ACTIVEMQ CONFIG FOR MCOLLECTIVE USER

目录

概述

Apache ActiveMQ 是 MCollective 官方推荐的中间件产品。这是一个很好的软件，但是它的 XML 配置文件庞大且笨拙，在一个复杂的 MCollective 部署中可能需要编辑很多区段。这里试图描述与 MCollective 相关的所有重要 ActiveMQ 设置。
在开始之前，可以参考网上的几个示例配置：

```
Single broker:
http://github.com/puppetlabs/marionette-collective/raw/master/ext/activemq/examples/single-broker/activemq.xml

Multi broker:
http://github.com/puppetlabs/marionette-collective/raw/master/ext/activemq/examples/single-broker/activemq.xml
```

版本限制，这里讨论的配置有相关的版本要求：

```
MCollective >= 2.0.0
ActiveMQ >=5.5.0
Stomp gem >=1.2.2
Activemq connector (included with mcollective 2.0.0 and newer)
```

对于一些配置需要在 MCollective 和 ActiveMQ 中都进行配置，其中一个的配置会影响其他配置。这里将会在**共享配置**中指出。对于 MCollective 配置文件详解，将会在其他文档中描述。

MCOLLECTIVE 如何使用 ACTIVEMQ

MCollective 通过 Stomp 协议与 ActiveMQ 建立连接，并给出一些凭证：
1) 它提供用户名和密码，ActiveMQ 可以使用这个做它能做的。
2) 如果使用 TLS，它同样会提供证书(并且验证 ActiveMQ server 的证书)
一旦允许连接，MCollective 会使用 Stomp 协议来创建订阅(subscription)。然后在以 mcollective.开头的队列和主题(queues and topics)上产生和消耗流量。

最低要求

ActiveMQ 默认相信它是在单个 JVM 实例不同进程间路由流量：它并不认为自己连接到网络，它使用一个宽松甚至不存在的安全模型。这意味着如果你仅仅启用 Stomp 流量，MCollective 将会工作的很好。(虽然有糟糕的安全问题和内存使用问题)

主题和队列名称(TOPIC AND QUEUE NAME)

MCollective 使用如下目的地名称，这个列表使用标准 **ActiveMQ destination wildcards**. “COLLECTIVE”是正在使用的 collective 的名称；默认情况下，这是 mcollective，但是如果你使用 subcollectives， 每个 subcollective 的实现同默认的 collective：mcollective 一样。

Topics:

```
ActiveMQ.Advisory.> (内建主题，所有的 ActiveMQ 生产者和消费者都需要对其有权限)
COLLECTIVE.*.agent (针对每个代理插件，这里的*是插件的名称)
```

Queues:

```
COLLECTIVE.nodes (用于直接寻址；这是使用 JMS 选择器的一个单个地址，而不是一组地址)
COLLECTIVE.replay.> (这里紧接着的部分是请求 ID)
```

共享配置：Subcollectives 同时需要在 Mcollective client 和 server 配置文件中配置。ActiveMQ 必需允许在任何 MCollective client 和 server 期望使用的 subcollective 能传输流量。

ACTIVEMQ DESTINATION WILDCARDS

ActiveMQ 支持目的地通配符来提供简单的支持联合的名称层次结构。这个概念在金融市场流行很长时间，用来作为一种以层次结构组织事件（如价格变化）的方式并且用也通配符能够很容易订阅你所感兴趣的信息范围。

假设想通过证券交易所订阅发布价格消息，可能会使用如下结构：

```
PRICE.STOCK.NASDAQ.ORCL 发布 ORACLE 在 NASDAQ 的股票价格
PRICE.STOCK.NYSE.IBM    发布 IBM 在 NYK 证券交易所的股票价格
```

订阅者同样会使用相同目的地来订阅正需要的价格，或者使用通配符来定义层次模式匹配目的地。

通配符支持,使用如下通配符：

```
.用来作为路径分隔符
*用来匹配路径任意名称
>用来递归匹配任意以这个名称开始的目的地
```

例如：

```
订阅      含义
PRICE.> 任意交易所任意产品的任意价格
PRICE.STOCK.>    任意交易所一个股票的任意价格
PRICE.STOCK.NASDAQ.*     NASDAQ 的任意股票价格
PRICE.STOCK.*.IBM    任意交易所的任意 IBM 股票价格
```

