.activemq.artemis.api.core.TrustManagerFactoryPlugin** 을 구현하는 클래스의 이름을 정의합니다. **javax.net.ssl.TrustManagerFactory** 를 반환하는 단일 메서드가 있는 간단한 인터페이스입니다. **TrustManagerFactory** 는 기본 **javax.net.ssl.SSLContext** 가 초기화될 때 사용됩니다. 이를 통해 브로커 및 클라이언트에서 신뢰하는 사람 또는 내용을 세부적으로 사용자 지정할 수 있습니다. **trustManagerFactoryPlugin** 의 값은 신뢰 관리자(즉, **trustAll**,**trustStoreProvider**,**trustStorePath**,**trustStorePassword**, **crlPath**)에 적용되는 다른 모든 SSL 매개변수보다 우선합니다. 지정된 플러그인을 브로커의 Java 클래스 경로에 배치해야합니다. 기본적으로 클래스 경로의 ***일부이므로 <broker_instance_dir>*** **/lib** 디렉터리를 사용할 수 있습니다.

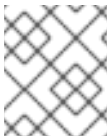
매개변수	와 함께 사용...	설명
trustStorePassword	둘 다	승인자에서 사용하는 경우 이는 서버 측 신뢰 저장소의 암호입니다. 이는 2방향 SSL(즉, 상호 인증)을 사용하는 경우에만 어셉터와 관련이 있습니다. 커넥터에 사용하면 클라이언트 측 신뢰 저장소의 암호입니다. 이 값은 서버에서 구성할 수 있지만 클라이언트가 다운로드하여 사용합니다. 클라이언트가 서버에 설정된 것과 다른 암호를 사용해야 하는 경우 customary **javax.net.ssl.trustStorePassword** 시스템 속성 또는 ActiveMQ별 **org.apache.activemq.ssl.trustStorePassword** 시스템 속성을 사용하여 서버 측 설정을 재정의할 수 있습니다. ActiveMQ별 시스템 속성은 클라이언트의 다른 구성 요소가 이미 표준 Java 시스템 속성을 사용하고 있는 경우 유용합니다.
sniHost	둘 다	acceptor에 사용되는 경우 **sniHost** 는 들어오는 SSL 연결의 **server_name** 확장을 일치시키는 데 사용되는 정규식입니다(이 확장에 대한 자세한 내용은 https://tools.ietf.org/html/rfc6066) 을 참조하십시오. 이름이 일치하지 않으면 수락자에 대한 연결이 거부됩니다. 이 경우 WARN 메시지가 기록됩니다. 들어오는 연결에 **server_name** 확장이 포함되지 않으면 연결이 허용됩니다. 커넥터에서 사용되는 경우 **sniHost** 값은 SSL 연결의 **server_name** 확장에 사용됩니다.
sslProvider	둘 다	**JDK** 와 **OPENSSL** 간에 SSL 공급자를 변경하는 데 사용됩니다. 기본값은 **JDK** 입니다. **OPENSSL** 로 설정하면 기본적으로 설치된 OpenSSL을 사용하도록 classpath에 **netty-tcnative** 를 추가할 수 있습니다. 이 옵션은 OpenSSL을 통해 지원되지만 JDK 공급자를 통해 지원되지 않는 특수 암호suite-elliptic 곡선 조합을 사용하려는 경우에 유용할 수 있습니다.
trustStorePath	둘 다	어셉터에서 사용하는 경우 이 경로는 서버가 신뢰하는 모든 클라이언트의 키를 보유하는 서버 측 SSL 키 저장소의 경로입니다. 이는 2방향 SSL(즉, 상호 인증)을 사용하는 경우에만 어셉터와 관련이 있습니다. 커넥터에 사용되는 경우 이는 클라이언트가 신뢰하는 모든 서버의 공개 키를 보유하는 클라이언트 측 SSL 키 저장소의 경로입니다. 이 값은 서버에서 구성할 수 있지만 클라이언트가 다운로드하여 사용합니다. 클라이언트에서 서버에 설정된 것과 다른 경로를 사용해야 하는 경우 사용자 지정 **javax.net.ssl.trustStore** 시스템 속성 또는 ActiveMQ별 **org.apache.activemq.ssl.trustStore** 시스템 속성을 사용하여 서버 측 설정을 재정의할 수 있습니다. ActiveMQ별 시스템 속성은 클라이언트의 다른 구성 요소가 이미 표준 Java 시스템 속성을 사용하고 있는 경우 유용합니다.

매개변수	와 함께 사용...	설명
useDefaultSslContext	커넥터	커넥터가 클라이언트(SSLContext.setDefault)를 통해 프로그래밍 방식으로 설정할 수 있는 "기본" SSL 컨텍스트(**SSLContext.getDefault()를 통해)**를 사용할 수 있습니다. 이 매개변수가 **true** 로 설정된 경우 **sslEnabled** 를 제외한 다른 모든 SSL 관련 매개변수는 무시됩니다. 유효한 값은 **true** 또는 **false**입니다. 기본값은 **false**입니다.
verifyHost	둘 다	커넥터에서 사용할 때 서버의 SSL 인증서의 CN 또는 주체 대체 이름 값은 해당 인증서가 일치하는지 확인하기 위해 연결된 호스트 이름과 비교됩니다. 이는 단방향 및 양방향 SSL 모두에 유용합니다. 수락자에서 사용되는 경우 연결 클라이언트의 SSL 인증서의 CN 또는 Subject Alternative Name 값은 해당 인증서가 일치하는지 확인하기 위해 호스트 이름과 비교됩니다. 이는 양방향 SSL에만 유용합니다. 유효한 값은 **true** 또는 **false**입니다. 커넥터의 기본값은 **true** 이고 어셉터의 경우 **false** 입니다.
wantClientAuth	어셉터	이 어셉터에 연결하는 클라이언트에는 양방향 SSL이 요청되지만 필수는 아님을 알려줍니다. 유효한 값은 **true** 또는 **false**입니다. 기본값은 **false**입니다. 속성 **needClientAuth** 가 **true** 로 설정된 경우 해당 속성이 우선 순위가 수행되고 **wantClientAuth** 는 무시됩니다.

부록 B. 주소 설정 구성

*아래 표에는 **address-setting** 의 모든 구성 요소가 나열되어 있습니다. 일부 요소는 **DEPRECATED**로 표시됩니다. 잠재적인 문제를 방지하려면 제안된 교체를 사용하십시오.*

*표 **B.1.** 주소 설정 요소*

이름	설명
address-full-policy	**max-size-bytes** 로 구성된 주소가 가득 차면 발생하는 작업을 결정합니다. 사용 가능한 정책은 다음과 같습니다. **PAGE**: 전체 주소로 전송되는 메시지는 디스크로 표시됩니다. **DROP**: 전체 주소로 전송되는 메시지는 자동으로 삭제됩니다. **FAIL**: 전체 주소로 전송된 메시지는 삭제되고 메시지 생산자가 예외를 수신합니다. **BLOCK**: 메시지 생산자는 시도 할 때 차단하고 추가 메시지를 보냅니다.  참고 BLOCK 정책은 AMQP, OpenWire 및 Core Protocol 프로토콜에서만 작동하므로 흐름 제어를 사용할 수 있습니다.
auto-create-addresses	클라이언트가 큐가 없는 주소에 매핑된 큐에서 메시지를 보내거나 사용하려고 할 때 주소를 자동으로 생성할지 여부입니다. 기본값은 **true**입니다.
auto-create-dead-letter-resources	브로커가 전달되지 않은 메시지를 수신하기 위해 dead letter address 및 queue를 자동으로 생성하는지 여부를 지정합니다. 기본값은 **false**입니다. 매개변수가 **true** 로 설정되면 브로커는 dead letter**address** 및 관련 dead letter 큐를 정의하는 <address> 요소를 자동으로 생성합니다. 자동으로 생성된 < **address** > 요소의 이름은 < **dead-letter-address>에 지정하는 name 값과 일치합니다**.
auto-create-jms-queues	dePRECATED: 대신 **auto-create-queues** 를 사용합니다. JMS 생산자 또는 소비자가 이러한 대기열을 사용하려고 할 때 이 브로커가 주소 설정에 해당하는 JMS 대기열을 자동으로 생성해야 하는지 여부를 결정합니다. 기본값은 **false**입니다.
auto-create-jms-topics	dePRECATED: 대신 **auto-create-queues** 를 사용합니다. JMS 생산자 또는 소비자가 이러한 대기열을 사용하려고 할 때 이 브로커가 주소 설정에 해당하는 JMS 주제를 자동으로 생성해야 하는지 여부를 결정합니다. 기본값은 **false**입니다.
auto-create-queues	클라이언트가 메시지를 보내거나 큐에서 메시지를 사용하려고 할 때 큐를 자동으로 생성할지 여부입니다. 기본값은 **true**입니다.

이름	설명
auto-delete-addresses	브로커에 더 이상 큐가 없는 경우 자동 생성 주소를 삭제할지 여부입니다. 기본값은 **true**입니다.
auto-delete-jms-queues	대신 **auto-delete-queues** 를 사용합니다. AMQ Broker가 소비자가 없고 메시지가 없는 경우 자동 생성된 JMS 대기열을 자동으로 삭제해야 하는지 여부를 결정합니다. 기본값은 **false**입니다.
auto-delete-jms-topics	대신 **auto-delete-queues** 를 사용합니다. AMQ Broker가 소비자가 없고 메시지가 없는 경우 자동 생성된 JMS 주제를 자동으로 삭제해야 하는지 여부를 결정합니다. 기본값은 **false**입니다.
auto-delete-queues	큐에 소비자가 없고 메시지가 없는 경우 자동 생성 대기열을 삭제할지 여부입니다. 기본값은 **true**입니다.
config-delete-addresses	구성 파일이 다시 로드되면 이 설정은 구성 파일에서 삭제된 주소(및 해당 대기열)를 처리하는 방법을 지정합니다. 다음 값을 지정할 수 있습니다. **OFF (기본값)** 구성 파일이 다시 로드되면 주소가 삭제되지 않습니다. **FORCE** 주소 및 큐는 구성 파일이 다시 로드될 때 삭제됩니다. 큐에 메시지가 있는 경우 해당 메시지도 제거됩니다.
config-delete-queues	구성 파일이 다시 로드되면 이 설정은 구성 파일에서 삭제된 대기열을 처리하는 방법을 지정합니다. 다음 값을 지정할 수 있습니다. **OFF (기본값)** 구성 파일이 다시 로드되면 큐가 삭제되지 않습니다. **FORCE** 구성 파일이 다시 로드되면 큐가 삭제됩니다. 큐에 메시지가 있는 경우 해당 메시지도 제거됩니다.
dead-letter-address	브로커가 dead message를 보내는 주소입니다.
dead-letter-queue-prefix	브로커가 자동으로 생성된 dead letter 큐의 이름에 적용되는 접두사입니다. 기본값은 **DLQ입니다.**
dead-letter-queue-suffix	브로커가 자동으로 생성된 dead letter 큐에 적용되는 접미사입니다. 기본값은 정의되지 않습니다(즉, 브로커는 접미사를 적용하지 않음).
default-address-routing-type	자동 생성된 주소에 사용되는 routing-type입니다. 기본값은 **MULTICAST** 입니다.
default-max-consumers	한 번에 이 큐에 허용된 최대 소비자 수입니다. 기본값은 **200** 입니다.

