



Introduction to ActiveMQ Administration

This document shall provide a brief introduction to the management of ActiveMQ. Please refer to the project's documentation for more details.

Topics

- Project History
- Admin Console
- Config Files Overview
- Management API / Jolokia
- Client connection options
- Replication /Master/Slave
- Persistence
- Security
- Instrumentation



HISTORY OF ACTIVEMQ CLASSIC

- **2004:** ActiveMQ project initiated by LogicBlaze (later acquired by IONA).
- **2005:** Donated to the Apache Software Foundation; became a top-level Apache project.
- **2006–2010:** Rapid adoption as a reliable, open-source JMS broker.
 - Gained support for persistent messaging, clustering, and message selectors.
- **2011:** Integrated with **Apache Camel**, enabling advanced routing and EIP patterns.
- **2013–2016:** Maturity phase with support for:
 - MQTT, AMQP, STOMP protocols
 - Improved failover and store-and-forward features



ACTIVEMQ CLASSIC – RECENT EVOLUTION

- **2017:** Apache ActiveMQ **Artemis** introduced as a next-gen broker.
 - Based on HornetQ, optimized for performance and modern protocols.
- **ActiveMQ Classic** remains:
 - The **reference implementation for JMS 1.1 & 2.0**
 - Actively maintained for **legacy compatibility**
- **2020+:**
 - Regular updates (6.x series), improved security and tooling.
 - Widely used in traditional Java EE and Spring-based systems.
- **Today:** Coexists with Artemis — still relevant for existing JMS infrastructure.





ADMIN CONSOLE

ActiveMQ ships with an administration console that offers information and allows to send test messages. This section very quickly introduces most important parts. Compared to other message systems, ActiveMQ admin console offers only basic functions. This section will introduce most important dialogues.

Please note, that admin console is a Java webapplication that is hosted by the integrated Jetty app server. That means every broker can run its own console. See section [Config Files](#) for how to enable/disable admin console.



List of Queues



Home | **Queues** | Topics | Subscribers | Connections | Network | Scheduled | Send

Support | Logout

Queue Name Queue Name Filter

Queues:

Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
broker-test	0	0	0	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause
test1	0	0	0	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause
test2	0	0	0	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause
test3	0	0	0	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause
users	0	0	0	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause

Queue Views

- Graph
- XML

Topic Views

- XML

Subscribers Views

- XML


Useful Links


- Documentation
- FAQ
- Downloads
- Forums

Copyright 2005-2024 The Apache Software Foundation.



List of Topics




http://www.apache.org/

Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Support | Logout

Topic Name Create

Topics

Name ↑	Number Of Consumers	Messages Enqueued	Messages Dequeued	Operations
ActiveMQ.Advisory.MasterBroker	0	1	0	Send To Active Subscribers Active Producers Delete
ActiveMQ.Advisory.Queue	0	9	0	Send To Active Subscribers Active Producers Delete

Queue Views

- Graph
- XML

Topic Views

- XML

Subscribers Views

- XML



Useful Links

- Documentation
- FAQ
- Downloads
- Forums

Copyright 2005-2024 The Apache Software Foundation.



Send Test Messages



Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | SendSupport | Logout

Send a JMS Message

Message Header			
Destination	<input type="text" value="foo.bar"/>	Queue or Topic	<input type="text" value="Queue"/>
Correlation ID	<input type="text"/>	Persistent Delivery	<input type="checkbox"/>
Reply To	<input type="text"/>	Priority	<input type="text"/>
Type	<input type="text"/>	Time to live	<input type="text"/>
Message Group	<input type="text"/>	Message Group Sequence Number	<input type="text"/>
delay(ms)	<input type="text"/>	Time(ms) to wait before scheduling again	<input type="text"/>
Number of repeats	<input type="text"/>	Use a CRON string for scheduling	<input type="text"/>
Number of messages to send	<input type="text" value="1"/>	Header to store the counter	<input type="text" value="JMSXMessageCounter"/>

SendReset

Message body

Queue Views

- Graph
- XML

Topic Views

- XML

Subscribers Views


- XML


Useful Links

- Documentation
- FAQ
- Downloads
- Forums



List connected clients





Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Support | Logout

Create Durable Topic Subscribers ↑

Create Durable Topic Subscriber

Client ID

Subscriber Name

Topic Name

JMS Selector

Queue Views

Graph

XML

Topic Views

XML

Subscribers Views

XML

Useful Links

Documentation

FAQ

Downloads

Forums

Active Durable Topic Subscribers

Client ID	Subscription Name	Connection ID	Destination	Selector	Pending Queue Size	Dispatched Queue Size	Dispatched Counter	Enqueue Counter	Dequeue Counter	Operations
-----------	-------------------	---------------	-------------	----------	--------------------	-----------------------	--------------------	-----------------	-----------------	------------

