AutoPilot® M6 Plug-in for Kafka JMX

Installation and User's Guide

Document Title: Autopilot M6 Plug-in for Kafka JMX
Document Release Date: March 2025
Document Number: AP/KAF 610.003.3
Product Release: 6.0.34.7

Published by:

Research & Development

meshIQ

88 Sunnyside Blvd, Suite 101

Plainview, NY 11803

Table of Contents

CHAPTER 1 INTRODUCTION 1

1.1 How This Guide is Organized 1
1.2 History of This Document 1
1.2.1 User Feedback 2
1.3 Related Documents 2
1.4 Intended Audience 2
1.5 System Requirements 2
1.5.1 Platforms 2
1.5.2 Other Requirements 2
1.6 Technical Support 2
1.7 Conventions 2

CHAPTER 2 ABOUT AUTOPILOT/KAFKA EXPERT 4

2.1 Introduction 4

CHAPTER 3: INSTALLATION & CONFIGURATION 6

3.1 Installation Preparation 6
3.1.1 Installation Materials 6
3.1.2 Licensing Information 6
3.2 Pre-Requisites 6
3.2.1 Java 6
3.2.2 Enable JMX for Remote Communication on Kafka Component 6
3.2.3.1 Configuring Apache Kafka with SSL 8
3.2.3.2 Setting up a Kafka client with SSL 16
3.2.3 Deploying Process Wrapper 23
3.3 Installation 33
3.4 Configuration: Securing Kafka Server Components 34
3.5 Configuration: Using TNT4J Stream-JMX for Kafka Monitoring 37
3.5.1 Configuring Client-Side Properties using SSL 37
3.5.2 Configure tnt4j.properties 41
3.5.3 ConnectionsStanzaZK_*.cfg 42
3.5.3.1 JMX Sampling Agent (sampler) options 43
3.5.3.2 Using Zookeeper Orchestrated Approach 46
3.5.4 General Stream JMX Configuration 47
3.5.4.1 System Properties Used 48
3.6 Starting Stream JMX 50
3.6.1 Start Using Default Scripts Shipped with Package 50
3.6.2 Start from CLI 50
3.6.2.1 Using stream-jmx-connect utility available under executables ($STREAM_JMX_HOME/bin) 50
3.6.2.2 To connect to local JVM process 51
3.6.2.3 To Connect to JMX Service Over URL 51

CHAPTER 4: KAFKA JMX METRICS 54

4.1 Zookeeper Metrics 55
4.2 Broker Metrics 56
4.3 Connect Metrics 57
4.4 Schema Registry Metrics 58
4.5 KSQL Metrics 59
4.6 Kafka Rest Proxy Metrics 60

CHAPTER 5: KAFKA JMX SAMPLE POLICIES 62

5.1 Create & Deploy Policy Manager 62
5.2 Deploy Policies Under Created Policy Manager 62
5.3 Kafka Resources 63

APPENDIX A: REFERENCES 66

A.1 MESHIQ DOCUMENTATION 66
APPENDIX B: CONVENTIONS 68
B.1 TYPOGRAPHICAL CONVENTIONS 68
B.2 NAMING CONVENTIONS 69

Figures

FIGURE 3-1. CONNECTING TO JCONSOLE 7
FIGURE 3-2. ZOOKEEPER METRICS THROUGH JCONSOLE 8
FIGURE 3-3. DEPLOY PROCESS WRAPPER 24
FIGURE 3-4. MODIFY KAFKA MONITOR 25
FIGURE 3-5. CREATE KAFKA MONITOR – GENERAL TAB 25
FIGURE 3-6. CREATE KAFKA MONITOR – FACT OPTIONS TAB 26
FIGURE 3-7. CREATE KAFKA MONITOR – LOGGING TAB 27
FIGURE 3-8. CREATE KAFKA MONITOR – RECORDING TAB 28
FIGURE 3-9. CREATE KAFKA MONITOR – RESTART-RECOVERY TAB 29
FIGURE 3-10. CREATE KAFKA MONITOR – SECURITY TAB 30
FIGURE 3-11. CREATE KAFKA MONITOR – STREAMING OPTIONS TAB 31
FIGURE 3-12. CREATE KAFKA MONITOR – TCP OPTIONS TAB 31
FIGURE 3-13. CREATE KAFKA MONITOR – UDP OPTIONS TAB 32
FIGURE 3-14 CONFIGURED CERTIFICATES 35
FIGURE 3-15 SSL ENABLED FOR KAFKA RUNNING 36
FIGURE 3-16 CREATED CLIENT CERTIFICATES 37
FIGURE 3-17 KAFKA JMX WITH SSL METRICS 40
FIGURE 4-1. ZOOKEEPER METRICS 55
FIGURE 4-2. BROKER METRICS 56
FIGURE 4-3. CONNECT METRICS 57
FIGURE 4-4. SCHEMA REGISTRY METRICS 58
FIGURE 4-5. KSQL METRICS 59
FIGURE 4-6. KAFKA REST PROXY METRICS 60
FIGURE 5-1. DEPLOY POLICY MANAGER 62
FIGURE 5-2. DEPLOY POLICY UNDER POLICY MANAGER 63
FIGURE 5-3. KAFKA RESOURCES 64

Tables

TABLE 1-1. DOCUMENT HISTORY 1
TABLE 3-1. KAFKA MONITOR – GENERAL PROPERTIES 25
TABLE 3-2. KAFKA MONITOR – FACT OPTIONS PROPERTIES 26
TABLE 3-3. KAFKA MONITOR – LOGGING PROPERTIES 27
TABLE 3-4. KAFKA MONITOR – RECORDING PROPERTIES 28
TABLE 3-5. KAFKA MONITOR – RESTART-RECOVERY PROPERTIES 29
TABLE 3-6. KAFKA MONITOR – SECURITY PROPERTIES 30
TABLE 3-7. KAFKA MONITOR – STREAMING OPTIONS PROPERTIES 31
TABLE A-1. MESHIQ DOCUMENTATION 66
TABLE B-1. TYPOGRAPHICAL CONVENTIONS 68
TABLE B-2. AUTOPILOT RELATED NAMING CONVENTIONS 69

Chapter 1 Introduction

Welcome to the meshIQ Plug-in for Kafka Expert Installation and User's Guide. The expert is compatible with Apache Kafka, Confluent Kafka & Tibco Kafka. This guide describes installation and use of the Kafka expert. This plug-in is designed to work with AutoPilot, its components, and other plug-ins, and run simultaneously without interference or performance degradation.

1.1 How This Guide is Organized

Chapter 1: Identifies the users and history of the document. System requirements for this plug-in are outlined. All other system and platform information is listed in the AutoPilot Installation and User's Guides.

Chapter 2: Contains a brief description of Kafka Expert.

Chapter 3: Provides instruction for new installations of the Kafka Expert.

Chapter 4: Metrics collected by Kafka Expert, for each Kafka components (Zookeeper, Broker, Connect, Schema Registry, KSQL, Rest Proxy.

Chapter 5: Sample Policies shipped with Kafka Expert

Appendix A: Provides a detailed list of all reference information required for the installation of AutoPilot.

Appendix B: Contains conventions used in AutoPilot and documents typographical conventions.

1.2 History of This Document

Table 1-1. Document History

Release Date:	Document Number	For AutoPilot Version	Summary
February 2018	**AP/KAF 610.001**	**AP 6.0 or higher**	**Original issue**
August 2018	**AP/KAF 610.002**	**AP 6.0 or higher**	**General update**
January 2021	**AP/KAF 610.003**	**AP 6.0 or higher**	**Updates to 3.3.1, 3.3.1.4, 3.3.2 and 3.3.2.3.**
May 2022	**AP/KAF 610.003.1**	**AP 6.0 or higher**	**Changed title to *AutoPilot® M6 Plug-in for Kafka Installation and User's Guide***
October 2024	**AP/KAF 610.003.2**	**AP 6.0 or higher**	**Revised Chapters 2 and 3. Combined Chapter 4 content into Chapter 3. Moved Chapter 5 to Chapter 4 with revised sections. Moved Chapter 6 to Chapter 5 and added a new section. Updated chapter numbering for consistency and made necessary revisions.**
March 2025	**AP/KAF 610.003.3**	**AP 6.0 or higher**	**Added sections 3.4, and 3.5.1**

1.2.1User Feedback

meshIQ encourages all Users and Administrators of AutoPilot to submit comments, suggestions, corrections and recommendations for improvement for all AutoPilot documentation. Please send your comments via email to: *support@meshiq.com*. You will receive a written response, along with the status of any proposed change, update, or correction.

1.3 Related Documents

The complete listing of related and referenced documents is listed in *Appendix A* of this guide.

1.4 Intended Audience

The Kafka Expert Guide is intended for use by installers and administrators of meshIQ AutoPilot with Apache Kafka and related components.

1.5 System Requirements

This section defines system and platform prerequisite support requirements for Kafka Expert.

1.5.1Platforms

Kafka Expert is compatible with the following platforms:

- Windows NT/2000 or later/XP
- Unix (Linux)

1.5.2Other Requirements

Kafka Expert requires the following conditions:

- AutoPilot 6.0 or higher
- Java Run Time Environment 11.x
- Apache Kafka or Confluent Kafka or Tibco Kafka
- Target operating system environment
- Installer may need administrative privileges for the target platform
- Since communication between Kafka and AutoPilot is done via JMX it is necessary to have a proper installed configuration for operation of the expert. (Refer to *section 3.2.2*).