配置文件路径和格式

默认名称：activemq.xml

默认路径：

```
/etc/activemq (Red Hat-like); /etc/activemq/instances-enabled (Debian/Ubuntu)
```

文件格式： Java’s Beans XML Fromat

注意：在 ActiveMQ5.5 中，<broker>元素的一级子元素必须按照字母顺序排列，这是语法规定。在 5.6 的版本中不存在这个问题。

必需配置

不管以什么方式使用，必须设置如下配置项：

TRANSPORT CONNECTOR(S)

通常最好的做法是仅启用你需要的 transport connector。例如，如果你需要使用 Stomp over TLS，那么就不要启用单独 Stomp 传输。如果你不使用 Network of Brokers，那么就禁用 OpenWire 协议。Transport Connector 的 Name 属性只需要本地唯一即可。

Stomp

ActiveMQ 必须通过网络监听 Stomp 连接，否则 MCollective 无法连接，这时通常如下设置：

```
    <transportConnectors>
      <transportConnector name="stomp+nio"
uri="stomp+nio://0.0.0.0:61613"/>
    </transportConnectors>
```

注意这里的 Stomp URI 的协议/端口/参数还可以不同：

```
Unencrypted: stomp+nio://0.0.0.0:61613
CA-verified TLS: stomp+ssl://0.0.0.0:61614?needClientAuth=true
Anonymous TLS: stomp+ssl://0.0.0.0:61614
```

可以选择其他接口或主机名而不是使用 0.0.0.0

OpenWire

如果使用 Network of Brokers(即 ActiveMQ 集群)而不是单个 ActiveMQ Server，它们通过 OpenWire 协议相互通信，因此需要一个该协议的 Transport Connector：

```
    <transportConnectors>
      <transportConnector name="openwire+ssl"
uri="ssl://0.0.0.0:61617?needClientAuth=true"/>
    </transportConnectors>
```

对于 OpenWire URIs 协议/端口/参数还可以如下：

```
Unencrypted: tcp://0.0.0.0:61616
CA-verified TLS: ssl://0.0.0.0:61617?needClientAuth=true
Anonymous TLS: ssl://0.0.0.0:61617
```

可以选择其他接口或主机名而不是使用 0.0.0.0

Stomp 和 OpenWire 标准端口：

```
61613 for unencrypted Stomp
61614 for Stomp with TLS
61616 for unencrypted OpenWire
61617 for OpenWire with TLS
```

共享配置：

MCollective 需要知道： Stomp 使用的端口，连接 ActiveMQ 的主机名或者 IP，是否使用 TLS(如果使用 TLS，MCollective Client 和 Server 也需要相应配置)

Network of Brokers 情况下，ActiveMQ Servers 需要知道：OpenWire 使用端口，连接 ActiveMQ Server 节点的主机名或者 IP，是否使用 TLS。

回复队列处理(REPLAY QUEUE PRUNING)

MCollective 通过全局唯一的消息名称，使用名称如 mcollective.replay.<UNIQUE ID>的一次性队列发送回复消息。这些需要再五分钟后被删除，以免阻塞 ActiveMQ 的可用内存。默认情况下，队列永久存在，因此必须来配置删除策略。

为*.replay.>队列使用<policyEntry>元素。作如下配置，设置 gcInactiveDestinations 为 True，inactiveTimoutBeforeGC 为 300000ms(五分钟)：

```
    <destinationPolicy>
      <policyMap>
        <policyEntries>
          <policyEntry queue="*.reply.>" gcInactiveDestinations="true"
inactiveTimoutBeforeGC="300000" />
          <policyEntry topic=">" producerFlowControl="false"/>
        </policyEntries>
      </policyMap>
    </destinationPolicy>
```

对所有 Topics 禁用 Producer Flow Control

这是推荐设置，如之前配置示例所示，为所有的 Topics 设置 producerFlowControl 为 False。如果设置为 True，在高流量时 MCollective Servers 会出现阻塞。