Offline Durable Topic Subscribers

Client ID	Subscription Name	Connection ID	Destination	Selector	Pending Queue Size	Dispatched Queue Size	Dispatched Counter	Enqueue Counter	Dequeue Counter	Operations
-----------	-------------------	---------------	-------------	----------	--------------------	-----------------------	--------------------	-----------------	-----------------	------------



Active Non-Durable Topic Subscribers

Client ID	Subscription Name	Connection ID	Destination	Selector	Pending Queue Size	Dispatched Queue Size	Dispatched Counter	Enqueue Counter	Dequeue Counter	Operations
sample-...		ID:mark...	ActiveM...		0	0	0	0	0	

Copyright 2005-2024 The Apache Software Foundation.



Broker Protocols - which connections offers your instance



Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Support | Logout

Connections

Connector http

Name	Remote Address	Active	Slow
------	----------------	--------	------

Connector ws

Name	Remote Address	Active	Slow
------	----------------	--------	------

Connector openwire

Name	Remote Address	Active	Slow
sample-client-org.apache.activemq.ActiveMQConnectionFactory@765d7657	tcp://127.0.0.1:50266	true	false

Network Connectors

Name	Message TTL	Consumer TTL	Dynamic Only	Conduit Subscriptions	Bridge Temps	Decrease Priorities	Dispatch Async
------	-------------	--------------	--------------	-----------------------	--------------	---------------------	----------------

Queue Views

- Graph
- XML

Topic Views

- XML

Subscribers Views

- XML



Useful Links

- Documentation
- FAQ
- Downloads
- Forums

Copyright 2005-2024 The Apache Software Foundation.



Broker Network - all running & connected instances



Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Support | Logout

Network Bridges

Remote Broker	Remote Address	Created By Duplex	Messages Enqueued	Messages Dequeued
replica02	tcp://127.0.0.1:61626	false	0	0

Queue Views

- Graph
- XML

Topic Views

- XML

Subscribers Views

- XML

Useful Links

- Documentation
- FAQ
- Downloads
- Forums

Copyright 2005-2024 The Apache Software Foundation.



Tasks

- Create a queue *sample-queue* and send a message
- Use simple-listener to receive message
- Create a topic *sample-topic* and send a message
- Use simple-listener to receive message
- Connect simple listener/producer and observe results in Admin console
- Change admin password, [hint](#)
- Disable admin console and restart instance



CONFIG FILES OVERVIEW

- Config files main mechanism to configure ActiveMQ
- Multiple files with multiple notations -> easy to confuse
- Broker setup vs authentication vs authorization vs persistence ... -> easy to confuse
- Embedded instances, Docker images



1. `activemq.xml`

- **Location:** `$ACTIVEMQ_HOME/conf/`
- **Purpose:** Primary broker configuration file.
- **Details:**
 - Defines broker instances, transport connectors (e.g., OpenWire, MQTT, AMQP), persistence, destinations (queues/topics), and plugins.
 - Customizable using Spring XML syntax.
 - Example: Add SSL connectors, configure persistence adapters, or define virtual destinations.



2. `jetty.xml`

- **Location:** `$ACTIVEMQ_HOME/conf/`
- **Purpose:** Configures the embedded Jetty web server.
- **Details:**
 - Manages HTTP endpoints, including the web console and REST interfaces.
 - Can be used to secure the web interface with SSL or basic authentication.



3. `jetty-realm.properties`

- **Location:** `$ACTIVEMQ_HOME/conf/`
- **Purpose:** Defines web console user credentials.
- **Details:**
 - Used in conjunction with `jetty.xml` for basic authentication.
 - Format: `username: password, role`



4. login.config

- **Location:** `$ACTIVEMQ_HOME/conf/`
- **Purpose:** JAAS (Java Authentication and Authorization Service) login module configuration.
- **Details:**
 - Specifies authentication mechanisms (e.g., property files, LDAP).
 - Used when `jaasAuthenticationPlugin` is enabled in `activemq.xml`.



5. `users.properties`

- **Location:** `$ACTIVEMQ_HOME/conf/`
- **Purpose:** Defines user credentials for JAAS authentication.
- **Details:**
 - Format: `username=password`
 - Used when JAAS is configured with `PropertiesLoginModule`.



6. `groups.properties`

- **Location:** `$ACTIVEMQ_HOME/conf/`
- **Purpose:** Maps users to groups for authorization.
- **Details:**
 - Format: `group=user1,user2,...`
 - Required when using group-based access control.