1.6 Technical Support

If you need additional technical support, you can contact meshIQ by telephone or by e-mail. To contact meshIQ technical support by telephone, call **(800) 963-9822 ext. 1**, if you are calling from outside the United States dial **001-516-801-2100**. To contact meshIQ technical support by email, send a message to *mysupport@meshiq.com*. To access the meshIQ automated support system, go to *http://mysupport.meshiq.com/*. (A user name and password are required.) Contact your local AutoPilot Administrator for further information.

1.7 Conventions

Refer to *Appendix B* for conventions used in this guide.

This Page Intentionally Left Blank

Chapter 2 About AutoPilot/Kafka Expert

2.1 Introduction

Stream JMX is designed to monitor and manage your Apache Kafka and related components. Other flavours of Kafka and its components shipped by vendors like Confluent and Tibco can also be monitored. Information is processed by the Stream JMX process and integrated into the AutoPilot infrastructure. Communication with all Kafka components is via JMX either locally or remotely across a network, and all metrics produced via JMX can be collected & monitored. Components that can be monitored include:

- Zookeeper
- Kafka Broker
- Kafka Connect
- Confluent Schema Registry
- Confluent KSQL
- Confluent Rest Proxy
- Kafka Mirror Maker 2.0

This information can be combined with information provided by AutoPilot for other components, such as the operating system or log files, to get further insight into the performance and operation of the Apache Kafka environment.

This Page Intentionally Left Blank

Chapter 3: Installation & Configuration

3.1 Installation Preparation

This section contains general information related to preparing for and installing the Kafka Expert software.

3.1.1Installation Materials

Installation can be performed from installation media or by download through the meshIQ Resource Center.

Prior to installation, review all text files and installation procedures provided on the meshIQ Resource Center. It is recommended that all installation related materials are printed to allow the installer to review them prior to installation, and better to follow the detailed instructions within.

3.1.2Licensing Information

A copy of the standard Licensing Agreement is imbedded in the installation software and is provided on the Resource Center. The formal licensing agreement has been furnished in the purchase agreement package.

3.2 Pre-Requisites

This section lists the prerequisites that need to be completed before installing Stream JMX, along with the steps required to do so.

3.2.1Java

Stream JMX is a Java-based process that requires Java to be installed on the VM or server where it starts. You can use the same Java installation that AP uses for Stream JMX deployments.

3.2.2Enable JMX for Remote Communication on Kafka Component

Stream JMX communicates with and collects metrics from the Kafka component over JMX. Since Kafka does not enable this by default, you should define it in the startup scripts of all Kafka components. The example provided below shows how to enable JMX on the Kafka broker component. You can use the same options for all Kafka components; the only difference will be the environment variable names.

```
export KAFKA_JMX_OPTS="-Dcom.sun.management.jmxremote=true -
Dcom.sun.management.jmxremote.host=HostName -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.local.only=false -
Dcom.sun.management.jmxremote.ssl=false

export JMX_PORT=9991
```

For other few Kafka Components, Confluent suggests using a separate environment variable, instead of KAFKA_JMX_OPTS. For components not mentioned below should use KAFKA_JMX_OPTS.

REST Proxy	KAFKAREST_JMX_OPTS
ksqlDB	KSQL_JMX_OPTS
Schema Registry	SCHEMA_REGISTRY_JMX_OPTS

To validate whether JMX is enabled correctly, use jconsole from a windows host. The Java Monitoring & Management console (AKA jconsole) is available, if java is installed on the windows host, and JAVA_HOME, JRE_HOME environment variables should be set correctly.

To test, open a command prompt and type **jconsole,** press enter. This will open Java monitoring & management window, where remote host:port should be provided. The same window also allows defining user credentials, if authentication is enabled.

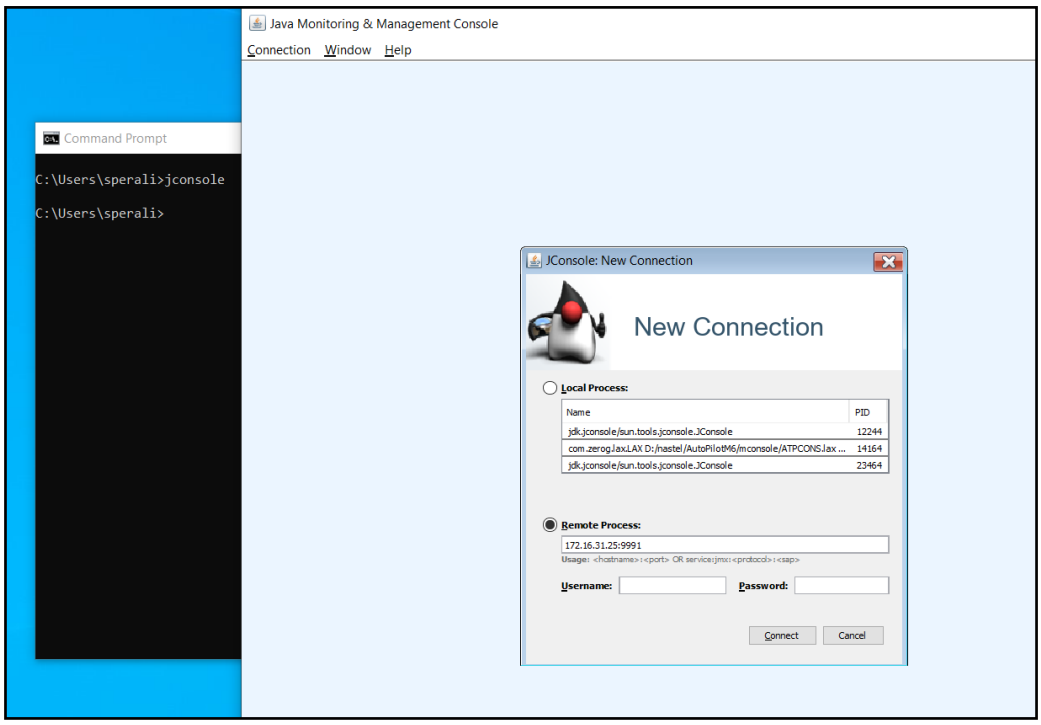


Figure 3-1. Connecting to jconsole

Once connected, switch to mbeans tab, to make sure metrics are seen.

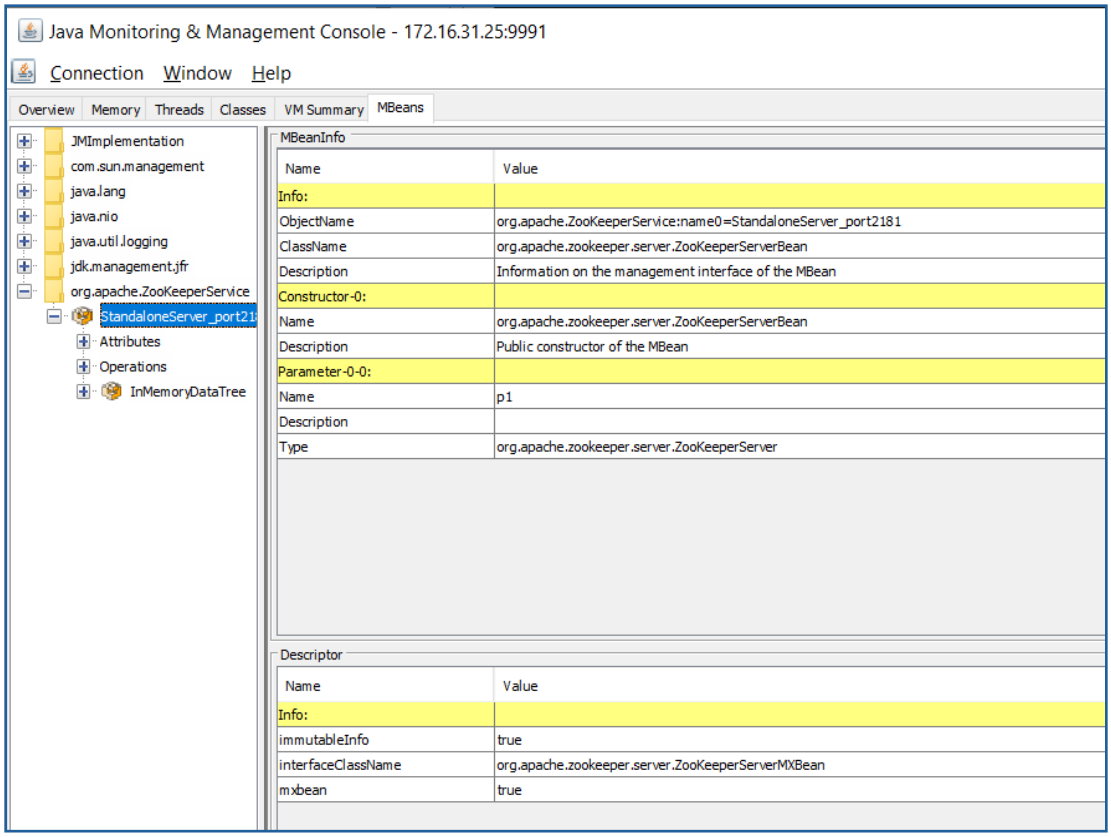


Figure 3-2. Zookeeper Metrics through jconsole

3.2.3.1 Configuring Apache Kafka with SSL

Configuring Apache Kafka with SSL involves several steps to ensure secure communication between clients and brokers.