推荐配置

这里列举的配置是推荐使用的配置，与安全，认证，授权相关。

TLS 证书

如果在 Stomp 或者 OpenWire Transport Connector 中使用 TLS，ActiveMQ 需要一个 keystore 文件和一个 truststore 文件，以及每个文件相应密码。

```
    <sslContext>
      <sslContext
         keyStore="keystore.jks" keyStorePassword="secret"
         trustStore="truststore.jks" trustStorePassword="secret"/>
   </sslContext>
```

示例中使用的是 CA-verified TLS，如果使用 anonymous TLS，可以忽略 truststore 属性。

这个配置默认 keystore.jks 和 truststore.jks 文件与 activemq.xml 在同一目录。

关于生成这两个文件的过程，请参考《MCollective 安全综述》文档。

这里冗余的嵌套 sslContext 不是错误，而是语法所致。这里的密码是之前创建时的密码。

编辑完 activemq.xml 文件之后，需要仔细检查确保 activemq.xml world-unreadable，因为现在它包含敏感信息。

认证（用户和组）

当连接时，MCollective Clients 和 Servers 提供 username,password 和可选的 SSL 认证。ActiveMQ 可以使用它们来作验证。一般通过添加合适的元素至<plugins>元素中来设置验证。ActiveMQ 提供三种方式：simpleAuthenticationPlugin, jaasAuthenticationPlugin, jaasCertificateAuthenticationPlugin。simpleAuthenticationPlugin 在 activemq.xml 文件中直接定义用户，它也是最简单和广泛测试的一种方式，示例如下：

```
    <plugins>
      <simpleAuthenticationPlugin>
        <users>
          <authenticationUser username="mcollective"
password="marionette" groups="mcollective,everyone"/>
          <authenticationUser username="admin" password="secret"
groups="mcollective,admins,everyone"/>
        </users>
      </simpleAuthenticationPlugin>
      <!-- ... authorization goes below... -->
    </plugins>
```

这里创建了两个用户，期望 MCollective Server 以 mcollective 登陆，而 Admin 用户以 admin 登陆。注意到除非设置授权访问，否则这些用户拥有相同能力。

共享配置：MCollective Servers 和 Clients 使用时都需要用户名和密码。这个用户必须要有对于这个 Server 或者 Client 相应的权限（权限设置如下）。

授权（组权限）

授权简单说就是给用户组设置权限。默认情况下，ActiveMQ 允许所有人对所有 topics 或 quene 拥有读权限和写权限，而 admin 能够创建 topic 或者 quene。

通过设置<authorizationPlugin>元素中的规则，可以调节权限控制。注意以下几点：

1) 权限是通过组划分的，用户组是最小的权限划分单位。
2) MCollective 在它们知道将会接收到内容之前创建 subscription。也就是说任何能够对一个 destination 读写的角色一定对这个 destination 有 admin 权限。可以将 admin 作为读写的一个超集。

基本限制示例

```
    <plugins>
      <!-- ...authentication stuff... -->
      <authorizationPlugin>
        <map>
          <authorizationMap>
            <authorizationEntries>
```

```
            <authorizationEntry queue=">" write="admins"
read="admins" admin="admins" />
            <authorizationEntry topic=">" write="admins"
read="admins" admin="admins" />
            <authorizationEntry topic="mcollective.>"
write="mcollective" read="mcollective" admin="mcollective" />
            <authorizationEntry queue="mcollective.>"
write="mcollective" read="mcollective" admin="mcollective" />
            <authorizationEntry topic="ActiveMQ.Advisory.>"
read="everyone" write="everyone" admin="everyone"/>
          </authorizationEntries>
        </authorizationMap>
      </map>
    </authorizationPlugin>
  </plugins>
```

这意味 admin 能够发出命令而 MCollective server 能够读写和应答这些命令。但是它也意味着，MCollective server 能够发出命令，这可能不是你想看到的。

注意到组‘everyone’并不是特殊的，需要手动确保所有的用户都是这个组的一员。对于 everyone,ActiveMQ 没有类似 mcollective 这样的 wildcard。注意到这里的 topic: ActiveMQ.Advisory.与前面介绍的相对应，内建主题，所有 ActiveMQ 生产者和消费者都需要对其有权限。