7. `log4j.properties`

- **Location:** `$ACTIVEMQ_HOME/conf/`
- **Purpose:** Controls logging behavior.
- **Details:**
 - Defines log levels, appenders, and logging formats.
 - Helps with troubleshooting broker and client behavior.



8. `activemq.policy` (optional)

- **Location:** `$ACTIVE MQ_HOME/conf/`
- **Purpose:** Java security policy file (if security manager is used).
- **Details:**
 - Defines permissions for classes and code sources.
 - Used when running ActiveMQ with a security manager.



JOLOKIA API

Jolokia is a management API, with which all configuration and functions of ActiveMQ can be observed and manipulated. In fact admin console is build using this API.

In order to run the following examples, a running broker instance is necessary. See [Hello world example](#) how to run an instance. If you use another instance responses may vary.

First example returns a list of everything, that is accessible via Jolokia.

```
curl -XGET -u admin:admin -H "Origin:http://localhost" http://localhost:8161/api/jolokia/list
```

Note that it is necessary to set origin header. Without it ActiveMQ will refuse to answer - see section [security](#).



Next example lists all available destinations. Note that broker name needs to be set according to your instance.

```
curl -XGET -u admin:admin -H "Origin:http://localhost" "http://localhost:8161/api/jolokia/search/org.apache.activemq:type=Broker,brokerName=single,destinationType=Queue,destinationName=*"
```

Response will look like this. Note, if no queue exists yet, response will be empty.

```
{
  "request": {
    "mbean": "org.apache.activemq:brokerName=single,destinationName=*,destinationType=Queue,type=Broker",
    "type": "search"
  },
  "value": [
    "org.apache.activemq:brokerName=single,destinationName=test01,destinationType=Queue,type=Broker",
    "org.apache.activemq:brokerName=single,destinationName=test,destinationType=Queue,type=Broker"
  ],
  "status": 200
}
```



The following two example query an MBean. The first one collects all attributes of broker object.

```
curl -XGET -u admin:admin -H "Origin:http://localhost" "http://localhost:8161/api/jolokia/read/org.apache.activemq:type=Broker,brokerName=single/*"
```

This request will result in a response like this.

```
{
  "request": {
    "mbean": "org.apache.activemq:brokerName=single,type=Broker",
    "type": "read"
  },
  "value": {
    "StatisticsEnabled": true,
    "TemporaryQueueSubscribers": [],
    "TotalConnectionsCount": 1,
    "TotalMessageCount": 1,
    "TempPercentUsage": 0,
    "MemoryPercentUsage": 0,
    "TransportConnectors": {
      "openwire": "tcp://markus-workstation-starwit:61616?maximumConnections=1000&wireFormat.maxFrameSize=104857600",
      "http": "http://markus-workstation-starwit:8080?transport.maxConnections=1000&transport.wireFormat.maxFrameSize=104857600&transport.keepAlive=true&transport.trace=true&transport.useKeepAlive=true&transport.connectionTimeout=30000",
      "ws": "ws://markus-workstation-starwit:61614?maximumConnections=1000&wireFormat.maxFrameSize=104857600"
    }
  },
  ....
}
```



If you want to query a single field - very handy to build a simple monitoring script - the following example gets a single field.

```
curl -XGET -u admin:admin -H "Origin:http://localhost" "http://localhost:8161/api/jolokia/read/org.apache.activemq:type=Broker,brokerName=single/TotalMessageCount"
```

There is a lot more this API can be used for. For an overview of all Mbeans see documentation here:

<https://activemq.apache.org/components/classic/documentation/jmx>

An other way to get a full list can be found in section [Java Mission Control](#).

Tasks

- Add a queue on admin console and observe output of queue listing.
- Modify destination request to topics. Add topics on admin console and observe output.
- Add an additional protocol to single instance (see next section), restart broker and query broker object.



PROTOCOLS

ActiveMQ offers a number of communication protocols. For each protocol a connector needs to be configured. Following configuration extract is from *activemq.xml*. It shows connectors for the **single** instance.

```
<transportConnectors>
  <!-- DOS protection, limit concurrent connections to 1000 and frame size to 100MB -->
  <transportConnector name="openwire" uri="tcp://0.0.0.0:61616?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
  <transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
  <transportConnector name="http" uri="http://0.0.0.0:8080?transport.maxConnections=1000&transport.wireFormat.maxFrameSize=104857600&transport.keepAlive=true&transport.trace=true&transport.useKeepAlive=true&transport.connectionTimeout=30000" />
</transportConnectors>
```

For obvious reasons your instances should only offer protocols, that your clients need. So the example shows only a subset of the supported protocols. A complete list can be found here:

<https://activemq.apache.org/components/classic/documentation/protocols>



Tasks

- Select a protocol and add a connector to single instance. Start broker and check via admin console, that new connector is available.
- **Bonus challenge:** Enable MQTT and send/receive messages using a MQTT messaging tool



CLIENT CONNECTION OPTIONS

<https://activemq.apache.org/components/classic/documentation/uri-protocols>



REPLICATION

One of the core features of message brokers, is the possibility to run multiple instances. This way setups that are able to deal with high loads become possible as well as minimizing outages via replication. This section is a (brief) introduction into this topic.