Step 1: Generate SSL Certificates

1. Create a Certificate Authority (CA):

```bash
openssl genrsa -out kafkaCA.key 2048
```

```
> openssl genrsa -out kafkaCA.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
....+++++
..............................+++++
e is 65537 (0x010001)
```

```
openssl req -x509 -new -nodes -key kafkaCA.key -sha256 -days 365 -out kafkaCA.pem
```

```
>  openssl req -x509 -new -nodes -key kafkaCA.key -sha256 -days 365 -out kafkaCA.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:NY
Locality Name (eg, city) [Default City]:
Organization Name (eg, company) [Default Company Ltd]:meshiq
Organizational Unit Name (eg, section) []:SE TS
Common Name (eg, your name or your server's hostname) []:nasksc
Email Address []:support@meshiq.com
>  ll
total 8
drwxrwxr-x.  2 nastel nastel   44 Oct 24 21:07 .
drwxr-xr-x. 10 nastel nastel  148 Oct 24 20:57 ..
-rw-------.  1 nastel nastel 1679 Oct 24 20:58 kafkaCA.key
-rw-rw-r--.  1 nastel nastel 1424 Oct 24 21:07 kafkaCA.pem
```

```

2. Generate a Key and Certificate for Kafka Broker:

```bash

openssl genrsa -out kafkaBroker.key 2048

```
> openssl genrsa -out kafkaBroker.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
....
....+++++
.........+++++
e is 65537 (0x010001)
```

openssl req -new -key kafkaBroker.key -out kafkaBroker.csr

```
>  openssl req -new -key kafkaBroker.key -out kafkaBroker.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:us
State or Province Name (full name) []:ny
Locality Name (eg, city) [Default City]:
Organization Name (eg, company) [Default Company Ltd]:meshiq
Organizational Unit Name (eg, section) []:SE TS
Common Name (eg, your name or your server's hostname) []:nasksc
Email Address []:support@meshiq.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:nastel
An optional company name []:meshiq
```

openssl x509 -req -in kafkaBroker.csr -CA kafkaCA.pem -CAkey kafkaCA.key -CAcreateserial -out kafkaBroker.crt -days 365

```
>  openssl x509 -req -in kafkaBroker.csr -CA kafkaCA.pem -CAkey kafkaCA.key -CAcreateserial -out kafkaBroker.crt -days 365
Signature ok
subject=C = us, ST = ny, L = Default City, O = meshiq, OU = SE TS, CN = nasksc, emailAddress = support@meshiq.com
Getting CA Private Key
>
```

Firefox

```
```

3. Create a Keystore and Truststore:

```bash

openssl pkcs12 -export -in kafkaBroker.crt -inkey kafkaBroker.key -out kafkaBroker.p12 -name kafkaBroker -CAfile kafkaCA.pem -caname root
```

```
>  openssl pkcs12 -export -in kafkaBroker.crt -inkey kafkaBroker.key -out kafkaBroker.p12 -name kafkaBroker -CAfile kafkaCA.pem -caname root
Enter Export Password:
Verifying - Enter Export Password:
```

> ll

total 28

drwxrwxr-x. 2 meshiq meshiq 155 Oct 25 12:04 .

drwxr-xr-x. 9 meshiq meshiq 137 Oct 25 11:55 ..

-rw-rw-r--. 1 meshiq meshiq 1338 Oct 25 12:02 kafkaBroker.crt

-rw-rw-r--. 1 meshiq meshiq 1127 Oct 25 12:01 kafkaBroker.csr

-rw-------. 1 meshiq meshiq 1679 Oct 25 11:58 kafkaBroker.key

-rw-------. 1 meshiq meshiq 2628 Oct 25 12:04 kafkaBroker.p12

-rw-------. 1 meshiq meshiq 1679 Oct 25 11:56 kafkaCA.key

-rw-rw-r--. 1 meshiq meshiq 1460 Oct 25 11:58 kafkaCA.pem

-rw-rw-r--. 1 meshiq meshiq   41 Oct 25 12:02 kafkaCA.srlkeytool -importkeystore -srckeystore kafkaBroker.p12 -destkeystore kafkaBroker.keystore.jks -srcstoretype PKCS12

```
>  keytool -importkeystore -srckeystore kafkaBroker.p12 -destkeystore kafkaBroker.keystore.jks -srcstoretype PKCS12
Importing keystore kafkaBroker.p12 to kafkaBroker.keystore.jks...
Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias kafkabroker successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard format using "keytool -importkeyst
ore -srckeystore kafkaBroker.keystore.jks -destkeystore kafkaBroker.keystore.jks -deststoretype pkcs12".
```

```
>  keytool -importkeystore -srckeystore kafkaBroker.keystore.jks -destkeystore kafkaBroker.keystore.jks -deststoretype pkcs12
Enter source keystore password:
Entry for alias kafkabroker successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled

Warning:
Migrated "kafkaBroker.keystore.jks" to Non JKS/JCEKS. The JKS keystore is backed up as "kafkaBroker.keystore.jks.old".
>  ll
total 40
drwxrwxr-x.  2 nastel nastel 4096 Oct 24 21:18 .
drwxr-xr-x. 10 nastel nastel  148 Oct 24 20:57 ..
-rw-rw-r--.  1 nastel nastel 1302 Oct 24 21:12 kafkaBroker.crt
-rw-rw-r--.  1 nastel nastel 1110 Oct 24 21:11 kafkaBroker.csr
-rw-------.  1 nastel nastel 1675 Oct 24 21:09 kafkaBroker.key
-rw-rw-r--.  1 nastel nastel 2633 Oct 24 21:18 kafkaBroker.keystore.jks
-rw-rw-r--.  1 nastel nastel 2278 Oct 24 21:18 kafkaBroker.keystore.jks.old
-rw-------.  1 nastel nastel 2604 Oct 24 21:14 kafkaBroker.p12
-rw-------.  1 nastel nastel 1679 Oct 24 20:58 kafkaCA.key
-rw-rw-r--.  1 nastel nastel 1424 Oct 24 21:07 kafkaCA.pem
-rw-rw-r--.  1 nastel nastel   41 Oct 24 21:12 kafkaCA.srl
```

keytool -import -trustcacerts -file kafkaCA.pem -alias kafkaCA -keystore kafkaBroker.truststore.jks

<pre>> keytool -import -trustcacerts -file kafkaCA.pem -alias kafkaCA -keystore kafkaBroker.truststore.jks

Enter keystore password:

Re-enter new password:

Owner: EMAILADDRESS=support@meshiq.com, CN=nasksc, OU=SE TS, O=meshiq, L=Default City, ST=NY, C=US

Issuer: EMAILADDRESS=support@meshiq.com, CN=nasksc, OU=SE TS, O=meshiq, L=Default City, ST=NY, C=US

Serial number: 39cc5cfbd5391827aaf5068f184f03737ead6186

Valid from: Thu Oct 24 21:07:48 CEST 2024 until: Fri Oct 24 21:07:48 CEST 2025

Certificate fingerprints:

MD5: 46:01:FB:C7:C4:82:F1:11:FE:B0:DB:BA:EC:AD:9D:32

SHA1: 92:D1:DD:4E:3E:D8:D4:FF:5C:6B:47:14:10:90:0C:D8:79:A6:A6:7E

SHA256: 20:C0:E2:86:47:70:0B:33:75:FD:77:1C:84:1A:50:C0:94:4E:1C:B7:B6:EE:40:FA:B5:AF:FC:07:2D:FB:5A:4C

Signature algorithm name: SHA256withRSA

Subject Public Key Algorithm: 2048-bit RSA key

Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false

AuthorityKeyIdentifier [

KeyIdentifier [

0000: D5 A6 85 1A DA F5 CB 1F 4F 70 35 EC 15 EB BF 12 ....Op5.....

0010: 9B D7 65 AF ..e.

]

]

#2: ObjectId: 2.5.29.19 Criticality=true

BasicConstraints:[

CA:true

PathLen:2147483647

]

#3: ObjectId: 2.5.29.14 Criticality=false

SubjectKeyIdentifier [

KeyIdentifier [

0000: D5 A6 85 1A DA F5 CB 1F 4F 70 35 EC 15 EB BF 12 ....Op5.....

0010: 9B D7 65 AF ..e.