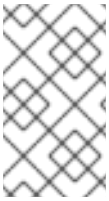
이름	설명
default-purge-on-no-consumers	소비자가 없으면 대기열의 콘텐츠를 제거할지 여부입니다. 기본값은 **false**입니다.
default-queue-routing-type	자동 생성된 큐에 사용되는 routing-type입니다. 기본값은 **MULTICAST** 입니다.
enable-metrics	Prometheus 플러그인과 같이 구성된 메트릭 플러그인이 일치하는 주소 또는 주소 *세트에* 대한 지표를 수집하는지 여부를 지정합니다. 기본값은**true**입니다.
expiry-address	만료된 메시지를 받을 주소입니다.
expiry-delay	기본 만료 시간을 사용하여 메시지에 사용할 만료 시간을 밀리초 단위로 정의합니다. 기본값은 **-1** 이며 이는 만료 시간이 없음을 의미합니다.
last-value-queue	큐에서 마지막 값만 사용하는지 여부입니다. 기본값은 **false**입니다.
management-browse-page-size	관리 리소스에서 검색할 수 있는 메시지 수입니다. 기본값은 **200** 입니다.
max-delivery-attempts	dead letter address로 보내기 전에 메시지를 전달하려고 시도하는 횟수입니다. 기본값은 **10**입니다.
max-redelivery-delay	redelivery-delay의 최대값(밀리초)입니다.
max-size-bytes	이 주소의 최대 메모리 크기(바이트)입니다. **address-full-policy** 가 **PAGING**,**BLOCK** 또는 **FAIL** 인 경우 이 값은 "K", "Mb", "GB"와 같은 바이트 표기법으로 지정됩니다. 기본값은 **-1** 이며, 이는 무한 바이트를 나타냅니다. 이 매개변수는 특정 주소 공간이 사용하는 메모리 양을 제한하여 브로커 메모리를 보호하는 데 사용됩니다. 이 설정은 현재 브로커 주소 공간에 저장된 클라이언트에서 보낸 총 바이트 양을 나타내지 않습니다. 브로커 메모리 사용률을 추정한 것입니다. 이 값은 런타임 조건 및 특정 워크로드에 따라 다를 수 있습니다. 주소 공간당 여유가 있을 수 있는 최대 메모리 양을 할당하는 것이 좋습니다. 일반적인 워크로드에서는 브로커에 메모리의 뛰어난 메시지의 페이로드 크기 중 약 150%에서 200%가 필요합니다.
max-size-bytes-reject-threshold	**address-full-policy** 가 **BLOCK** 인 경우 사용됩니다. 브로커가 메시지를 거부하기 전에 주소가 도달할 수 있는 최대 크기(바이트)입니다. AMQP 프로토콜에 대해서만 **max-size-bytes** 와 함께 작동합니다. 기본값은**-1** 이며 이는 제한이 없음을 의미합니다.
message-counter-history-day-limit	이 주소에 대한 메시지 카운터 기록을 유지하는 일 수입니다. 기본값은 **0**입니다.
page-size-bytes	페이징 크기(바이트)입니다. 또한 **K**,**Mb** 및 **GB** 와 같은 바이트 표기법을 지원합니다. 기본값은 **10485760** 바이트, 거의 10.5MB입니다.

이름	설명
redelivery-delay	취소된 메시지를 다시 전달할 때까지 대기하는 시간(밀리초)입니다. 기본값은 **0**입니다.
redelivery-delay-multiplier	redelivery-delay 매개변수에 적용할 수 있는 승수입니다. 기본값은 **1.0** 입니다.
redistribution-delay	메시지를 다시 배포하기 전에 마지막 소비자가 큐에서 종료된 후 밀리초 단위로 대기하는 시간을 정의합니다.Defines how to wait in milliseconds after the last consumer is closed on a queue before redistributing any messages. 기본값은 **-1** 입니다.
send-to-dla-on-no-route	**true** 로 설정하면 큐로 라우팅할 수 없는 경우 구성된 dead letter 주소로 메시지가 전송됩니다. 기본값은 **false**입니다.
slow-consumer-check-period	느린 소비자를 위해 초 단위로 확인하는 빈도입니다. 기본값은 **5** 입니다.
slow-consumer-policy	느린 소비자가 식별될 때 발생하는 상황을 결정합니다. 유효한 옵션은 **KILL** 또는 **NOTIFY** 입니다. **KILL** 은 소비자의 연결을 종료하여 동일한 연결을 사용하여 모든 클라이언트 스레드에 영향을 미칩니다. **NOTIFY** 는 **CONSUMER_SLOW** 관리 알림을 클라이언트에 보냅니다. 기본값은 **NOTIFY** 입니다.
slow-consumer-threshold	소비자 이전에 허용된 최소 메시지 사용률은 느린 것으로 간주됩니다. 초당 메시지 수로 측정됩니다. 기본값은 바인딩되지 않은 **-1** 입니다.

부록 C. 클러스터 연결 구성

아래 표에는 클러스터 연결 의 모든 구성 요소가 나열되어 있습니다.

표 C.1. 클러스터 연결 구성 요소

이름	설명
address	각 클러스터 연결은 **address** 필드에 지정된 값과 일치하는 주소에만 적용됩니다. 주소를 지정하지 않으면 모든 주소가 부하 분산됩니다. **address** 필드는 쉼표로 구분된 주소 목록도 지원합니다. address가 일치하지 않도록 exclude 구문 **!** 를 사용합니다. 다음은 몇 가지 예제 주소입니다. **jms.eu** **jms.eu** 로 시작하는 모든 주소와 일치합니다. **!jms.eu** **jms.eu**로 시작하는 사용자를 제외한 모든 주소와 일치 **jms.eu.uk,jms.eu.de** **jms.eu.uk** 또는 **jms.eu.de**로 시작하는 모든 주소와 일치합니다. **jms.eu,!jms.eu.uk** jms.eu로 시작하는 모든 주소와 일치하지만 **jms.eu.uk**로 시작하는 사용자는 일치하지 않습니다.  참고 두 개 이상의 클러스터 연결 간에 동일한 메시지를 배포할 수 있으므로 겹치는 주소(예: "europe" 및 "europe.news")가 있는 클러스터 연결이 여러 개 없어야 합니다. 이로 인해 중복 제공이 발생할 수 있습니다.
call-failover-timeout	장애 조치(failover) 시도 중에 호출이 수행될 때 사용합니다.Use when a call is made during a failover attempt. 기본값은 **-1** 또는 시간 초과입니다.
call-timeout	패킷이 클러스터 연결을 통해 전송되고 차단 호출인 경우 **call-timeout** 은 예외를 throw하기 전에 브로커가 응답을 대기(밀리초)하는 기간을 결정합니다. 기본값은 **30000** 입니다.
check-period	클러스터 연결이 다른 브로커에서 ping을 수신하지 못한지 확인하는 간격(밀리초)입니다. 기본값은 **30000** 입니다.
confirmation-window-size	연결된 브로커에서 확인을 보내는 데 사용되는 창의 크기(바이트)입니다. 브로커가 **confirmation-window-size** 바이트를 수신하면 클라이언트에 알립니다. 기본값은 **1048576** 입니다. 값이 **-1** 이면 창이 없음을 의미합니다.
Connector-ref	올바른 클러스터 토폴로지를 갖도록 클러스터의 다른 브로커로 전송할 **커넥터** 를 식별합니다. 이 매개변수는 필수입니다.

이름	설명
connection-ttl	클러스터의 특정 브로커에서 메시지 수신을 중지하면 클러스터 연결이 활성 상태로 유지되는 시간을 결정합니다. 기본값은 **60000** 입니다.
discovery-group-ref	클러스터의 다른 브로커와 통신하는 데 사용할 **검색 그룹을** 가리킵니다. 이 요소에는 이전에 구성된 검색 그룹 의 **name** 속성과 일치해야 하는 **discovery-group- name** 속성이 포함되어야 합니다.
initial-connect-attempts	시스템이 처음에 클러스터의 브로커를 연결하려고 시도하는 횟수를 설정합니다. max-retry가 달성되면 이 브로커는 영구적으로 다운된 것으로 간주되며 시스템은 이 브로커로 메시지를 라우팅하지 않습니다. 기본값은 **-1** 이며 이는 무한 재시도를 의미합니다.
max-hops	체인에서 다른 브로커와 함께 간접적으로 연결할 수 있는 브로커에 메시지를 로드 밸런싱하도록 브로커를 구성합니다. 이를 통해 메시지 부하 분산을 계속 제공하는 동안 더 복잡한 토폴로지를 사용할 수 있습니다. 기본값은 **1** 입니다. 즉, 이 브로커에 직접 연결된 다른 브로커에게만 메시지가 배포됩니다. 이 매개변수는 선택 사항입니다.
max-retry-interval	재시도의 최대 지연 시간(밀리초)입니다. 기본값은 **2000** 입니다.
message-load-balancing	클러스터의 다른 브로커 간에 메시지를 배포할지 여부 및 방법을 결정합니다. **message-load-balancing** 요소를 포함하여 로드 밸런싱을 활성화합니다. 기본값은 **ON_DEMAND** 입니다. 또한 가치를 제공할 수 있습니다. 유효한 값은 다음과 같습니다. **OFF** 로드 밸런싱을 비활성화합니다. **STRICT** 부하 분산을 활성화하고 큐에 활성 소비자 또는 일치하는 선택기가 있는지 여부에 관계없이 일치하는 큐가 있는 모든 브로커에 메시지를 전달합니다. **ON_DEMAND** 로드 밸런싱을 활성화하고 메시지가 일치하는 선택기가 있는 브로커로만 전달되도록 합니다. **OFF_WITH_REDISTRIBUTION** 부하 분산을 비활성화하지만 메시지가 적절한 로컬 소비자를 사용할 수 없는 경우 일치하는 선택기가 있는 브로커로만 전달되도록 합니다.
min-large-message-size	메시지 크기가 **min-large-message-size** 보다 크면 네트워크를 다른 클러스터 멤버로 전송할 때 여러 세그먼트로 분할됩니다. 기본값은 **102400** 입니다.
notification-attempts	클러스터에 연결할 때 클러스터 연결이 자체 브로드캐스트해야 하는 횟수를 설정합니다. 기본값은 **2** 입니다.
notification-interval	클러스터에 연결할 때 클러스터 연결이 자체적으로 브로드캐스트되는 빈도를 설정합니다. 기본값은 **1000** 입니다.