Reference: <https://activemq.apache.org/components/classic/documentation/networks-of-brokers>

Please note that high-availability is very hard to achieve, as many, many more aspects than message broker config needs to be configured and set up properly. So if you aim at HA start by familiarizing yourself with the general concept. A good starting point is [Wikipedia](#).



Static

Most basic configuration of replicas is defining a fixed set of broker instances. Any instance then needs to know each other. Here is an example, how to configure this connection:

```
<networkConnectors>  
  <networkConnector uri="static:(tcp://localhost:61616)" />  
</networkConnectors>
```

[See here](#) for full config file.

Example setup can be found in folder broker-configs/replica.

Find in section [Replication](#) instructions how to run the example configuration.

Question: Any idea why this approach may not scale?



Discovery/Multicast

ActiveMQ also supports dynamic broker configuration. Here new instances discover already running brokers and are then ad-hoc added.

Necessary config in [activemq.xml](#) looks like this

```
<networkConnectors>
  <networkConnector uri="multicast://default"/>
</networkConnectors>

<transportConnectors>
  <transportConnector uri="tcp://localhost:0" discoveryUri="multicast://default"/>
</transportConnectors>
```

Find in section [Replication](#) instructions how to run the example configuration.

Question: Does multicast work in dynamic environments and why/why not?



Master/Slave

Another replication setup is called master/slave and here two or more instances can run on the same machine and only one is active, while the other ones waiting to stand-in if master goes down.

This can be achieved by sharing persistence storage between instance. In our example this will look like this:

```
<persistenceAdapter>  
  <kahaDB directory="../../persistence"/>  
</persistenceAdapter>
```

Note: This configuration is not advisable in productive environments!

Full example and instructions to run are located [here](#).

Question: Does this setup make sense in technologies like Kubernetes?



SECURITY

- Broker instances so far not secured - don't use them in production
- Brief intro into security configuration
- Full documentation <https://activemq.apache.org/components/classic/documentation/security>
- Topics
 - Authentication & Roles - Change passwords, add users
 - Authorization & auto creation of destinations
 - Disable Admin Console!
 - Notes on CORS
 - Static destination creation



Authentication & Roles

Clients should never use admin privilege to send/receive messages

Add non-privileged group in *groups.properties*

```
admins=admin  
clients=client1,client2
```

Add credentials for non-privileged users in *users.properties*

```
admin=admin  
client1=client1  
client2=client2
```



Authorization / Disable auto creation of destinations

Following extract from [activemq.xml](#) shows how to allow destination creation role based.

```
<plugins>
  <jaasAuthenticationPlugin configuration="activemq" />
  <authorizationPlugin>
    <map>
      <authorizationMap>
        <authorizationEntries>
          <authorizationEntry topic="ActiveMQ.Advisory.>"
            read="admins,clients"
            write="admins,clients"
            admin="admins,clients"/>
          <authorizationEntry queue=">" read="admins,clients"
            write="admins,clients"
            admin="admins" />
          <authorizationEntry topic=">" read="admins,clients"
            write="admins,clients"
            admin="admins" />
        </authorizationEntries>
      </authorizationMap>
    </map>
  </authorizationPlugin>
</plugins>
```



Static destination creation

In some scenarios dynamic creation of destination may be undesirable. Following snippet from [activemq.xml](#) shows, how to statically create queues/topics.

```
<destinations>
  <queue physicalName="prefonfigured.queue" />
  <topic physicalName="prefonfigured.topic" />
</destinations>
```



Disable Admin Console

- Admin console is very powerful - very dangerous!
- Should be disabled in production
- If not disabled , unreachable from outside networks
- Disable by commenting out this section in [activemq.xml](#):

```
<!--  
    Enable web consoles, REST and Ajax APIs and demos  
    The web consoles requires by default login, you can disable this in the jetty.xml file  
  
    Take a look at ${ACTIVEMQ_HOME}/conf/jetty.xml for more details  
-->  
<import resource="jetty.xml"/>  
  
</beans>
```



PERSISTENCE



INSTRUMENT & MONITOR ACTIVEMQ

As a Java application ActiveMQ can be instrumented and monitored using many standard tools. In this section, we will look into some of them. If you are interested in more sophisticated monitoring solutions, have a look into [Open Telemetry example](#).