]

]

Trust this certificate? [no]: yes

Certificate was added to keystore

</pre>

```

## Step 2: Configure Kafka Broker

Edit the `server.properties` file in your Kafka configuration directory to include:

```properties

listeners=PLAINTEXT://:9092,SSL://:9093

listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL

advertised.listeners=PLAINTEXT://<broker-ip>:9092,SSL://<broker-ip>:9093

ssl.keystore.location=/path/to/kafkaBroker.keystore.jks

ssl.keystore.password=<keystore-password>

ssl.key.password=<key-password>

ssl.truststore.location=/path/to/kafkaBroker.truststore.jks

ssl.truststore.password=<truststore-password>

```

nasks

listeners=PLAINTEXT://:9092,SSL://:9993

listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL

advertised.listeners=PLAINTEXT://192.168.117.139:9092,SSL://192.168.117.139:9993

ssl.keystore.location=/opt/kafka/confluent-7.5.3/etc/ssl/kafkaBroker.keystore.jks

ssl.keystore.password=meshiq

ssl.key.password=meshiq

ssl.truststore.location=/opt/kafka/confluent-7.5.3/etc/ssl/kafkaBroker.truststore.jks

ssl.truststore.password=meshIQ

nasks2

############################## Socket Server Settings
##############################
```

listeners=PLAINTEXT://:9092,SSL://:9993

listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL

advertised.listeners=PLAINTEXT://nasksc2:9092,SSL://nasksc2:9993

ssl.keystore.location=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.keystore.jks

ssl.keystore.password=meshiq

ssl.key.password=meshiqd>

ssl.truststore.location=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.truststore.jks

ssl.truststore.password=meshiq

1. Step2.1 Kafka broker JMX SSL settings

*For Example, SSL props in **kafka-run-class***

KAFKA_JMX_OPTS="$KAFKA_JMX_OPTS /

-Dcom.sun.management.jmxremote=true /

-Dcom.sun.management.jmxremote.authenticate=false /

-Dcom.sun.management.jmxremote.ssl=true /

-Djavax.net.ssl.keyStore=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.keystore.jks /

-Djavax.net.ssl.keyStorePassword=meshiq /

-Djavax.net.ssl.trustStore=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.truststore.jks /

-Djavax.net.ssl.trustStorePassword=meshiq"/

JMX port to use

if [$**JMX_PORT**]; then

KAFKA_JMX_OPTS="$KAFKA_JMX_OPTS -
Dcom.sun.management.jmxremote.port=$**JMX_PORT** "

fi

snippet where JMX_PORT can be set is in script provided by meshIQ KAFKA_START.sh

export JMX_PORT=9994

Kafka broker runtime

> ps -ef | grep server-2

Meshiq123 190252 1775 2 08:30 pts/1 00:04:15 /opt/java/jdk1.8.0_202/bin/java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent -XX:MaxInlineLevel=15 -Djava.awt.headless=true -Xloggc:/opt/kafka/confluent-6.1.1/bin/../logs/kafkaServer-gc.log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps

-XX:+PrintGCTimeStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=100M

-Dcom.sun.management.jmxremote=true

-Dcom.sun.management.jmxremote.authenticate=false

-Dcom.sun.management.jmxremote.ssl=false

-Djava.rmi.server.hostname=nasksc2

-Dcom.sun.management.jmxremote.port=9994

-Djavax.net.ssl.keyStore=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.keystore.jks

-Djavax.net.ssl.keyStorePassword=meshiq

-Djavax.net.ssl.trustStore=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.truststore.jks

-Djavax.net.ssl.trustStorePassword=meshiq

-Dkafka.logs.dir=/opt/kafka/confluent-6.1.1/bin/../logs

-Dlog4j.configuration=file:bin/../etc/kafka/log4j.properties

-cp /opt/kafka/confluent-6.1.1/bin/../share/java/kafka/*:/opt/kafka/confluent-6.1.1/bin/../share/java/confluent-telemetry/* kafka.Kafka etc/kafka/server-2.properties

Step 3: Configure Kafka Client

1. Create Client Keystore and Truststore:

Similar to the broker, create a keystore and truststore for your client application.

2. Configure Client Properties:

For a Java client, configure the properties:

```properties
security.protocol=SSL
ssl.truststore.location=/path/to/client.truststore.jks
ssl.truststore.password=<truststore-password>
ssl.keystore.location=/path/to/client.keystore.jks
ssl.keystore.password=<keystore-password>
ssl.key.password=<key-password>
```

Step 4: Restart Kafka Broker

After making changes to the `server.properties`, restart the Kafka broker for the changes to take effect.

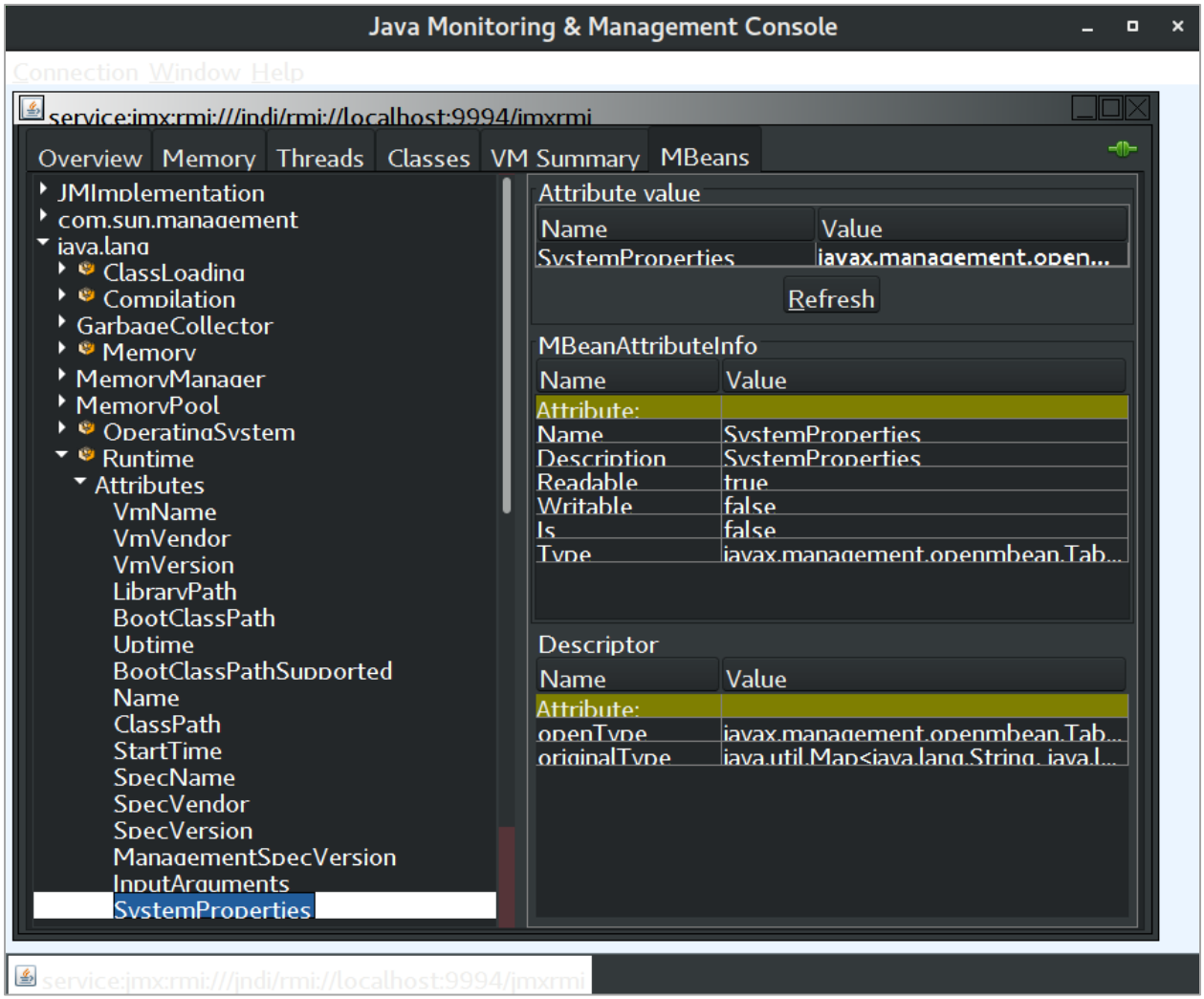
Step 5: Testing SSL Connection

You can test the SSL connection by using a Kafka client and trying to produce and consume messages.

==========jconsole ==========

jconsole -J-Djavax.net.ssl.keyStore=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.keystore.jks -J-Djavax.net.ssl.keyStorePassword=meshiq -J-Djavax.net.ssl.trustStore=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.truststore.jks -J-Djavax.net.ssl.trustStorePassword=meshiq

========== jconsole ==========



Troubleshooting Tips

- Ensure that the broker and client clocks are synchronized.

- Check logs for any SSL handshake errors.

- Use tools like `openssl s_client` to debug SSL connections.

By following these steps, you should have a secure Kafka setup using SSL.

3.2.3.2 Setting up a Kafka client with SSL

Step 1: Generate SSL Certificates

If you haven't already, you'll need to create a keystore and truststore for your client, similar to the Kafka broker setup.