이름	설명
producer-window-size	생산자 흐름의 크기(바이트)는 클러스터 연결을 제어합니다. 기본적으로 비활성화되어 있지만 클러스터에서 실제 메시지를 사용하는 경우 값을 설정할 수 있습니다. 값이 **-1** 이면 창이 없음을 의미합니다.
재연결-attempts	시스템이 클러스터의 브로커에 다시 연결하려고 시도하는 횟수를 설정합니다. max-retry가 달성되면 이 브로커는 영구적으로 다운된 것으로 간주되며 시스템은 이 브로커로 메시지 라우팅을 중지합니다. 기본값은 **-1** 이며 이는 무한 재시도를 의미합니다.
retry-interval	재시도 시도 사이의 간격(밀리초)을 결정합니다. 클러스터 연결이 생성되고 대상 브로커가 시작되지 않았거나 부팅 중인 경우 다른 브로커의 클러스터 연결이 백업될 때까지 대상에 대한 연결을 다시 시도합니다. 이 매개변수는 선택 사항입니다. 기본값은 500밀리초입니다.
retry-interval-multiplier	다시 연결할 때마다 **재시도 간격을** 늘리는 데 사용되는 multiplier입니다. 기본값은 1입니다.
use-duplicate-detection	클러스터 연결은 브릿지를 사용하여 브로커를 연결하고 브리지는 전달된 각 메시지에 중복 ID 속성을 추가하도록 구성할 수 있습니다. 브리지의 대상 브로커가 충돌하고 복구되면 소스 브로커에서 메시지가 다시 표시될 수 있습니다. **use-duplicate-detection** 을 **true** 로 설정하면 중복 메시지가 필터링되고 대상 브로커의 수신 시 무시됩니다. 기본값은 **true**입니다.

부록 D. 명령줄 툴

AMQ Broker에는 메시징 저널을 관리할 수 있도록 **CLI(**명령줄 인터페이스**)** 툴 세트가 포함되어 있습니다. 아래 표에는 각 도구의 이름과 설명이 나열되어 있습니다.

표 D.1. 명령줄 툴

툴	설명
EXP	특수하고 독립적인 XML 형식을 사용하여 메시지 데이터를 내보냅니다.
Imp	**exp** 에서 제공하는 출력을 사용하여 실행 중인 브로커로 저널을 가져옵니다.
data	저널 레코드에 대한 보고서를 출력하고 데이터를 압축합니다.
encode	문자열로 인코딩된 저널의 내부 형식을 표시합니다.
디코딩	인코딩에서 내부 저널 형식을 가져옵니다.

각 툴에 사용할 수 있는 전체 명령 목록은 도움말 매개 변수 뒤에 툴 이름을 사용합니다. 아래 예제에서 **CLI** 출력에는 사용자가 명령 **./artemis** 도움말 데이터를 입력한 후 데이터 툴에서 사용할 수 있는 모든 명령이 나열됩니다.

```
$ ./artemis help data

NAME
        artemis data - data tools group
        (print|imp|exp|encode|decode|compact) (example ./artemis data print)

SYNOPSIS
        artemis data
        artemis data compact [--broker <brokerConfig>] [--verbose]
                [--paging <paging>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data decode [--broker <brokerConfig>] [--suffix <suffix>]
                [--verbose] [--paging <paging>] [--prefix <prefix>] [--file-size <size>]
                [--directory <directory>] --input <input> [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data encode [--directory <directory>] [--broker <brokerConfig>]
                [--suffix <suffix>] [--verbose] [--paging <paging>] [--prefix <prefix>]
                [--file-size <size>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data exp [--broker <brokerConfig>] [--verbose]
                [--paging <paging>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data imp [--host <host>] [--verbose] [--port <port>]
                [--password <password>] [--transaction] --input <input> [--user <user>]
        artemis data print [--broker <brokerConfig>] [--verbose]
```

```
        [--paging <paging>] [--journal <journal>]
        [--large-messages <largeMessges>] [--bindings <binding>]

COMMANDS
    With no arguments, Display help information

    print
        Print data records information (WARNING: don't use while a
        production server is running)

    ...
```

*툴의 도움말을 사용하여 각 툴의 명령을 실행하는 방법에 대한 자세한 내용을 확인할 수 있습니다. 예를 들어, **CLI**는 사용자가 **./artemis** 도움말 데이터 인쇄를 입력한 후 **data print** 명령에 대한 자세한 정보를 나열합니다.*

```
$ ./artemis help data print

NAME
    artemis data print - Print data records information (WARNING: don't use
    while a production server is running)

SYNOPSIS
    artemis data print [--bindings <binding>] [--journal <journal>]
        [--paging <paging>]

OPTIONS
    --bindings <binding>
        The folder used for bindings (default ../data/bindings)

    --journal <journal>
        The folder used for messages journal (default ../data/journal)

    --paging <paging>
        The folder used for paging (default ../data/paging)
```

부록 E. 메시징 저널 구성 CRYOSTAT

*아래 표에는 **AMQ Broker** 메시징 저널과 관련된 모든 구성 요소가 나열되어 있습니다.*

*표 **E.1.** 저널 구성 요소*

이름	설명
journal-directory	메시지 저널이 있는 디렉터리입니다. 기본값은 < ***broker_instance_dir>*/data/journal** 입니다. 최상의 성능을 위해 디스크 헤드 이동을 최소화하려면 저널이 자체 물리 볼륨에 있어야 합니다. 저널이 다른 파일(예: 바인딩 저널, 데이터베이스 또는 트랜잭션 조정기)을 작성할 수 있는 다른 프로세스와 공유되는 볼륨에 있는 경우 디스크 헤드는 이러한 파일 간에 빠르게 이동하므로 성능을 크게 줄일 수 있습니다. SAN을 사용하는 경우 각 저널 인스턴스에 자체 LUN(논리적 단위)이 지정되어야 합니다.
create-journal-dir	**true** 로 설정하면 저널 디렉터리가 아직 존재하지 않는 경우**저널 디렉터리에** 지정된 위치에 자동으로 생성됩니다. 기본값은**true**입니다.
journal-type	유효한 값은 **NIO** 또는 **ASYNCIO** 입니다. **NIO** 로 설정하면 브로커는 Java NIO 인터페이스를 itsjournal에 사용합니다. **ASYNCIO** 로 설정하고 브로커는 Linux 비동기 IO 저널을 사용합니다. **ASYNCIO** 를 선택했지만 Linux를 실행 중이 아니거나 libaio가 설치되지 않은 경우 브로커는 이를 감지하고 **NIO** 사용을 자동으로 대체합니다.
journal-sync-transactional	**true** 로 설정하면 브로커가 모든 트랜잭션 데이터를 트랜잭션 경계(즉, 커밋, 준비 및 롤백)의 디스크에 플러시합니다. 기본값은 **true**입니다.
journal-sync-non-transactional	**true** 로 설정하면 브로커가 비-데이터 메시지 데이터(전송 및 승인)를 매번 디스크에 플러시합니다. 기본값은 **true**입니다.
journal-file-size	각 저널 파일의 크기(바이트)입니다. 기본값은 **10485760** 바이트(10MiB)입니다.
journal-min-files	시작할 때 브로커가 사전 생성하는 최소 파일 수입니다. 파일은 기존 메시지 데이터가 없는 경우에만 미리 생성됩니다. 대기 중인 상태가 예상되는 데이터의 양에 따라 예상되는 총 데이터 양과 일치하도록 이 파일 수를 조정해야 합니다.
journal-pool-files	시스템은 필요에 따라 파일을 많이 생성하지만 파일을 회수하면 **journal-pool-files** 로 축소됩니다. 기본값은 **-1** 입니다. 즉, 생성된 저널에서 파일을 삭제하지 않습니다. 그러나 시스템이 무한히 증가할 수 있는 대상에 대한 페이징을 사용해야 하므로 시스템이 무한히 증가할 수 없습니다.

이름	설명
journal-max-io	언제든지 IO 큐에 있을 수 있는 최대 쓰기 요청 수를 제어합니다. 큐가 가득 차면 공간이 확보될 때까지 쓰기가 차단됩니다. NIO를 사용할 때 이 값은 항상 **1** 이어야 합니다. AIO를 사용하는 경우 기본값은 **500** 입니다. 총 최대 AIO는 OS 수준에서 설정된 값보다 높을 수 없습니다 (**/proc/sys/fs/aio-max-nr**).
journal-buffer-timeout	버퍼가 플러시되는 시점의 시간 제한을 제어합니다. AIO는 일반적으로 NIO보다 높은 플러시 속도로 견딜 수 있으므로 시스템은 NIO 및 AIO 둘 다에 대해 다른 기본값을 유지합니다. NIO의 기본값은 초당 **3333333** 나노초 또는 초당 300회, AIO의 기본값은 **50000** 나노초 또는 초당 2000회입니다.  참고 기본값은 처리량과 대기 시간 간에 적절한 균형을 제공하기 위해 선택되므로 시간 초과 값을 늘리면 대기 시간이 지남에 따라 시스템 처리량을 높일 수 있습니다.
journal-buffer-size	AIO의 시간 초과 버퍼 크기입니다. 기본값은 **490KiB** 입니다.
journal-compact-min-files	브로커가 저널을 압축하기 전에 필요한 최소 파일 수입니다. 압축 알고리즘은 **journal-compact-min-files** 이상이 있을 때까지 시작되지 않습니다. 기본값은 **10** 입니다.  참고 값을 **0** 으로 설정하면 압축이 비활성화되고 저널이 무기한 증가할 수 있으므로 위험할 수 있습니다.
journal-compact-percentage	압축을 시작하는 임계값입니다. 저널 데이터는 **journal-compact-percentage** 보다 적은 경우 라이브 데이터로 압축됩니다. 또한 저널에 **journal-compact-min-files** 데이터 파일이 있을 때까지 압축이 시작되지 않습니다. 기본값은 **30** 입니다.