详细限制示例

```
  <plugins>
    <!-- ...authentication stuff... -->
    <authorizationPlugin>
      <map>
        <authorizationMap>
          <authorizationEntries>
            <authorizationEntry queue=">" write="admins"
read="admins" admin="admins" />
            <authorizationEntry topic=">" write="admins"
read="admins" admin="admins" />
            <authorizationEntry queue="mcollective.>" write="admins"
read="admins" admin="admins" />
            <authorizationEntry topic="mcollective.>" write="admins"
read="admins" admin="admins" />
            <authorizationEntry queue="mcollective.nodes"
read="mcollective" admin="mcollective" />
            <authorizationEntry queue="mcollective.reply.>"
write="mcollective" admin="mcollective" />
```

```
            <authorizationEntry topic="mcollective.*.agent"
read="mcollective" admin="mcollective" />
            <authorizationEntry
topic="mcollective.registration.agent" write="mcollective"
read="mcollective" admin="mcollective" />
            <authorizationEntry topic="ActiveMQ.Advisory.>"
read="everyone" write="everyone" admin="everyone"/>
          </authorizationEntries>
        </authorizationMap>
      </map>
    </authorizationPlugin>
  </plugins>
```

这样的设置，admin 能够发出命令而 MCollective server 能够读写和应答这些命令。但是 server 不能发出命令，这取决于 mcollective.registration.agent 目的地。

对于 admin，能够读写命令，因为它们对所有的出口 mcollective. 目的地设置。admin 控制着整个体系。

结合多 Subcollective 的权限控制

之前例子只是针对单一的 collective：mcollective。这在 mcollective 中是默认的 main_collective。如果你设置了其他的 subcollectives。它们的目的地将会以它们的名字开始而不是 mcollective。如果你需要对每个 collective 单独控制。可以使用类似以下的模板设置。

```
  <plugins>
    <!-- ...authentication stuff... -->
    <authorizationPlugin>
      <map>
        <authorizationMap>
          <authorizationEntries>
            <!-- "admins" group can do anything. -->
            <authorizationEntry queue=">" write="admins"
read="admins" admin="admins" />
            <authorizationEntry topic=">" write="admins"
read="admins" admin="admins" />

            <%- @collectives.each do |collective| -%>
            <authorizationEntry queue="<%= collective %>.>"
write="admins,<%= collective %>-admins" read="admins,<%=
collective %>-admins" admin="admins,<%= collective %>-admins" />
            <authorizationEntry topic="<%= collective %>.>"
write="admins,<%= collective %>-admins" read="admins,<%=
collective %>-admins" admin="admins,<%= collective %>-admins" />
```

```
            <authorizationEntry queue="<%= collective %>.nodes"
read="servers,<%= collective %>-servers" admin="servers,<%=
collective %>-servers" />
            <authorizationEntry queue="<%= collective %>.reply.>"
write="servers,<%= collective %>-servers" admin="servers,<%=
collective %>-servers" />
            <authorizationEntry topic="<%= collective %>.*.agent"
read="servers,<%= collective %>-servers" admin="servers,<%=
collective %>-servers" />
            <authorizationEntry topic="<%=
collective %>.registration.agent" write="servers,<%=
collective %>-servers" read="servers,<%= collective %>-servers"
admin="servers,<%= collective %>-servers" />
            <%- end -%>
            <authorizationEntry topic="ActiveMQ.Advisory.>"
read="everyone" write="everyone" admin="everyone"/>
          </authorizationEntries>
        </authorizationMap>
      </map>
    </authorizationPlugin>
  </plugins>
```

这个例子中将用户分为以下用户组：
Admins 是‘super-admins’组，能控制所有的 servers
Servers 是‘super-server’组，能够接收和响应所有的它们认为是其中一员的 collective 的命令
COLLECTIVE-admins 仅仅能够命令它们特定的 collective。（因为所有的 servers 可能都会是默认的 collective: mcollective，那么 mcollective-admins 组某种意义上类似‘super-admins’）
COLLECTIVE-servers 能够仅仅接收和响应它们特定的 collective
因此，在认证设置中定义用户时，可以通过配置其属于用户组 group=’eu_collective-admins,uk_collectives_admins’来允许特定用户来对 EU 和 UK collectives 发布命令。同时可以根据需求设置其他用户和所属组。
共享配置:这些 subcollective 同时需要在 servers 和 clients 端进行相应的配置。