1. Generate a Key and Certificate for the Client:

```bash

openssl genrsa -out client.key 2048
```

```
/opt/nastel/ssl
> openssl genrsa -out client.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................................+++++
.....................................................................................................+++++
e is 65537 (0x010001)
```

openssl req -new -key client.key -out client.csr

```
> openssl req -new -key client.key -out client.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:NY
Locality Name (eg, city) [Default City]:
Organization Name (eg, company) [Default Company Ltd]:
Organizational Unit Name (eg, section) []:SETS
Common Name (eg, your name or your server's hostname) []:nasdom
Email Address []:support@meshiq.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:nastel
An optional company name []:meshiq
```

openssl x509 -req -in client.csr -CA kafkaCA.pem -CAkey kafkaCA.key -CAcreateserial -out client.crt -days 365

```
> ll
total 8
-rw-rw-r--. 1 nastel nastel 1127 Oct 25 16:20 client.csr
-rw-------. 1 nastel nastel 1675 Oct 25 16:18 client.key
drwxrwxr-x. 2 nastel nastel  221 Oct 25 16:30 nasksc2_certs
> ll nasksc2_certs/
total 36
-rw-rw-r--. 1 nastel nastel 1338 Oct 25 12:02 kafkaBroker.crt
-rw-rw-r--. 1 nastel nastel 1127 Oct 25 12:01 kafkaBroker.csr
-rw-rw-r--. 1 nastel nastel 1679 Oct 25 11:58 kafkaBroker.key
-rw-rw-r--. 1 nastel nastel 2306 Oct 25 12:07 kafkaBroker.keystore.jks
-rw-rw-r--. 1 nastel nastel 2628 Oct 25 12:04 kafkaBroker.p12
-rw-rw-r--. 1 nastel nastel 1101 Oct 25 12:11 kafkaBroker.truststore.jks
-rw-rw-r--. 1 nastel nastel 1679 Oct 25 11:56 kafkaCA.key
-rw-rw-r--. 1 nastel nastel 1460 Oct 25 11:58 kafkaCA.pem
-rw-rw-r--. 1 nastel nastel   41 Oct 25 12:02 kafkaCA.srl
> openssl x509 -req -in client.csr -CA nasksc2_certs/kafkaCA.pem -CAkey nasksc2_certs/kafkaCA.key -CAcreateserial -out clie
nt.crt -days 365
Signature ok
subject=C = US, ST = NY, L = Default City, O = Default Company Ltd, OU = SETS, CN = nasdom, emailAddress = support@meshiq.co
m
Getting CA Private Key
>
```

```

2. Create a Keystore and Truststore:

```bash

openssl pkcs12 -export -in client.crt -inkey client.key -out client.p12 -name kafkaClient -CAfile kafkaCA.pem -caname root
```
```

```
> openssl x509 -req -in client.csr -CA nasksc2_certs/kafkaCA.pem -CAkey nasksc2_certs/kafkaCA.key -CAcreateserial -out clie
nt.crt -days 365
Signature ok
subject=C = US, ST = NY, L = Default City, O = Default Company Ltd, OU = SETS, CN = nasdom, emailAddress = support@meshiq.co
m
Getting CA Private Key
> openssl pkcs12 -export -in client.crt -inkey client.key -out client.p12 -name kafkaClient -CAfile nasksc2_certs/kafkaCA.p
em -caname root
Enter Export Password:
Verifying - Enter Export Password:
```

```
> ll
total 16
-rw-rw-r--. 1 nastel nastel 1338 Oct 25 16:32 client.crt
-rw-rw-r--. 1 nastel nastel 1127 Oct 25 16:20 client.csr
-rw-------. 1 nastel nastel 1675 Oct 25 16:18 client.key
-rw-------. 1 nastel nastel 2628 Oct 25 16:34 client.p12
drwxrwxr-x. 2 nastel nastel  221 Oct 25 16:30 nasksc2_certs
>
```

keytool -importkeystore -srckeystore client.p12 -destkeystore client.keystore.jks -srcstoretype PKCS12

```
> keytool -importkeystore -srckeystore client.p12 -destkeystore client.keystore.jks -srcstoretype PKCS12
Importing keystore client.p12 to client.keystore.jks...
Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias kafkaclient successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled
> ll
total 20
-rw-rw-r--. 1 nastel nastel 1338 Oct 25 16:32 client.crt
-rw-rw-r--. 1 nastel nastel 1127 Oct 25 16:20 client.csr
-rw-------. 1 nastel nastel 1675 Oct 25 16:18 client.key
-rw-rw-r--. 1 nastel nastel 2657 Oct 25 16:37 client.keystore.jks
-rw-------. 1 nastel nastel 2628 Oct 25 16:34 client.p12
drwxrwxr-x. 2 nastel nastel  221 Oct 25 16:30 nasksc2_certs
```

keytool -import -trustcacerts -file kafkaCA.pem -alias kafkaCA -keystore client.truststore.jks

> keytool -import -trustcacerts -file nasksc2_certs/kafkaCA.pem -alias kafkaCA -keystore client.truststore.jks

Enter keystore password:
Re-enter new password:
Owner: EMAILADDRESS=support@meshiq.com, CN=nasksc2, OU=SETS, O=Default Company Ltd, L=Default City, ST=NY, C=US
Issuer: EMAILADDRESS=support@meshiq.com, CN=nasksc2, OU=SETS, O=Default Company Ltd, L=Default City, ST=NY, C=US
Serial number: 645fa4dbeee1991dbe78fa116f8a7ef8a98461de
Valid from: Fri Oct 25 11:58:16 EDT 2024 until: Sat Oct 25 11:58:16 EDT 2025
Certificate fingerprints:
SHA1: 0C:9F:99:38:21:49:06:A3:6B:B6:8B:0E:13:AA:35:19:53:7F:A7:0E
SHA256: EB:6B:6E:70:2E:0C:8A:EB:E8:1A:75:5F:B8:3D:1D:C5:C1:23:4C:6E:FF:BA:27:E4:14:E0:D9:0B:BA:95:DA:C8
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: B0 4F C6 53 67 C2 84 37 CC 15 A4 DC DF 28 53 F6 .O.Sg..7.....(S.
0010: 18 C0 F4 7E   ....
]

```
]

#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: B0 4F C6 53 67 C2 84 37   CC 15 A4 DC DF 28 53 F6  .O.Sg..7.....(S.
0010: 18 C0 F4 7E                                        ....
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
```

Step 2: Configure the Kafka Client

For a Java Kafka client, you need to set properties for SSL connection.

Configuration File Example

Alternatively, you can use a configuration file (e.g., `client.properties`):

```properties
bootstrap.servers=<broker-ip>:9093
security.protocol=SSL
ssl.truststore.location=/path/to/client.truststore.jks
ssl.truststore.password=<truststore-password>
ssl.keystore.location=/path/to/client.keystore.jks
ssl.keystore.password=<keystore-password>
ssl.key.password=<key-password>
```