부록 F. 추가 복제 고가용성 구성입니다.

다음 표에는 복제 고가용성 구성 섹션에 설명되지 않은 추가 **ha-policy** 구성 요소가 나열되어 있습니다. 이러한 요소에는 대부분의 일반적인 사용 사례에 충분한 기본 설정이 있습니다.

표 F.1. 복제 고가용성을 위한 추가 구성 요소

이름	사용됨	설명
check-for-live-server	임베디드 브로커 조정	마스터 브로커로 구성된 브로커에만 적용됩니다. 시작 시 원래 마스터 브로커가 자체 서버 ID를 사용하여 다른 라이브 브로커에 대해 클러스터를 점검하는지 여부를 지정합니다. 원래 마스터 브로커로 돌아가 두 브로커가 동시에 활성화되는 "스플릿 브레인" 상황을 피하려면 **true** 로 설정합니다. 이 속성의 기본값은 **false** 입니다.
cluster-name	임베디드 브로커 및 Zoo Cryostat 조정	복제에 사용할 클러스터 구성의 이름입니다. 이 설정은 여러 클러스터 연결을 구성하는 경우에만 필요합니다. 구성된 경우 이 이름의 클러스터 구성이 클러스터에 연결할 때 사용됩니다. 설정되지 않은 경우 구성에 정의된 첫 번째 클러스터 연결이 사용됩니다.
initial-replication-sync-timeout	임베디드 브로커 및 Zoo Cryostat 조정	복제 브로커가 초기 복제 프로세스가 완료되면 복제본이 필요한 모든 데이터를 수신했음을 확인할 때까지 기다립니다. 이 속성의 기본값은 Cryostat **밀리초** 입니다. 참고: 이 기간 동안 다른 저널 관련 작업이 차단됩니다.
max-saved-replicated-journals-size	임베디드 브로커 및 Zoo Cryostat 조정	백업 브로커에만 적용됩니다. 백업 브로커가 유지하는 백업 저널 파일 수를 지정합니다. 이 값에 도달하면 브로커는 가장 오래된 저널 파일을 삭제하여 각 새 백업 저널 파일에 대한 공간을 만듭니다. 이 속성의 기본값은 **2** 입니다.

부록 G. 브로커 속성

다음은 XML 구성을 사용하는 대신 내부 java 구성 빈에 직접 적용할 수 있는 AMQ Broker 속성 목록입니다.

criticalAnalyzerCheckPeriod

유형: long
Default: 0
XML 이름: critical-analyzer-check-period
설명: 기본값은 critical-analyzer-timeout의 절반이며 런타임 시 계산됩니다.

pageMaxConcurrentIO

유형: int
Default: 5
XML name: page-max-concurrent-io
설명: 페이징 중에 허용되는 최대 동시 읽기 수입니다.

messageCounterSamplePeriod

유형: long
Default: 10000
XML 이름: message-counter-sample-period
설명: 메시지 카운터에 사용할 샘플 기간(ms)입니다.

networkCheckNIC

유형: 문자열
기본값:
XML 이름: network-check-nic
설명: 주소를 확인하는 데 사용할 네트워크 인터페이스 카드 이름입니다.

globalMaxSize

type: long
default: -1
XML name: global-max-size
description: 모든 주소가 생성되는 메시지에 대해 구성된 전체 정책에 입력되기 전에 크기(바이트)를 입력합니다. "K", "Mb", "MiB", "GB" 등과 같은 바이트 표기법을 지원합니다.

journalFileSize

유형: int
Default: 10485760
XML 이름: journal-file-size
설명: 각 저널 파일의 크기(바이트)입니다. 바이트 표기법을 지원합니다(예: "K", "Mb", "MiB", "GB").

configurationFileRefreshPeriod

유형: long
Default: 5000
XML name: configuration-file-refresh-period
description: 구성 파일의 수정을 확인하는 빈도(ms)를 입력합니다.

diskScanPeriod

유형: int
Default: 5000
XML 이름: disk-scan-period
설명: 디스크의 전체 디스크를 스캔하는 빈도(밀리초)입니다.

journalRetentionDirectory

유형: 문자열
기본값:

XML 이름: journal-retention-directory
설명: journal-retention 메시지 및 보존 구성을 저장하는 디렉터리입니다.

networkCheckPeriod

유형: long
default: 10000
XML 이름: network-check-period
설명: 네트워크가 작동 중인지 확인할 빈도(밀리초)입니다.

journalBufferSize_AIO

유형: int
default: 501760
XML 이름: journal-buffer-size
설명: 저널의 내부 버퍼의 크기(바이트)입니다. 바이트 표기법(예: "K", "Mb", "MiB", "GB")을 지원합니다.

networkCheckURLList

유형: 문자열
기본값:
XML 이름: network-check-URL-list
설명: 브로커를 유지해야 하는지 확인하는 데 사용할 쉼표로 구분된 URL 목록입니다.

networkCheckTimeout

type: int
Default: 1000
XML name: network-check-timeout
description: ping에 사용할 시간 초과(밀리초)입니다.

pageSyncTimeout

유형: int
Default:
XML name: page-sync-timeout
설명: 페이지를 동기화하는 데 사용되는 시간 초과(나노초)입니다. 정확한 기본값은 저널이 ASYNCIO 또는 NIO인지 여부에 따라 달라집니다.

journalPoolFiles

type: int
default: -1
XML name: journal-pool-files
description: 사전 생성할 저널 파일의 수입니다.

criticalAnalyzer

유형: 부울
true
XML 이름: critical-analyzer
설명: 중요한 경로에 대한 응답 시간을 분석하고 브로커 로그, 종료 또는 중지를 결정해야 합니다.

readWholePage

type: boolean
Default: false
XML name: read-whole-page
description: 페이지 캐시가 제거된 후 메시지를 가져오는 동안 전체 페이지를 읽을지 여부를 지정합니다.

maxDiskUsage

유형: int
Default: 90
XML name: max-disk-usage
description: 시스템 블록 또는 실패 전 디스크 사용량의 최대 백분율입니다.

globalMaxMessages

type: long
default: -1
XML name: global-max-messages
Description: *모든 주소가 구성된 주소 전체 정책에 입력되기 전의 메시지 수입니다.* ***global-max-size****와 함께 작동하며, 두 제한에 도달하면 구성된 주소 전체 정책이 실행됩니다.*

internalNamingPrefix

유형: ***문자열***
기본값:
XML *이름:* ***internal-naming-prefix***
설명: ***Artemis****는 내부 큐와 주소를 사용하여 특정 동작을 구현합니다. 이러한 대기열과 주소는 기본적으로* ***"$.activemq.internal"*** *접두사를 추가하여 이름 지정이 사용자 이름****pacing****과 충돌하지 않도록 합니다. 이 값을 유효한* ***Artemis*** *주소로 설정하여 재정의할 수 있습니다.*

journalFileOpenTimeout

type: int
Default: 5
XML name: journal-file-open-timeout
Description: *시간 초과 및 실패 전에 새 저널 파일을 열 때 대기하는 시간(초)입니다.*

journalCompactPercentage

유형: ***int***
default: 30
XML *이름:* ***journal-compact-percentage***
설명: 저널을 압축하는 라이브 데이터의 백분율입니다.

createBindingsDir

type: boolean
default: true
XML name: create-bindings-dir
description: true 값은 서버에서 시작 시 바인딩 디렉터리를 생성합니다.

suppressSessionNotifications

유형: 부울
기본값: false
XML 이름: suppress-session-notifications
설명: SESSION_CREATED 및 SESSION_CLOSED 알림을 억제할지 여부입니다. 알림 오버헤드를 줄이려면 true 로 설정합니다. 그러나 MQTT 클라이언트의 클러스터에서 고유한 클라이언트 ID 사용률을 적용하는 데 필요합니다.

journalBufferTimeout_AIO

type: int
Default:
XML name: journal-buffer-timeout
description: The timeout,in nanoseconds, used to flush internal buffers on the journal. 정확한 기본값은 저널이 ASYNCIO 또는 NIO인지 여부에 따라 달라집니다.

journalType

type: JournalType
Default: ASYNCIO
XML name: journal-type
description: 사용할 저널 유형입니다.

name

유형: 문자열
기본값:
XML 이름: 이름
설명: 노드 이름. 설정된 경우 토폴로지 알림에 사용됩니다.

networkCheckPingCommand

유형: 문자열
기본값:
XML 이름: network-check-ping-command
설명: IPV4 주소를 ping하는 데 사용되는 ping 명령입니다.

temporaryQueueNamespace

유형: 문자열
기본값:
XML 이름: temporary-queue-namespace
설명: 임시 큐의 주소 설정을 찾는 데 사용할 네임스페이스입니다.

pagingDirectory

유형: 문자열
기본값: data/paging
XML name: paging-directory
description: The directory in which to store paged messages.

journalDirectory

유형: 문자열
기본값: data/journal
XML 이름: journal-directory
설명: 저널 파일을 저장하는 디렉터리입니다.

journalBufferSize_NIO

유형: int
default: 501760
XML 이름: journal-buffer-size

설명: 저널의 내부 버퍼의 크기(바이트)입니다. 바이트 표기법을 지원합니다(예: "K", "Mb", "MiB", "GB").

journalDeviceBlockSize

유형: Integer
Default:
XML name: journal-device-block-size
설명: 장치에서 사용하는 크기(바이트)입니다. 일반적으로 fstat/st_blksize로 변환됩니다. 이는 st_blksize로 반환된 값을 바이패스하는 방법입니다.

nodeManagerLockDirectory

유형: 문자열
기본값:
XML 이름: node-manager-lock-directory
설명: 노드 관리자 잠금 파일을 저장할 디렉터리입니다.

messageCounterMaxDayHistory

type: int
Default: 10
XML name: message-counter-max-day-history
설명: 메시지 카운터 기록을 유지할 일 수입니다.

largeMessagesDirectory

유형: 문자열
기본값: data/largemessages
XML 이름: large-messages-directory
설명: 대용량 메시지를 저장할 디렉터리입니다.