Java Mission Control

ActiveMQ makes use of Java's JMX interface and thus any JMX tool can be used, to observe a running broker. One of those tools is Java Mission Control. If you run in on the same machine as your broker instance, you can connect it right away.

- For a generell intro to JMX see here: https://en.wikipedia.org/wiki/Java_Management_Extensions
- Download JMC here: <https://jdk.java.net/jmc/9/>



Once connected, you can browse all available MBeans (JMX' way to expose data and functions). See the following picture for an example:

[22.0.1] .../.../apache-activemq-6.1.6/bin/activemq.jar start (48692) X

MBean Browser

MBean Tree

Filter:

> JImplementation

> com.sun.management

> java.lang

> GarbageCollector

> MemoryManager

> MemoryPool

ClassLoading

Compilation

Memory

OperatingSystem

Runtime

Threading

> java.nio

> java.util.logging

> jdk.management.jfr

> jolokia

> Config

100.90.218.29-48692-31834a2b-servlet

> ServerHandler

100.90.218.29-48692-31834a2b-servlet

> org.apache.activemq

> Broker

> single

> PersistenceAdapter

> Queue

> broker-test

test1

test2

test3

> Topic

> ActiveMQ.Advisory.TempQueue_ActiveMQ.Advisory.TempTopic

ActiveMQ.Advisory.Connection

ActiveMQ.Advisory.Consumer.Queue.broker-test

ActiveMQ.Advisory.MasterBroker

ActiveMQ.Advisory.Producer.Queue.broker-test

ActiveMQ.Advisory.Queue

> clientConnectors

Health

Log4JConfiguration

MBean Features

AttributesOperationsNotificationsMetadata

Operations

addConnector : String

addNetworkConnector : String

addQueue : void

addTopic : void

browseQueue : CompositeData[]

createDurableSubscriber : javax.management.ObjectName

destroyDurableSubscriber : void

disableStatistics : void

enableStatistics : void

gc : void

getTransportConnectorByType : String

queryQueues : String

queryTopics : String

reloadLog4jProperties : void

removeConnector : boolean

removeNetworkConnector : boolean

removeQueue : void

removeTopic : void

resetStatistics : void

restart : void

start : void

stop : void

stopGracefully : void

terminateJVM : void

Name

Value

name

test3

Execute

addQueue("test3")

14/05/2025, 17:23:06 Result for 'addQueue("test3")' X



Tasks

- Start a broker instance
- Run JMC and connect to instance
- Choose on Mbean and extract value, trigger a method



HawtIO

TODO



ActiveMQ, Prometheus, Grafana

Another way to get insights of a running ActiveMQ instance is by extracting data via a Java agent. A pre-configured instance in folder `single-jmx` is provided, that exposes metrics on port 7878. In this example these metrics are collected by Prometheus and visualized with Grafana. Setup shall provide a development environment, to explore metrics and visualizations. It is not intended to be used in a productive environment.

Start instance *single-jmx* like so:

Windows	Linux
<pre>cd single-jmx\bin single.bat start</pre>	<pre>cd single-jmx/bin ./single.sh start</pre>

Metrics should now be available under <http://localhost:7878/metrics>

Start Prometheus & Grafana

Prometheus is a metric collector and is used in many monitoring setups. Grafana is a visualization tool, that can be used to create dashboards displaying metric data. In order run and configure both tools, we can either use Docker compose or manual setup.



Docker Compose

Docker is necessary to run compose scripts. Install for your [Linux](#) distribution or [Docker Desktop](#) for Windows.

Now run Docker compose file like so:

```
cd docker-compose
docker compose -f compose-monitoring.yaml up
```

This is going to run Prometheus (<http://localhost:9090/>) and Grafana (<http://localhost:3000/>). Both are pre-configured such that:

- Prometheus collects metrics from ActiveMQ
- Grafana uses Prometheus as data source
- An ActiveMQ dashboard displaying metrics



Manual Setup

Download Prometheus here: <https://prometheus.io/download/> In folder `prometheus` you find a config file, that can be used to run Prometheus. Check that metrics are collected.

To setup Grafana manually:

- download and install Grafana: <https://grafana.com/grafana/download>
- configure Prometheus data source
- import `Dashboard`

Tasks

- Run setup
- Connect producer/listener and observe output