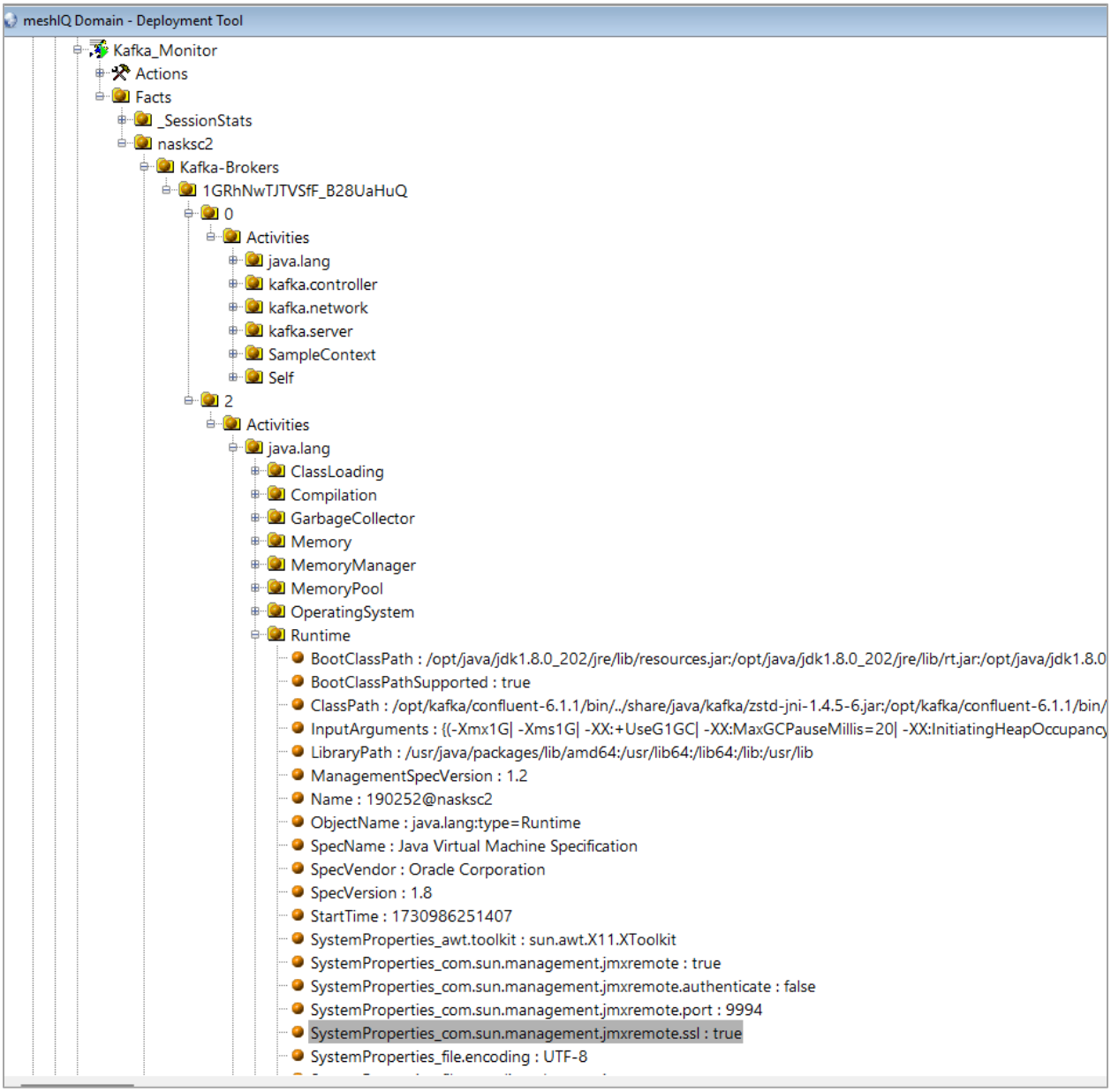
nasdom *stream-jmx start sh script*

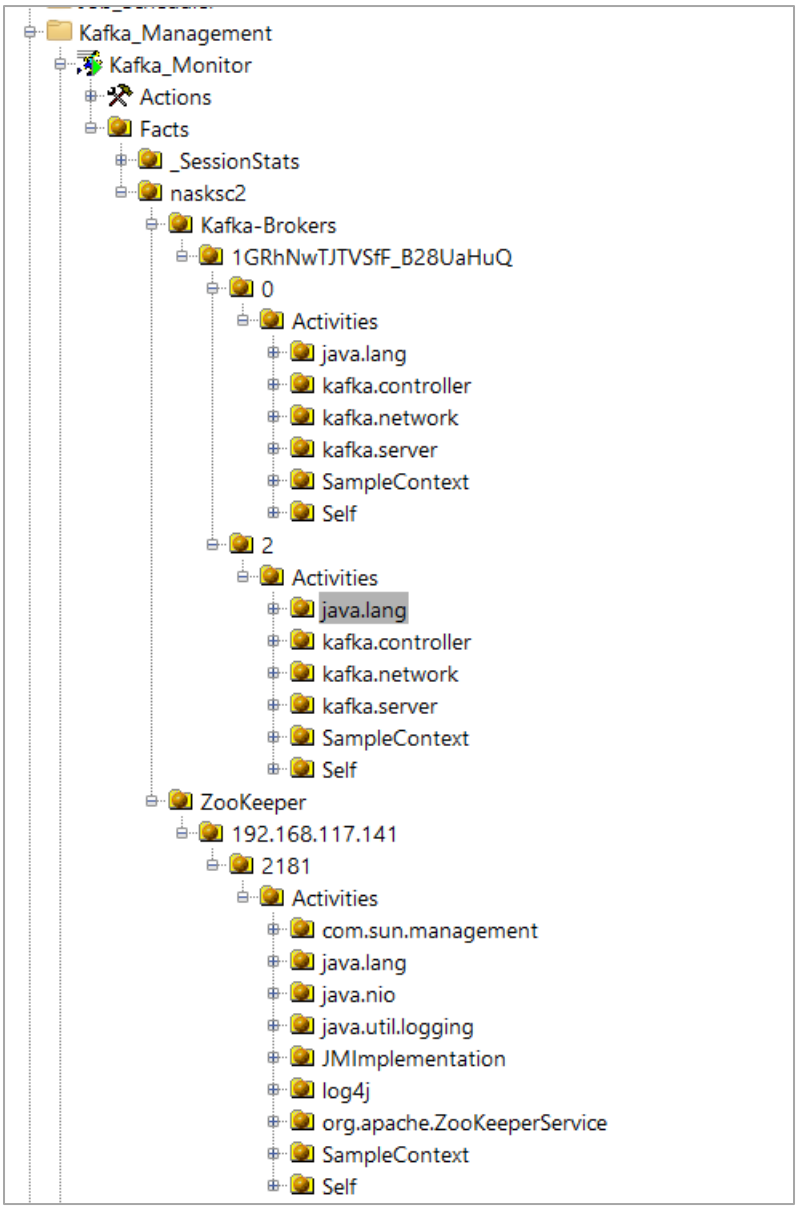
```
TNT4JOPTS="$TNT4JOPTS -Djavax.net.ssl.keyStore=/opt/nastel/ssl/client.keystore.jks\
-Djavax.net.ssl.keyStorePassword=mypswd\
-Djavax.net.ssl.trustStore=/opt/nastel/ssl/client.truststore.jks\
-Djavax.net.ssl.trustStorePassword=mypswd\
-Djavax.net.ssl.trustStoreType=jks\
-Djavax.net.ssl.keyStoreType=jks"

### stream JMX java runtime env
>  ps -ef | grep jmx
```

meshiq 195162 130980 0 09:14 pts/2 00:00:00 vim tnt4j-stream-jmx_KAFKA_nasksc2.log

meshiq 199052 199036 99 12:27 pts/1 00:00:26 /opt/nastel/java/jdk-11.0.2/bin/java -Dtnt4j.dump.on.vm.shutdown=true -Dtnt4j.dump.on.exception=true -Dtnt4j.dump.provider.default=true -Dtnt4j.config=/opt/tnt4j-stream-jmx/tnt4j-stream-jmx-1.16.0/run/../config/tnt4j.KAFKA.nasksc2.properties -Dlog4j2.configurationFile=file:/opt/tnt4j-stream-jmx/tnt4j-stream-jmx-1.16.0/run/../config/log4j2.xml -Dfile.encoding=UTF-8 -Dsjmx.serviceId=KAFKA_nasksc2 -server -Xms1G -Xmx2G -XX:MaxDirectMemorySize=1G -XX:+UseG1GC -XX:MaxGCPauseMillis=200 -XX:InitiatingHeapOccupancyPercent=60
-Djavax.net.ssl.keyStore=/opt/nastel/ssl/client.keystore.jks
-Djavax.net.ssl.keyStorePassword=meshiq
-Djavax.net.ssl.trustStore=/opt/nastel/ssl/client.truststore.jks
-Djavax.net.ssl.trustStorePassword=meshiq
-Djavax.net.ssl.trustStoreType=jks
-Djavax.net.ssl.keyStoreType=jks
-classpath /opt/tnt4j-stream-jmx/tnt4j-stream-jmx-1.16.0/run/../opt/tnt4j-stream-jmx-zk-1.16.0-all.jar:/opt/nastel/java/jdk-11.0.2/lib/tools.jar com.jkoolcloud.tnt4j.stream.jmx.SamplingAgent -connect -f:../config/connectionsStanzaZK_KAFKA.nasksc2.cfg -slp:compositeDelimiter=_

Step 3: Kafka JMX with SSL metrics monitor via meshIQ platform




Kafka_Management
Kafka_Monitor
Actions
Facts
_SessionStats
nasksc2
Kafka-Brokers
1GRhNwTJTVSfF_B28UaHuQ
0
Activities
java.lang
kafka.controller
kafka.network
kafka.server
SampleContext
Self
2
Activities
java.lang
kafka.controller
kafka.network
kafka.server
SampleContext
Self
ZooKeeper
192.168.117.141
2181
Activities
com.sun.management
java.lang
java.nio
java.util.logging
JMImplementation
log4j
org.apache.ZooKeeperService
SampleContext
Self