NETWORK OF BROKERS 配置

可以把多个 ActiveMQ servers 分组至多个 Network of Brokers，这些 brokers 能够在它们之间路由本地 MCollective 流量。这样做有如下好处：

1) 规模-推荐每个 ActiveMQ broker 连接 800 左右 MCollective Servers，通过使用多个 broker 来进行扩展。
2) 高可用-Mcollective Clients 和 Servers 能够尝试在一个 failover pool 中连接多个 brokers。
3) 分区弹性-如果数据中心内部链接 down 掉，每个本地 MCollective 系统另一半仍工作正常。

4) 网络隔离和流量限制-如果 clients 默认仅向本地机器发送消息，你可以在能够发布一个 collective 全局命令时取得很好性能。

5) 安全-目的地过滤能够阻止特定用户向特定数据中心发送请求。

这通常比只配置单个 broker 更复杂。

关于 ActiveMQ 集群设计规划请参考文档《MCollective 生产环境安装部署思路》。这里假定已经明确：哪些 ActiveMQ brokers 能够相互通信；哪些类型流量需要从其他 broker 中排出。下面介绍其中一些必选和可选配置。

BROKER NAME

必须配置。主元素<broker>有一个<brokerName>属性，对于单个 broker 部署，这可以使用默认 localhost。在一个 network of brokers 中，每个 brokerName 必须部署环境全局唯一。重复会导致消息循环。

```
<broker xmlns="http://activemq.apache.org/schema/core"
brokerName="uk-datacenter-broker"
dataDirectory="${activemq.base}/data"
destroyApplicationContextOnStop="true">
```

OPENWIRE TRANSPORTS

必须配置。对于 network of brokers 配置，所有的 broker 需要启用 OpenWire 网络协议。具体配置可参考之前描述。

NETWORK CONNECTORS

必须配置。如果使用 network of brokers，需要配置哪些 brokers 能够相互通信。在一对连接的 brokers 其中之一的 broker 上，设置两个双向的 network connector：一个为 topics 设置，另一个是为 queues 设置。（二者唯一的不同是 queue connector 的 conduitSubscriptions 必须设置为 False）。示例配置如下：

```
    <networkConnectors>
      <networkConnector
        name="stomp1-stomp2-topics"
        uri="static:(tcp://stomp2.example.com:61616)"
        userName="amq"
        password="secret"
        duplex="true"
        decreaseNetworkConsumerPriority="true"
        networkTTL="2"
        dynamicOnly="true">
        <excludedDestinations>
          <queue physicalName=">" />
        </excludedDestinations>
      </networkConnector>
      <networkConnector
        name="stomp1-stomp2-queues"
        uri="static:(tcp://stomp2.example.com:61616)"
        userName="amq"
```

```
        password="secret"
        duplex="true"
        decreaseNetworkConsumerPriority="true"
        networkTTL="2"
        dynamicOnly="true"
        conduitSubscriptions="false">
        <excludedDestinations>
          <topic physicalName=">" />
        </excludedDestinations>
      </networkConnector>
    </networkConnectors>
```

备注：

1) 如果 OpenWire 使用 TLS，需要改变 URI 的格式为：（注意协议和端口的变化）
   static:(ssl://stomp2.example.com:61617)
2) 需要根据网络环境设置 TTL
3) 用户名和密码是必须的。推荐用户需要对所有的 queues 和 topics 拥有所有权限。
4) 每个 connector 的 name 属性需要全局唯一，最简单的做法就是结合两个主机名和 queues 或 topics
5) 可选的，也可以在两个 brokers 上分别设置非双向的 connector 连接。