NetworkCheckPing6Command

유형: 문자열
기본값:
XML 이름: network-check-ping6-command
설명: IPV6 주소를 ping하는 데 사용되는 ping 명령입니다.

memoryWarningThreshold

유형: int
default: 25
XML 이름: memory-warning-threshold
설명: 경고가 생성되는 사용 가능한 메모리의 백분율입니다.

mqttSessionScanInterval

유형: long
default: 5000
XML 이름: mqtt-session-scan-interval
설명: 만료된 MQTT 세션을 스캔할 빈도(밀리초)입니다.

journalMaxAtticFiles

유형: int
기본값:
XML 이름: journal-max-attic-files
설명:

journalSyncTransactional

type: boolean
default: true
XML name: journal-sync-ovnal
Description: true 로 설정된 경우 클라이언트에 응답을 반환하기 전에 트랜잭션 데이터가 저널에 동기화 될 때까지 기다립니다.

logJournalWriteRate

유형: 부울
default: false
XML 이름: log-journal-write-rate
설명: 저널 쓰기와 관련된 메시지를 기록할지 여부를 지정합니다.

journalMaxIO_AIO

type: int
Default:
XML name: journal-max-io
description: 한 번에 AIO 큐에 있을 수 있는 최대 쓰기 요청 수입니다. 기본값은 500 AIO 및 NIO의 경우 1 입니다.

messageExpiryScanPeriod

유형: long
default: 30000
XML 이름: message-expiry-scan-period
설명: 만료된 메시지를 스캔하는 빈도(밀리초)입니다.

criticalAnalyzerTimeout

유형: long
default: 120000
XML 이름: critical-analyzer-timeout
설명: 중요한 경로에서 시간 초과를 분석하는 데 사용되는 기본 시간 초과입니다.

messageCounterEnabled

type: boolean
Default: false

XML name: message-counter-enabled
description: true 값은 메시지 카운터가 활성화됨을 나타냅니다.

journalCompactMinFiles

유형: int
Default: 10
XML name: journal-compact-min-files
설명: 브로커가 압축 파일로 시작하기 전에 최소 데이터 파일 수입니다.

createJournalDir

type: boolean
default: true
XML name: create-journal-dir
Description: true 값은 journal 디렉터리가 생성됨을 의미합니다.

addressQueueScanPeriod

유형: long
Default: 30000
XML 이름: address-queue-scan-period
설명: 삭제해야 하는 주소와 큐를 스캔하는 빈도(밀리초)입니다.

memoryMeasureInterval

type: long
default: -1
XML name: memory-measure-interval
description: The frequency, in milliseconds, to sample JVM memory. 값 -1 은 메모리 샘플링을 비활성화합니다.

journalSyncNonTransactional

유형: 부울
default: true
XML 이름: journal-sync-non-non- Cryostatal
설명: true 인 경우 클라이언트에 응답을 반환하기 전에 비 트랜잭션 데이터가 저널에 동기화될 때까지 기다립니다.

connectionTtlCheckInterval

유형: long
Default: 2000
XML name: connection-ttl-check-interval
설명: ttl 위반에 대한 연결을 확인하는 빈도(밀리초)입니다.

rejectEmptyValidatedUser

type: boolean
default: false
XML name: reject-empty-validated-user
description: true 에서는 서버에서 검증된 사용자가 없는 메시지를 허용하지 않습니다. JMS에서 이 값은 JMSXUserID 입니다.

journalMaxIO_NIO

type: int
Default:
XML name: journal-max-io
description: 한 번에 AIO 큐에 있을 수 있는 최대 쓰기 요청 수입니다. 기본값은 500 AIO 및 NIO의 경우 1 입니다. 현재 브로커 속성은 정수 및 측정값을 바이트 단위로만 지원합니다.

transactionTimeoutScanPeriod

유형: long
default: 1000
XML 이름: transaction-timeout-scan-period
설명: 시간 초과 트랜잭션을 스캔하는 빈도(밀리초)입니다.

systemPropertyPrefix

유형: 문자열
기본값:
XML 이름: system-property-prefix
설명: 구성에 대한 시스템 속성을 구문 분석하는 데 사용되는 접두사입니다.

transactionTimeout

유형: long
default: 300000
XML name: transaction-timeout
description: 생성 후 리소스 관리자에서 트랜잭션을 제거하기 전의 기간(밀리초)입니다.

journalLockAcquisitionTimeout

유형: long
default: -1
XML 이름: journal-lock-acquisition-timeout
설명: 저널에서 파일 잠금을 얻기 위해 대기하는 빈도(밀리초)입니다.

journalBufferTimeout_NIO

유형: int
Default:
XML name: journal-buffer-timeout
설명: 저널의 내부 버퍼를 플러시하는 데 사용되는 시간 초과(나노초)입니다. 정확한 기본값은 저널이 ASYNCIO 또는 NIO인지 여부에 따라 달라집니다.

journalMinFiles

type: int
Default: 2

XML name: journal-min-files
description: 사전 생성할 저널 파일의 수입니다.

G.1. BRIDGECONFIGURATIONS

bridgeConfigurations.<name>.retryIntervalMultiplier

유형: double
Default: 1
XML 이름: retry-interval-multiplier
설명: 연속 재시도 간격에 적용할 수 있는 multiplier입니다.

bridgeConfigurations.<name>.maxRetryInterval

type: long
Default: 2000
XML name: max-retry-interval
description: retry-interval-multiplier로 인한 재시도 간격 증가에 대한 제한입니다.

bridgeConfigurations.<name>.filterString

유형: 문자열
기본값:
XML 이름: filter-string
설명:

bridgeConfigurations.<name>.connectionTTL

유형: long
default: 60000
XML 이름: connection-ttl
설명: 클라이언트에서 데이터를 수신하지 않은 경우 연결을 활성 상태로 유지하는 기간입니다. 기간은 ping 기간보다 커야 합니다.

bridgeConfigurations.<name>.confirmationWindowSize

type: int
Default: 1048576
XML name: confirmation-window-size
description: bridge가 확인을 보낸 후 수신된 바이트 수입니다. 바이트 표기법(예: "K", "Mb", "MiB", "GB")을 지원합니다.

bridgeConfigurations.<name>.staticConnectors

유형: 목록
기본값:
XML 이름: static-connectors
설명:

bridgeConfigurations.<name>.reconnectAttemptsOnSameNode

type: int
Default:
XML name: reconnect-attempts-on-same-node
description:

bridgeConfigurations.<name>.concurrency

type: int
Default: 1
XML name: concurrency
description: 동시 작업자 수. 더 많은 작업자가 높은 대기 시간 네트워크에서 처리량을 높일 수 있습니다. 기본값은 1입니다.

bridgeConfigurations.<name>.transformerConfiguration

type: TransformerConfiguration
Default:

XML name: transformer-configuration
description:

bridgeConfigurations.<name>.transformerConfiguration.className

유형: 문자열
기본값:
XML 이름: class-name
설명:

bridgeConfigurations.<name>.transformerConfiguration.properties

유형: Map
Default:
XML name: property
description: A KEY/VALUE to set on the transformer (예: properties.MY_PROPERTY=MY_VALUE)

bridgeConfigurations.<name>.password

유형: 문자열
기본값:
XML 이름: password
*설명: 지정되지 않은 경우 **cluster-password**가 사용됩니다.*

bridgeConfigurations.<name>.queueName

유형: 문자열
기본값:
XML 이름: queue-name
설명: 이 브리지에서 사용하는 큐의 이름입니다.

bridgeConfigurations.<name>.forwardingAddress

유형: 문자열
기본값:
XML 이름: forwarding-address
설명: 전달할 주소입니다. 생략하면 원래 주소가 사용됩니다.

bridgeConfigurations.<name>.routingType

type: ComponentConfigurationRoutingType
Default: PASS
XML name: routing-type
description: How the routing-type of the bridged messages is set.

bridgeConfigurations.<name>.name

유형: 문자열
기본값:
XML 이름: 이름
설명: 이 브리지의 고유한 이름입니다.

bridgeConfigurations.<name>.ha

type: boolean
default: false
XML name: ha
description: 이 브리지가 장애 조치를 지원하는지 여부를 지정합니다.

bridgeConfigurations.<name>.initialConnectAttempts

type: int
default: -1
XML name: initial-connect-attempts
설명: 초기 연결 시도의 최대 수입니다. 기본값 -1 은 제한이 없음을 의미합니다.

bridgeConfigurations.<name>.retryInterval

유형: long
Default: 2000
XML 이름: retry-interval
설명: 연속 재시도 사이의 간격(밀리초)입니다.

bridgeConfigurations.<name>.producerWindowSize

유형: int
Default: 1048576
XML 이름: producer-window-size
설명: Producer flow control. 바이트 표기법을 지원합니다(예: "K", "Mb", "MiB", "GB").

bridgeConfigurations.<name>.clientFailureCheckPeriod

type: long
Default: 30000
XML name: check-period
description: 브리지의 클라이언트가 서버에서 ping을 수신하지 못한지 확인하는 간격(밀리초)입니다. 이 검사를 비활성화하려면 값을 -1 로 지정합니다.

bridgeConfigurations.<name>.discoveryGroupName

유형: 문자열
기본값:
XML 이름: discovery-group-ref
설명:

bridgeConfigurations.<name>.user

유형: 문자열
기본값:

XML 이름: user
설명: 사용자 이름. 지정되지 않은 경우 cluster-user가 사용됩니다.

bridgeConfigurations.<name>.useDuplicateDetection

type: boolean
Default: true
XML name: use-duplicate-detection
Description: 전송된 메시지에 중복 탐지 헤더가 삽입되는지 여부를 지정합니다.

bridgeConfigurations.<name>.minLargeMessageSize

type: int
Default: 102400
XML name: min-large-message-size
description: 메시지가 큰 메시지로 간주되는 크기(바이트)입니다. 대규모 메시지는 여러 세그먼트로 네트워크를 통해 전송됩니다. 바이트 표기법(예: "K", "Mb", "MiB", "GB")을 지원합니다.

G.2. AMQPCONNECTIONS

AMQPConnections.<name>.reconnectAttempts

type: int
Default: -1
XML name: reconnect-attempts
description: 실패 후 재연결 시도 횟수입니다.

AMQPConnections.<이름>.password

유형: 문자열
기본값:
XML 이름: password
설명: 연결에 사용되는 암호입니다. 지정하지 않으면 익명 연결이 시도됩니다.

AMQPConnections.<name>.retryInterval

유형: int
Default: 5000
XML 이름: retry-interval
설명: 연속 재시도 사이의 간격(밀리초)입니다.