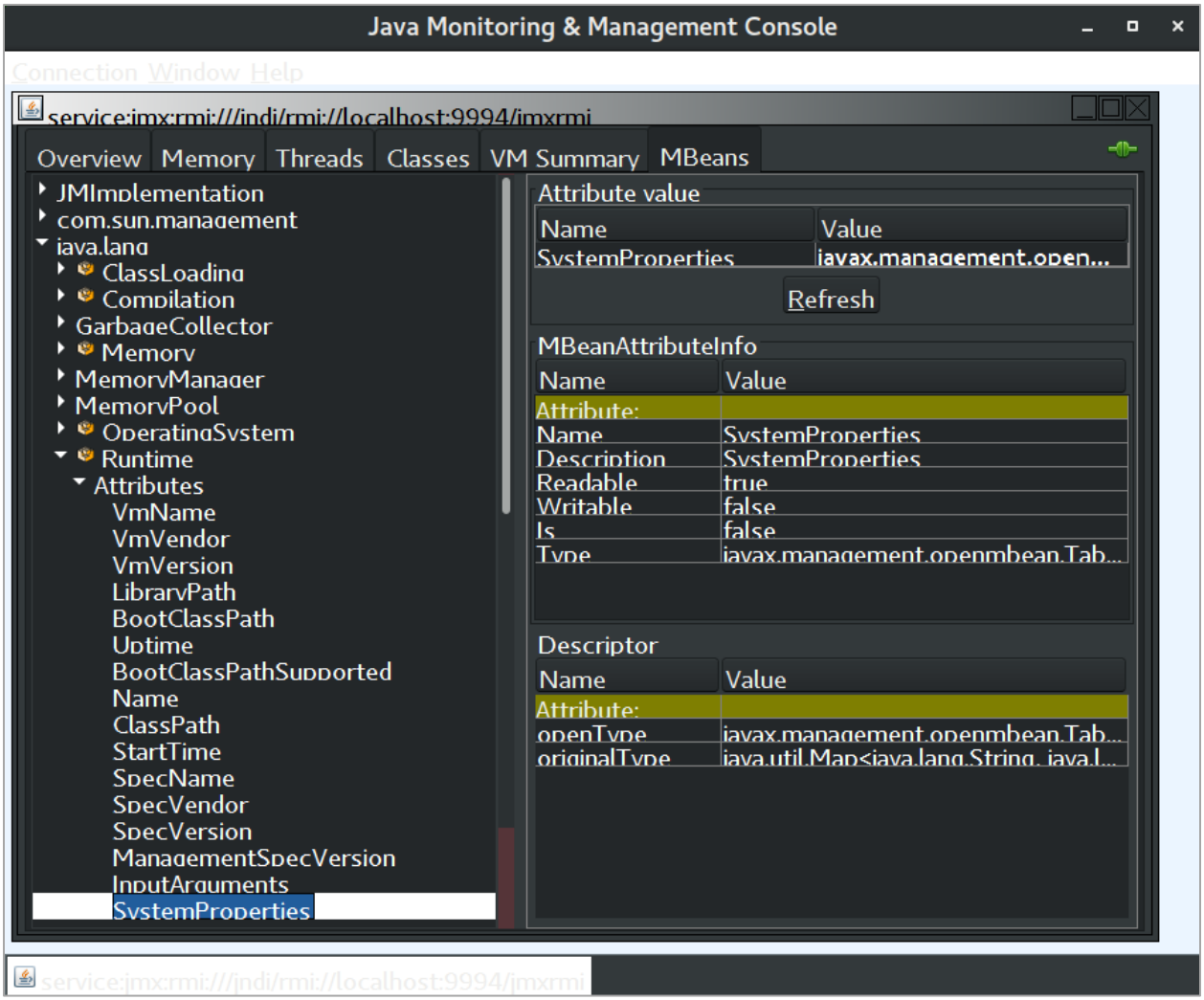
Step 4: Test the Setup

Run your client application and check the logs to ensure that it connects successfully to the Kafka broker using SSL.

==========jconsole ==========

jconsole -J-Djavax.net.ssl.keyStore=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.keystore.jks -J-Djavax.net.ssl.keyStorePassword=meshiq -J-Djavax.net.ssl.trustStore=/opt/kafka/confluent-6.1.1/ssl/kafkaBroker.truststore.jks -J-Djavax.net.ssl.trustStorePassword=meshiq

========== jconsole ==========



Troubleshooting Tips

-- Enable traces for log appender in log4j2.xml:

```
<Loggers>
    <!-- ### jKool LLC TNT4J Stream-JMX log level and appenders used -->
    <AsyncRoot level="TRACE">
        <AppenderRef ref="console"/>
        <AppenderRef ref="file"/>
    </AsyncRoot>
    <!-- ### change to DEBUG to have all the sampling process logged as it was
         ### using "-Dcom.jkoolcloud.tnt4j.stream.jmx.agent.trace=true" property. -->
    <AsyncLogger name="com.jkoolcloud.tnt4j.stream.jmx" level="TRACE"/>
    <!-- #### streamed activity entities logger #### -->
    <AsyncLogger name="com.jkoolcloud.tnt4j.stream.jmx.activities_log" level="TRACE" additivity="false">
        <AppenderRef ref="activities_log"/>
    </AsyncLogger>
    <!-- #### disable other APIs logging into Stream-JMX log #### -->
    <!-- <asyncLogger name="com.jkoolcloud.tnt4j.sink.impl.slf4" level="NONE"/> -->
</Loggers>
```

- Ensure the truststore contains the CA certificate.

- Verify that the keystore contains the client certificate and key.

- Check for any SSL handshake errors in the logs.

- Ensure that the broker is correctly configured to accept SSL connections.

By following these steps, your Kafka client should be securely set up with SSL.

3.2.3Deploying Process Wrapper

Process wrapper deployed on CEP runs on a port, and all the data/metrics/information sent over this port are published as facts. Stream JMX sends the collected metrics to AP using the same method. It is preferred to use separate process wrappers to monitor & manage metrics of additional Kafka components. For example, Kafka_Monitor process wrapper is for Zookeeper, Broker & Connect components, while Confluent_Monitor process wrapper is for Schema registry, KSQL, and Kafka Rest Proxy components.

To Deploy Process Wrapper:

1. Right-click on CEP -> Deploy Expert -> Wrappers -> Select Process Wrappers.
2. In the create window, under General tab, set the Context field to Kafka_Management and Name to Kafka_Monitor.
3. Navigate to Fact options tab in create window and set the Expire Facts (ms) value to 60,0000 (10 mins).
4. Navigate to TCP Options tab in the create window, enable the publishing of TCP facts, and define a port on which process wrapper runs (port should not be in use already).
5. Deploy the expert, and once deployed, it is visible under CEP in deployment view.

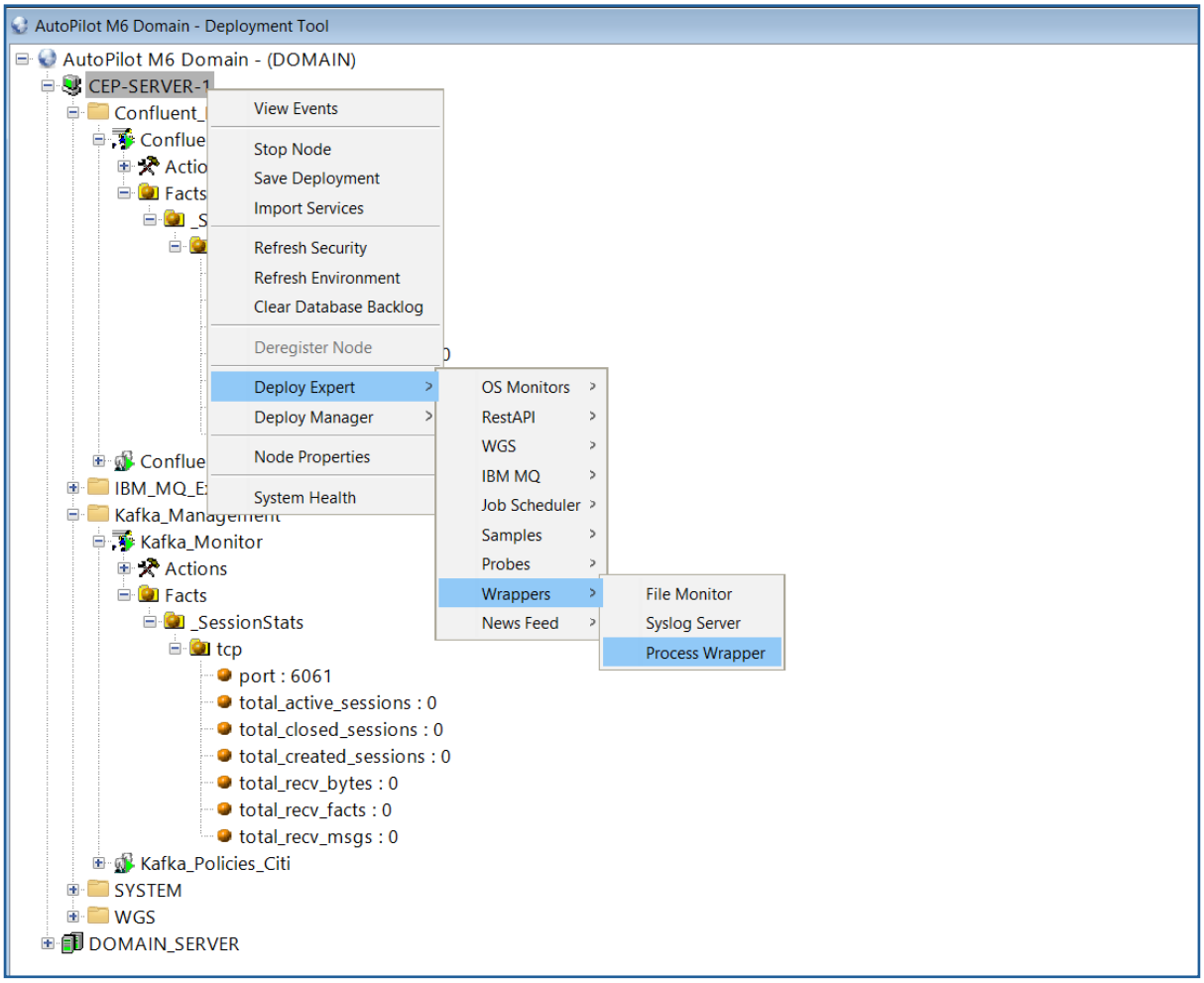


Figure 3-3. Deploy Process Wrapper

For only minor configuration changes you can edit the monitor properties as follows:

1. Open the AutoPilot Console.
2. Click the Deployment Tool  to display Directory Viewer (if not already displayed).
3. Right-click Kafka_Monitor and select Properties.