目的地过滤

可选配置。假想一个星型的网络拓扑。有 uk_collective, us_collective, zh_collective 三个集群（它们各有一个中心 broker）连接至同一个中心 broker：center_broker。如果希望 center_broker 至 uk_broker 的连接仅仅传输 mcollective collective 和 uk 特定的 uk_collective collective 的流量。可以设置如下：

```
    <dynamicallyIncludedDestinations>
      <topic physicalName="mcollective.>" />
      <queue physicalName="mcollective.>" />
      <topic physicalName="uk_collective.>" />
      <queue physicalName="uk_collective.>" />
    </dynamicallyIncludedDestinations>
```

在这个配置中，连接至中心 broker 的 admin 用户能够对 uk_collective 中的节点发布命令，但是连接至 uk broker 的 admin 用户不能对 us_collective 中的节点发布命令。

换过来，如果 uk broker 连接至中心 broker，但希望其他不是 uk_collective 的节点忽略 uk_collective 特定的流量，可以配置忽略 uk_collective 目的地：

```
    <excludedDestinations>
       <topic physicalName="uk_collective.>" />
       <queue physicalName="uk_collective.>" />
    </excludedDestinations>
```

在这种情况下，连接至中心 broker 的 admin 用户无法向 uk_collective 中的节点发布命令。也就谁说它将通过 mcollective collective 来发布命令来跨过它们的界限。

优化配置

优化措施有以下几点：turn off dedicated task runner, increase heap, and increase memory and temp usage in activemq.xml。通过这些优化措施能够使 ActiveMQ 单台 server 连接至 800 左右 MCollective Servers。取决于流量和使用模式以及 topics 和 queues 的数量等。

不使用 DEDICATED TASK RUNNER

在 ActiveMQ 启动时设置 -Dorg.apache.activemq.UseDedicatedTaskRunner=false。MCollective 产生大量的 queues 和 topics，因此如果不对每个目的地使用一个线程将会节省很多内存使用。

这个配置没有在 activemq.xml 中配置；这是 JVM 的一个额外参数，在 ActiveMQ 启动时设置。在 Red Hat-like 环境下，可以在与 activemq.xml 同目录的 wrapper 配置文件中 activemq-wrapper.conf 文件中配置。配置为：

```
wrapper.java.additional.4=-Dorg.apache.activemq.UseDedicatedTaskRunner=false
```

增加 JVM HEAP

同样地，Max Heap 同样被设置在 wrapper 配置文件 activemq-wrapper.conf 中，配置如下：

```
wrapper.java.maxmemory=512
```

当然这个也可以在 ActiveMQ 启动时使用启动参数-Xmx512m

MEMORYUSAGE AND TEMPUSAGE FOR MEMMAGE

随着部署环境变大，可能需要增加<systemUsage>中的<memoryUsage>和<tempUsage>，只是暂时也没有推荐的设置。大多数用户在使用默认设置后发现问题时，选择双倍 memoryUsage 和 tempUsage 值后发现问题消除，这不是最有效的，但却是有用的策略。

```
    <systemUsage>
        <systemUsage>
            <memoryUsage>
                <memoryUsage limit="20 mb"/>
            </memoryUsage>
            <storeUsage>
                <storeUsage limit="1 gb"/>
            </storeUsage>
            <tempUsage>
                <tempUsage limit="100 mb"/>
            </tempUsage>
        </systemUsage>
    </systemUsage>
```

其他配置

这里的配置是默认配置中都有的，但是是 MCollective 不需要使用的配置。当然如果需要，可用来作为别的用途。

持久化：MCollective 很少使用，仅在 Network of Brokers 情况下使用

```
    <persistenceAdapter>
      <kahaDB directory="${activemq.base}/data/kahadb"/>
    </persistenceAdapter>
```

管理上下文:用于监控，不需要

```
    <managementContext>
      <managementContext createConnector="false"/>
    </managementContext>
```

Statics Broker

```
    <plugins>
      <!--
        Enable the statisticsBrokerPlugin to allow ActiveMQ to collect
        statistics.
      -->
      <statisticsBrokerPlugin/>
      <!-- ...auth, etc... -->
    </plugins>
```

Jetty：这个是 broker 之外

```
 <import resource="jetty.xml"/>
```