AMQPConnections.<name>.connection Cryostats

유형: AMQPMirrorBrokerConnection Cryostat
기본값:
XML 이름: amqp-connection
설명: AMQP Broker Connection은 4가지 유형을 지원합니다. 1. 미러 - 브로커는 다른 브로커에 대한 AMQP 연결을 사용하고 메시지를 복제하고 유선을 통해 승인을 보냅니다. 2. 보낸 사람 - 특정 큐에서 수신된 메시지는 다른 엔드포인트로 전송됩니다. 3. 수신자 - 브로커가 다른 끝점에서 메시지를 가져옵니다. 4. 피어 - 브로커는 다른 엔드포인트에서 발신자와 수신자를 모두 생성하여 처리 방법을 알고 있습니다. 이는 현재 Apache Cryostat Dispatch에 의해 구현되어 있습니다. 현재는 미러 유형만 지원됩니다.

AMQPConnections.<name>.connectionElements.<name>.messageAcknowledgments

type: boolean
Default:
XML name: message-acknowledgments
description: If true, message acknowledgments are mirrored.

AMQPConnections.<name>.connectionElements.<name>.queueRemoval

유형: 부울
기본값:
XML 이름: queue-removal
설명: 미러 큐가 주소와 큐의 이벤트를 삭제하는지 여부를 지정합니다.

AMQPConnections.<name>.connectionElements.<name>.addressFilter

유형: 문자열
기본값:
XML 이름: address-filter
***Description:** 미러가 소스 주소를 기반으로 대상 서버로 전달되는 이벤트를 결정하는 데 사용하는 필터를 지정합니다.*

AMQPConnections.<name>.connectionElements.<name>.queueCreation

유형: 부울
기본값:
XML 이름: queue-creation
설명: 미러 큐가 주소 및 큐에 대한 이벤트를 생성하는지 여부를 지정합니다.

AMQPConnections.<name>.autostart

type: boolean
Default: true
XML name: auto-start
***description:** 서버가 시작될 때 브로커 연결이 시작되는지 여부를 지정합니다.*

AMQPConnections.<name>.user

유형: 문자열
기본값:
XML 이름: 사용자
설명: 연결에 사용되는 사용자 이름입니다. 지정하지 않으면 익명 연결이 시도됩니다.

AMQPConnections.<name>.uri

유형: 문자열
기본값:
XML 이름: uri
*설명: **AMQP** 연결의 **URI**입니다.*

G.3. DIVERTCONFIGURATION

divertConfigurations.<name>.transformerConfiguration

type: TransformerConfiguration
Default:
XML name: transformer-configuration
description:

divertConfigurations.<name>.transformerConfiguration.className

유형: 문자열
기본값:
XML 이름: class-name
설명:

divertConfigurations.<name>.transformerConfiguration.properties

유형: Map
Default:
XML name: property
description: A KEY/VALUE to set on the transformer (예: …
properties.MY_PROPERTY=MY_VALUE).

divertConfigurations.<name>.filterString

유형: 문자열
기본값:
XML 이름: filter-string
설명:

divertConfigurations.<name>.routingName

유형: 문자열
기본값:

XML 이름: routing-name
설명: 차이점의 라우팅 이름입니다.

divertConfigurations.<name>.address

유형: 문자열
기본값:
XML 이름: address
설명: 이 주소가 다릅니다.

divertConfigurations.<name>.forwardingAddress

유형: 문자열
기본값:
XML 이름: forwarding-address
설명: 다이버의 전달 주소입니다.

divertConfigurations.<name>.routingType

type: ComponentConfigurationRoutingType(MULTICAST ANYCAST STRIP PASS)
Default:
XML name: routing-type
설명: 다양한 메시지의 라우팅 유형 설정 방법.

divertConfigurations.<name>.exclusive

유형: 부울
기본값: false
XML 이름: exclusive
설명: 이것이 배타적인지 여부를 지정합니다.

G.4. ADDRESSSETTINGS

addressSettings.<address>.configDeleteDiverts

유형: DeletionPolicy(OFF FORCE)
기본값:
XML name: config-delete-addresses
설명:

addressSettings.<address>.expiryQueuePrefix

유형: SimpleString
Default:
XML name: expiry-queue-prefix
설명:

addressSettings.<address>.defaultConsumerWindowSize

type: int
Default:
XML name: default-consumer-window-size
설명:

addressSettings.<address>.maxReadPageBytes

유형: int
기본값:
XML 이름: max-read-page-bytes
설명:

addressSettings.<address>.deadLetterQueuePrefix

유형: SimpleString
Default:
XML 이름: dead-letter-queue-prefix
설명:

addressSettings.<address>.defaultGroupRebalancePauseDispatch

유형: 부울
기본값:
XML 이름: default-group-rebalance-pause-dispatch
설명:

addressSettings.<address>.autoCreateAddresses

유형: 부울
기본값:
XML 이름: auto-create-addresses
설명:

addressSettings.<address>.slowConsumerThreshold

유형: long
기본값:
XML 이름: slow-consumer-threshold
설명:

addressSettings.<address>.managementMessageAttributeSizeLimit

type: int
Default:
XML name: management-message-attribute-size-limit
설명:

addressSettings.<address>.autoCreateExpiryResources

유형: 부울
기본값:

XML 이름: auto-create-expiry-resources
설명:

addressSettings.<address>.pageSizeBytes

유형: int
기본값:
XML 이름: page-size-bytes
설명:

addressSettings.<address>.minExpiryDelay

type: Long
Default:
XML name: min-expiry-delay
description:

addressSettings.<address>.defaultConsumersBeforeDispatch

유형: Integer
Default:
XML name: default-consumers-before-dispatch
설명:

addressSettings.<address>.expiryQueueSuffix

유형: SimpleString
Default:
XML name: expiry-queue-suffix
설명:

addressSettings.<address>.configDeleteQueues

유형: DeletionPolicy(OFF FORCE)
기본값:
XML name: config-delete-queues
설명:

addressSettings.<address>.enableIngressTimestamp

유형: 부울
기본값:
XML 이름: enable-ingress-timestamp
설명:

addressSettings.<address>.autoDeleteCreatedQueues

유형: 부울
기본값:
XML 이름: auto-delete-created-queues
설명:

addressSettings.<address>.expiryAddress

유형: SimpleString
기본값:
XML 이름: expiry-address
설명:

addressSettings.<address>.managementBrowsePageSize

유형: int
기본값:
XML 이름: management-browse-page-size
설명:

addressSettings.<address>.autoDeleteQueues

유형: 부울
기본값:
XML 이름: auto-delete-queues
설명:

addressSettings.<address>.retroactiveMessageCount

유형: long
기본값:
XML 이름: retroactive-message-count
설명:

addressSettings.<address>.maxExpiryDelay

유형: 긴
기본값:
XML 이름: max-expiry-delay
설명:

addressSettings.<address>.maxDeliveryAttempts

유형: int
Default:
XML name: max-delivery-attempts
설명:

addressSettings.<address>.defaultGroupFirstKey

유형: SimpleString
Default:
XML 이름: default-group-first-key
설명:

addressSettings.<address>.slowConsumerCheckPeriod

유형: long
기본값:
XML 이름: slow-consumer-check-period
설명:

addressSettings.<address>.defaultPurgeOnNoConsumers

유형: 부울
기본값:
XML 이름: default-purge-on-no-consumers
설명:

addressSettings.<address>.defaultLastValueKey

유형: SimpleString
Default:
XML 이름: default-last-value-key
설명:

addressSettings.<address>.autoCreateQueues

유형: 부울
기본값:
XML 이름: auto-create-queues
설명:

addressSettings.<address>.defaultExclusiveQueue

유형: 부울
기본값:

XML 이름: default-exclusive-queue
설명:

addressSettings.<address>.defaultMaxConsumers

유형: Integer
기본값:
XML 이름: default-max-consumers
설명:

addressSettings.<address>.defaultQueueRoutingType

type: RoutingType(MULTICAST ANYCAST)
Default:
XML name: default-queue-routing-type
설명:

addressSettings.<address>.messageCounterHistoryDayLimit

유형: int
Default:
XML name: message-counter-history-day-limit
설명:

addressSettings.<address>.defaultGroupRebalance

유형: 부울
기본값:
XML 이름: default-group-rebalance
설명:

addressSettings.<address>.defaultAddressRoutingType

type: RoutingType(MULTICAST ANYCAST)
Default:
XML name: default-address-routing-type
설명:

addressSettings.<address>.maxSizeBytesRejectThreshold

유형: long
기본값:
XML 이름: max-size-bytes-reject-threshold
설명:

addressSettings.<address>.pageCacheMaxSize

유형: int
기본값:
XML 이름: page-cache-max-size
설명:

addressSettings.<address>.autoCreateDeadLetterResources

유형: 부울
기본값:
XML 이름: auto-create-dead-letter-resources
설명:

addressSettings.<address>.maxRedeliveryDelay

유형: long
기본값:
XML 이름: max-redelivery-delay
설명:

addressSettings.<address>.configDeleteAddresses

유형: DeletionPolicy
Default:
XML name: config-delete-addresses
설명:

addressSettings.<address>.deadLetterAddress

유형: SimpleString
기본값:
XML 이름: dead-letter-address
설명:

addressSettings.<address>.autoDeleteQueuesMessageCount

유형: long
기본값:
XML 이름: auto-delete-queues-message-count
설명:

addressSettings.<address>.autoDeleteAddresses

유형: 부울
기본값:
XML 이름: auto-delete-addresses
설명:

addressSettings.<address>.addressFullMessagePolicy

type: AddressFullMessagePolicy
Default:
XML name: address-full-policy
description:

addressSettings.<address>.maxSizeBytes

유형: ***long***
기본값:
XML 이름: max-size-bytes
설명:

addressSettings.<address>.defaultDelayBeforeDispatch

유형: ***Long***
Default:
XML 이름: default-delay-before-dispatch
설명:

addressSettings.<address>.redistributionDelay

유형: ***long***
기본값:
XML 이름: redistribution-delay
설명:

addressSettings.<address>.maxSizeMessages

유형: ***long***
기본값:
XML 이름: max-size-messages
설명:

addressSettings.<address>.redeliveryMultiplier

유형: ***double***
Default:

XML 이름: redelivery-delay-multiplier
설명:

addressSettings.<address>.defaultRingSize

유형: long
기본값:
XML 이름: default-ring-size
설명:

addressSettings.<address>.defaultLastValueQueue

유형: 부울
기본값:
XML 이름: default-last-value-queue
설명:

addressSettings.<address>.slowConsumerPolicy

유형: SlowConsumerPolicy(KILL NOTIFY)
기본값:
XML 이름: slow-consumer-policy
설명:

addressSettings.<address>.redeliveryCollisionAvoidanceFactor

유형: double
Default:
XML 이름: redelivery-collision-avoidance-factor
설명:

addressSettings.<address>.autoDeleteQueuesDelay

유형: long
기본값:
XML 이름: auto-delete-queues-delay
설명:

addressSettings.<address>.autoDeleteAddressesDelay

유형: long
기본값:
XML 이름: auto-delete-addresses-delay
설명:

addressSettings.<address>.expiryDelay

유형: Long
기본값:
XML 이름: expiry-delay
설명:

addressSettings.<address>.enableMetrics

유형: 부울
기본값:
XML 이름: enable-metrics
설명:

addressSettings.<address>.sendToDLAOnNoRoute

유형: 부울
기본값:
XML 이름: send-to-d-l-a-on-no-route
설명:

addressSettings.<address>.slowConsumerThresholdMeasurementUnit

유형: SlowConsumerThresholdMeasurementUnit(MESSAGES_PER_SECOND MESSAGES_PER_MINUTE MESSAGES_PER_HOUR MESSAGES_PER_DAY)
기본값:
XML 이름: slow-consumer-threshold-measurement-unit
설명:

addressSettings.<address>.redeliveryDelay

유형: long
기본값:
XML 이름: redelivery-delay
설명:

addressSettings.<address>.deadLetterQueueSuffix

유형: SimpleString
기본값:
XML 이름: dead-letter-queue-suffix
설명:

addressSettings.<address>.defaultNonDestructive

유형: 부울
기본값:
XML 이름: default-non-destructive
설명:

G.5. FEDERATIONCONFIGURATIONS

federationConfigurations.<name>.transformerConfigurations

type: FederationTransformerConfiguration

Default:
XML name: transformer
description: Optional transformer configuration.

federationConfigurations.<name>.transformerConfigurations.<name>.transformerConfigurations

type: TransformerConfiguration
Default:
XML name: transformer
description: 사용자 지정 변환기를 추가하여 메시지를 수정할 수 있습니다.

federationConfigurations.<name>.transformerConfigurations.<name>.transformerConfiguration.<name>.className

유형: 문자열
기본값:
XML 이름: class-name
설명: Transformer 구현의 클래스 이름입니다.

federationConfigurations.<name>.transformerConfigurations.<name>.transformerConfiguration.<name>.properties

유형: Map
Default:
XML name: property
description: A KEY/VALUE to set on the transformer (예: ...properties.MY_PROPERTY=MY_VALUE).

federationConfigurations.<name>.queuePolicies

type: FederationQueuePolicyConfiguration
Default:
XML name: queue-policy
description:

federationConfigurations.<name>.queuePolicies.<name>.priorityAdjustment

유형: ***Integer***
Default:
XML name: priority-adjustment
description: *소비자 연결 시 해당 우선순위를 사용하여 업스트림 소비자를 생성하는 데 사용되지만 조정되어 로컬 소비자가 원격 소비자보다 분산되도록 합니다.*

federationConfigurations.<name>.queuePolicies.<name>.excludes

유형: ***Matcher***
Default:
XML name: exclude
설명: ***제외할 큐와 일치하는 큐 목록입니다.***

federationConfigurations.<name>.queuePolicies.<name>.excludes.<name>.queueMatch

유형: *문자열*
기본값:
XML 이름: queue-match
설명: ***적용할 큐 일치 패턴입니다. 존재하지 않는 경우 모든 큐가 일치합니다.***

federationConfigurations.<name>.queuePolicies.<name>.transformerRef

유형: *문자열*
기본값:
XML 이름: transformer-ref
설명: ***페더레이션 전송 시 메시지를 변환하도록 구성할 변압기의 참조 이름입니다.***

federationConfigurations.<name>.queuePolicies.<name>.includes

유형: *일치*
기본값:

XML 이름: queue-match
설명:

federationConfigurations.<name>.queuePolicies.<name>.excludes.<name>.queueMatch

유형: 문자열
기본값:
XML 이름: queue-match
설명: 적용할 큐 일치 패턴입니다. 존재하지 않는 경우 모든 큐가 일치합니다.

federationConfigurations.<name>.queuePolicies.<name>.includeFederated

유형: 부울
기본값:
XML 이름: include-federated
*설명: 값이 **false** 로 설정되면 구성이 이미 제공된 소비자(즉, 페더레이션 큐의 소비자)를 다시 제공하지 않습니다. 이렇게 하면 대칭 또는 폐쇄 루프 토폴로지에서 제공되지 않는 소비자와 메시지가 시스템 전체에서 끝없이 흐름되는 상황을 방지할 수 있습니다.*

federationConfigurations.<name>.upstreamConfigurations

type: FederationUpstreamConfiguration
Default:
XML name: upstream
description:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration

type: FederationConnectionConfiguration
Default:
XML name: connection-configuration
description: the streams connection configuration.

federationConfigurations.<name>.upstreamConfigurations.
<name>.connectionConfiguration.priorityAdjustment

유형: int
기본값:
XML 이름: priority-adjustment
설명:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.retryIntervalMultiplier

type: double
Default: 1
XML name: retry-interval-multiplier
description: The multiplier to apply to the retry-interval.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.shareConnection

유형: boolean
Default: false
XML name: share-connection
Description: true 로 설정하면 동일한 브로커에 대해 구성된 다운스트림 및 업스트림 연결이 있는 경우 동일한 연결이 공유됩니다.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.maxRetryInterval

type: long
Default: 2000
XML name: max-retry-interval
description: The maximum value for retry-interval.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.connectionTTL

유형: long
기본값:
XML 이름: connection-t-t-l
설명:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.circuitBreakerTimeout

type: long
default: 30000
XML name: circuit-breaker-timeout
description: 이 연결이 장애 조치를 지원하는지 여부를 지정합니다.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.callTimeout

유형: long
Default: 30000
XML name: call-timeout
description: The duration to wait for a reply.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.staticConnectors

유형: 목록
기본값:
XML 이름: static-connectors
설명: 커넥터를 통해 구성된 커넥터 참조 목록입니다.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.reconnectAttempts

type: int
Default: -1
XML name: reconnect-attempts
description: 실패 후 재연결 시도 횟수입니다.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.password

유형: 문자열
기본값:
XML 이름: password
설명: 암호. 지정하지 않으면 페더레이션 암호가 사용됩니다.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.callFailoverTimeout

유형: long
default: -1
XML name: call-failover-timeout
설명: 장애 조치 중에 응답을 기다리는 기간입니다. 값 -1 은 제한이 없음을 의미합니다.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.hA

유형: 부울
기본값:
XML 이름: h-a
설명:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.initialConnectAttempts

유형: int
default: -1
XML 이름: initial-connect-attempts
설명: 연결에 대한 초기 시도 횟수입니다.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.retryInterval

유형: long
Default: 500
XML name: retry-interval
description: 연속 재시도 사이의 간격(밀리초)입니다.

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.clientFailureCheckPeriod

유형: long
기본값:
XML 이름: client-failure-check-period
설명:

federationConfigurations.<name>.upstreamConfigurations.<name>.connectionConfiguration.username

유형: 문자열
기본값:
XML 이름: 사용자 이름
설명:

federationConfigurations.<name>.upstreamConfigurations.<name>.policyRefs

유형: 컬렉션
기본값:
XML 이름: policy-refs
설명:

federationConfigurations.<name>.upstreamConfigurations.<name>.staticConnectors

유형: 목록
기본값:
XML 이름: static-connectors
설명: 커넥터를 통해 구성된 커넥터 참조 목록입니다.

federationConfigurations.<name>.downstreamConfigurations

type: FederationDownstreamConfiguration
Default:
XML name: downstream
description:

federationConfigurations.<name>.downstreamConfigurations.
<name>.connectionConfiguration

type: FederationConnectionConfiguration
Default:
XML name: connection-configuration
description: the streams connection configuration.

federationConfigurations.<name>.downstreamConfigurations.
<name>.connectionConfiguration.priorityAdjustment

유형: int
기본값:
XML 이름: priority-adjustment
설명:

federationConfigurations.<name>.downstreamConfigurations.
<name>.connectionConfiguration.retryIntervalMultiplier

type: double
Default: 1
XML name: retry-interval-multiplier
description: The multiplier to apply to the retry-interval.

federationConfigurations.<name>.downstreamConfigurations.
<name>.connectionConfiguration.shareConnection

유형: boolean
Default: false
XML name: share-connection
Description: true 로 설정하면 동일한 브로커에 대해 구성된 다운스트림 및 업스트림 연결이 있는 경우 동일한 연결이 공유됩니다.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.maxRetryInterval

type: long
Default: 2000
XML name: max-retry-interval
description: The maximum value for the retry-interval.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.connectionTTL

유형: long
기본값:
XML 이름: connection-t-t-l
설명:

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.circuitBreakerTimeout

type: long
default: 30000
XML name: circuit-breaker-timeout
description: 이 연결이 장애 조치를 지원하는지 여부를 지정합니다.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.callTimeout

유형: long
Default: 30000

XML name: call-timeout
description: The duration to wait for a reply.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.staticConnectors

유형: 목록
기본값:
XML 이름: static-connectors
설명: 커넥터를 통해 구성된 커넥터 참조 목록입니다.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.reconnectAttempts

type: int
Default: -1
XML name: reconnect-attempts
description: 실패 후 재연결 시도 횟수입니다.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.password

유형: 문자열
기본값:
XML 이름: password
설명: 암호. 지정하지 않으면 페더레이션 암호가 사용됩니다.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.callFailoverTimeout

유형: long
default: -1
XML name: call-failover-timeout
설명: 장애 조치 중에 응답을 기다리는 기간입니다. 값 -1 은 제한이 없음을 의미합니다.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.hA

유형: 부울
기본값:
XML 이름: h-a
설명:

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.initialConnectAttempts

유형: int
default: -1
XML 이름: initial-connect-attempts
설명: 연결에 대한 초기 시도 횟수입니다.

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.retryInterval

유형: long
Default: 500
XML name: retry-interval
description: *연속 재시도 사이의 기간(밀리초)입니다.*

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.clientFailureCheckPeriod

유형: long
기본값:
XML 이름: client-failure-check-period
설명:

federationConfigurations.<name>.downstreamConfigurations.<name>.connectionConfiguration.username

유형: 문자열
기본값:
XML 이름: 사용자 이름
설명:

federationConfigurations.<name>.downstreamConfigurations.<name>.policyRefs

유형: 컬렉션
기본값:
XML 이름: policy-refs
설명:

federationConfigurations.<name>.downstreamConfigurations.<name>.staticConnectors

유형: 목록
기본값:
XML 이름: static-connectors
설명: 커넥터를 통해 구성된 커넥터 참조 목록입니다.

federationConfigurations.<name>.federationPolicys

유형: FederationPolicy
기본값:
XML 이름: policy-set
설명:

federationConfigurations.<name>.addressPolicies

type: FederationAddressPolicyConfiguration
Default:
XML name: address-policy
description:

federationConfigurations.<name>.addressPolicies.<name>.autoDeleteMessageCount

유형: 긴
기본값:
XML 이름: auto-delete-message-count
설명: 큐를 자동으로 삭제하기 전에 동적으로 생성된 원격 대기열에 남아 있을 수 있는 최대 메시지 수입니다.

federationConfigurations.<name>.addressPolicies.<name>.enableDivertBindings

유형: 부울
기본값:
XML 이름: enable-divert-bindings
*설명: **true** 로 설정하면 필요에 따라 다양한 바인딩을 수신할 수 있습니다. 주소 정책에 포함된 주소와 일치하는 주소가 있는 다양한 바인딩이 있는 경우 다이버트의 전달 주소와 일치하는 큐 바인딩이 수요를 생성합니다. 기본값은 **false**입니다.*

federationConfigurations.<name>.addressPolicies.<name>.includes.{NAME}.addressMatch

*유형: **Matcher***
기본값:
XML 이름: include
설명:

federationConfigurations.<name>.addressPolicies.<name>.maxHops

type: int
default:
XML name: max-hops
***description:** 메시지가 페더레이션할 수 있는 홉 수입니다.*

federationConfigurations.<name>.addressPolicies.<name>.transformerRef

유형: 문자열
기본값:

XML 이름: transformer-ref
설명: 페더레이션 전송 시 메시지를 변환하도록 구성할 변압기의 참조 이름입니다.

federationConfigurations.<name>.addressPolicies.<name>.autoDeleteDelay

type: Long
Default:
XML name: auto-delete-delay
description: 업스트림 큐를 자동으로 삭제하기 전에 다운스트림 브로커의 연결이 끊어진 후 밀리초 단위입니다.

federationConfigurations.<name>.addressPolicies.<name>.autoDelete

유형: 부울
기본값:
XML 이름: auto-delete
설명: 주소 페더레이션의 경우 다운스트림은 업스트림 주소에 있는 queue을 동적으로 생성합니다. 다운스트림의 연결이 끊어지고 지연 및 메시지 수 매개변수가 충족되면 업스트림 큐가 삭제되는지 여부를 지정합니다.

federationConfigurations.<name>.addressPolicies.<name>.excludes.{NAME}.addressMatch

유형: Matcher
기본값:
XML 이름: include
설명:

G.6. CLUSTERCONFIGURATIONS

clusterConfigurations.<name>.retryIntervalMultiplier

type: double
Default: 1
XML name: retry-interval-multiplier
description: The multiplier to apply to the retry-interval.

clusterConfigurations.<name>.maxRetryInterval

type: long
Default: 2000
XML name: max-retry-interval
description: The maximum value for retry-interval.

clusterConfigurations.<name>.address

유형: 문자열
기본값:
XML name: address
description: 이 클러스터 연결이 적용되는 주소의 이름입니다.

clusterConfigurations.<name>.maxHops

type: int
Default: 1
XML name: max-hops
description: 클러스터 토폴로지가 전파되는 최대 홉 수입니다.

clusterConfigurations.<name>.connectionTTL

유형: long
default: 60000
XML 이름: connection-ttl
설명: 클라이언트에서 데이터를 수신하지 않은 경우 연결을 활성 상태로 유지하는 기간입니다.

clusterConfigurations.<name>.clusterNotificationInterval

type: long
Default: 1000

XML name: notification-interval
description: 클러스터 연결이 클러스터에 있음을 알리는 간격입니다.

clusterConfigurations.<name>.confirmationWindowSize

type: int
Default: 1048576
XML name: confirmation-window-size
description: 서버의 데이터를 확인하는 데 사용되는 창의 크기(바이트)입니다. 바이트 표기법(예: "K", "Mb", "MiB", "GB")을 지원합니다.

clusterConfigurations.<name>.callTimeout

유형: long
Default: 30000
XML name: call-timeout
description: The duration to wait for a reply.

clusterConfigurations.<name>.staticConnectors

유형: 목록
기본값:
XML 이름: static-connectors
설명: 커넥터 이름 목록입니다.

clusterConfigurations.<name>.clusterNotificationAttempts

type: int
Default: 2
XML name: notification-attempts
description: 이 클러스터 연결이 클러스터에 해당 존재를 알리려고 하는 횟수입니다.

clusterConfigurations.<name>.allowDirectConnectionsOnly

유형: 부울
*기본값: **false***
*XML 이름: **allow-direct-connections-only***
*설명: 나열된 **connector-refs**에 대한 클러스터 연결을 제한합니다.*

clusterConfigurations.<name>.reconnectAttempts

type: int
Default: -1
XML name: reconnect-attempts
description: 실패 후 재연결 시도 횟수입니다.

clusterConfigurations.<name>.duplicateDetection

type: boolean
Default: true
XML name: use-duplicate-detection
Description: 전송된 메시지에 중복 탐지 헤더가 삽입되었는지 여부를 지정합니다.

clusterConfigurations.<name>.callFailoverTimeout

유형: long
default: -1
XML name: call-failover-timeout
*설명: 장애 조치 중에 응답을 기다리는 기간입니다. 값 **-1** 은 제한이 없음을 의미합니다.*

clusterConfigurations.<name>.messageLoadBalancingType

type: MessageLoadBalancingType(OFF STRICT ON_DEMAND OFF_WITH_REDISTRIBUTION)
Default:
XML name: message-load-balancing-type
설명:

clusterConfigurations.<name>.initialConnectAttempts

유형: int
default: -1
XML 이름: initial-connect-attempts
설명: 연결에 대한 초기 시도 횟수입니다.

clusterConfigurations.<name>.connectorName

유형: 문자열
기본값:
XML 이름: connector-ref
설명: 사용할 커넥터 참조의 이름입니다.

clusterConfigurations.<name>.retryInterval

유형: long
Default: 500
XML name: retry-interval
description: 연속 재시도 사이의 간격(밀리초)입니다.

clusterConfigurations.<name>.producerWindowSize

유형: int
Default: 1048576
XML 이름: producer-window-size
설명: Producer flow control. 바이트 표기법(예: "K", "Mb", "MiB", "GB")을 지원합니다.

clusterConfigurations.<name>.clientFailureCheckPeriod

유형: long
기본값:
XML 이름: client-failure-check-period
설명:

clusterConfigurations.<name>.discoveryGroupName

유형: 문자열
기본값:
XML 이름: discovery-group-name
설명: 이 cluster-connection에서 사용하는 검색 그룹의 이름입니다.

clusterConfigurations.<name>.minLargeMessageSize

type: int
Default:
XML name: min-large-message-size
description: 메시지가 큰 메시지로 간주되는 크기(바이트)입니다. 대규모 메시지는 여러 세그먼트로 네트워크를 통해 전송됩니다. 바이트 표기법(예: "K", "Mb", "MiB", "GB")을 지원합니다.

G.7. CONNECTIONROUTERS

connectionRouters.<name>.cacheConfiguration

type: CacheConfiguration
Default:
XML name: cache
description: 캐시 항목이 유지되고 캐시가 해당 항목을 제거하는 빈도를 제어합니다.

connectionRouters.<name>.cacheConfiguration.persisted

type: boolean
default: false
XML name: persisted
description: true 값은 캐시 항목이 유지됨을 의미합니다.

connectionRouters.<name>.cacheConfiguration.timeout

유형: int
Default: -1
XML name: timeout
description: 캐시 항목을 제거하기 전에 시간 초과(밀리초)입니다.

connectionRouters.<name>.keyFilter

유형: 문자열
기본값:
XML 이름: 키-필터
설명: 대상 키에 대한 필터입니다.

connectionRouters.<name>.keyType

type: KeyType(CLIENT_ID SNI_HOST SOURCE_IP USER_NAME ROLE_NAME)
Default:
XML name: key-type
description: 선택 사항.

connectionRouters.<name>.localTargetFilter

유형: 문자열
기본값:
XML 이름: local-target-filter
설명: 로컬 대상을 찾는 필터입니다.

connectionRouters.<name>.poolConfiguration

type: PoolConfiguration
Default:
XML name: pool
description: The pool configuration.

connectionRouters.<name>.poolConfiguration.quorumTimeout

유형: int
Default: 3000
XML name: quorum-timeout
description: 준비 대상의 최소 수를 가져오는 데 사용되는 시간 초과(밀리초)입니다.

connectionRouters.<name>.poolConfiguration.password

유형: 문자열
기본값:
XML 이름: password
설명: 대상에 액세스하기 위한 암호입니다.

connectionRouters.<name>.poolConfiguration.localTargetEnabled

type: boolean
default: false
XML name: local-target-enabled
description: true 값은 로컬 대상이 활성화됨을 나타냅니다.

connectionRouters.<name>.poolConfiguration.checkPeriod

type: int
Default: 5000
XML name: check-period
description: 대상이 준비되었는지 확인하는 데 사용되는 기간(밀리초)입니다.

connectionRouters.<name>.poolConfiguration.quorumSize

type: int
Default: 1
XML name: quorum-size
description: 준비된 최소 대상 수입니다.

connectionRouters.<name>.poolConfiguration.staticConnectors

유형: 목록
기본값:
XML 이름: static-connectors
설명: 커넥터를 통해 구성된 커넥터 참조 목록입니다.

connectionRouters.<name>.poolConfiguration.discoveryGroupName

유형: 문자열
기본값:
XML 이름: discovery-group-name
설명: 이 브릿지에서 사용하는 검색 그룹의 이름입니다.

connectionRouters.<name>.poolConfiguration.clusterConnection

유형: 문자열
기본값:
XML 이름: cluster-connection
설명: 클러스터 연결의 이름입니다.

connectionRouters.<name>.poolConfiguration.username

유형: 문자열
기본값:
XML 이름: 사용자 이름
설명: 대상에 액세스할 사용자 이름입니다.

connectionRouters.<name>.policyConfiguration

type: NamedPropertyConfiguration
Default:

XML name: policy-configuration
description:

connectionRouters.<name>.properties.{PROPERTY}

type: Properties
Default:
XML name: property
description: A set of Key value pairs specific to each named property.

2024-12-17에 최종 업데이트된 문서