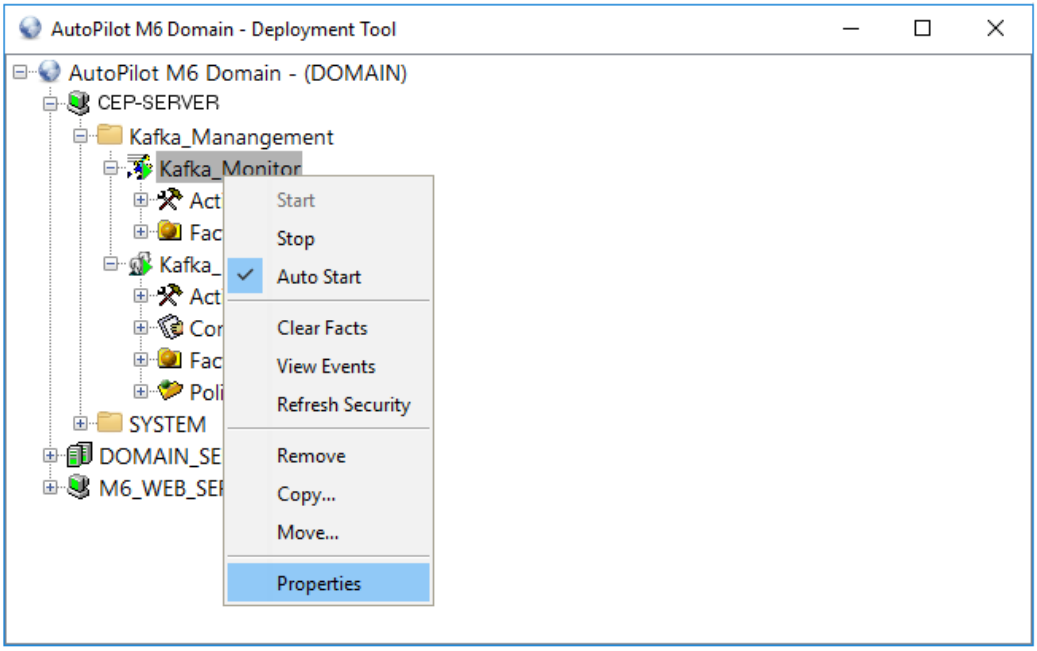


Figure 3-4. Modify Kafka Monitor

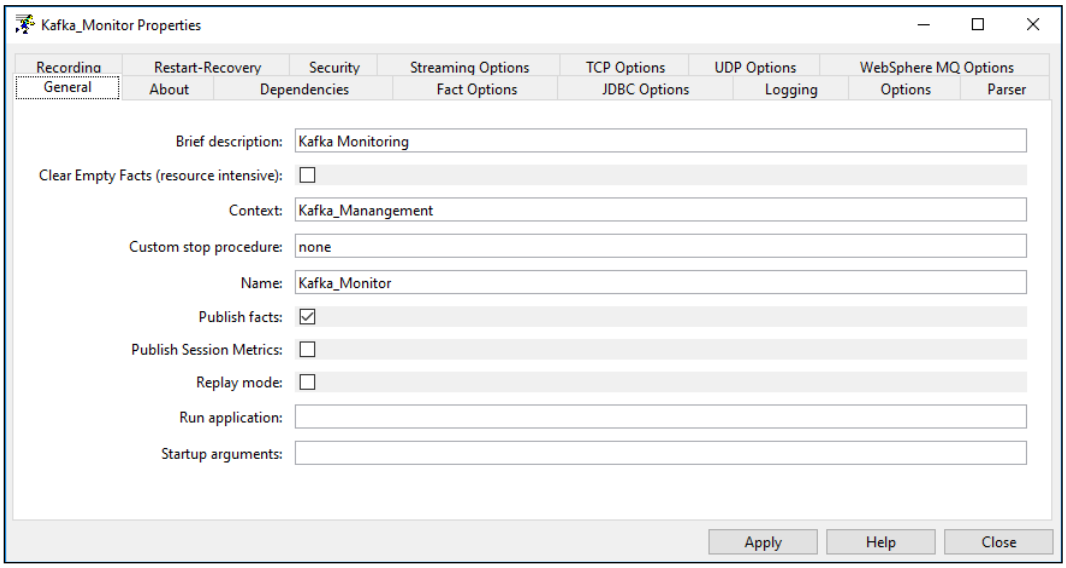


Figure 3-5. Create Kafka Monitor – General Tab

4. On the **General** tab, the following fields could be changed as required. Other parameters are not applicable to Kafka monitoring.

Table 3-1. Kafka Monitor – General Properties

Property	Description
Brief description	Short description of the service.
Context	User-defined category that will be registered in the Domain Server. Context is displayed as folder icon under each Managed Node.
Name	Name that uniquely identifies the service in the Domain Server. Enter or modify the Service Name as required, or in accordance with local guidelines. Variations of names are used when deploying services on multiple Nodes. No spaces or blanks are recommended in Service Name formats. For example, KAFKA_Monitor.

5. The following properties are available for the Kafka expert. Review (if updating existing expert) or configure data elements as follows.

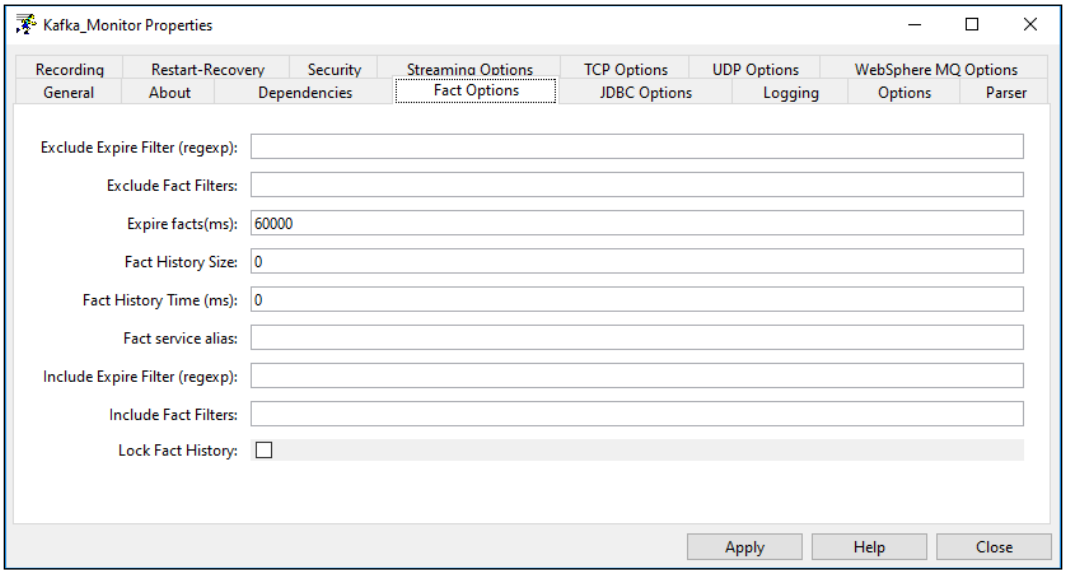


Figure 3-6. Create Kafka Monitor – Fact Options Tab

Table 3-2. Kafka Monitor – Fact Options Properties

Property	Description
Exclude Expire Filter (regexp)	Facts that match the specified regular expression are not expired.
Exclude Fact Filters	Comma separated list of fact paths to exclude during publishing. For example: *SYSTEM*, *FactName*
Expire facts(ms)	User-defined time in which facts that have not been updated within a specific time automatically expire (in milliseconds). The default is 0, which means never expire. However, in most applications, 0 should not be used. In cases where certain data is no longer published, if 0 is used, these facts will never expire. It is recommended that this value be 50% larger than the sample rate.
Fact History Size	Automatically maintains the specified number of samples for each published fact in memory.
Fact History Time	Automatically maintain fact history not exceeding specified time in milliseconds.
Include Fact Filters	Comma separated list of fact paths to include during publishing. For example: *SYSTEM*, *FactName*
Fact service alias	If supported by the expert, specifies the alternative service name that the expert will publish its facts under.
Include Expire Filter (regexp)	Facts that match the specified regular expression are expired.
Lock Fact History	Enables/disables history collection after accumulating the first history batch up to **Fact History Time** or **Fact History Size** which ever limit is reached first. If disabled newer history samples replace older on a rolling basis.

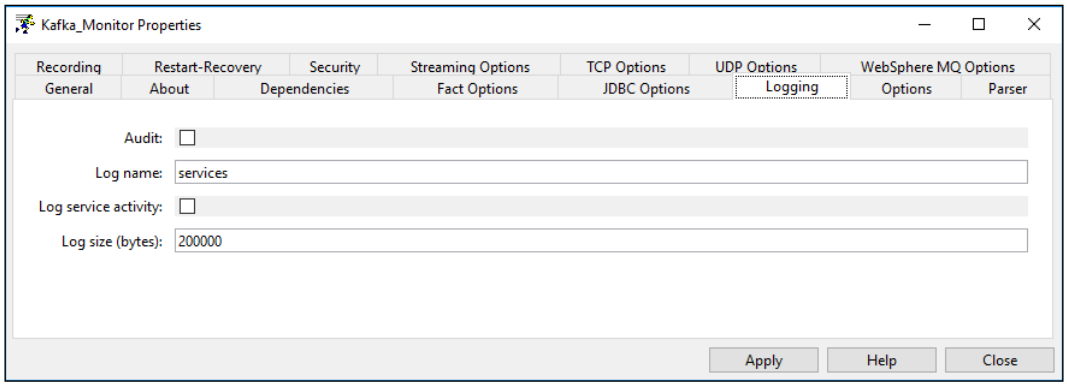


Figure 3-7. Create Kafka Monitor – Logging Tab

Table 3-3. Kafka Monitor – Logging Properties	
Property	**Description**
Audit	Enable/Disable service audit trace. Default is disabled.
Log name	Log name associated with the service. The default name is Services, but may be changed as required.
Log service activity	Enable/Disable service activity trace. Default is disabled.
Log size (bytes)	Log size in bytes. Real log size is the maximum value of the server.log.size and logsize.

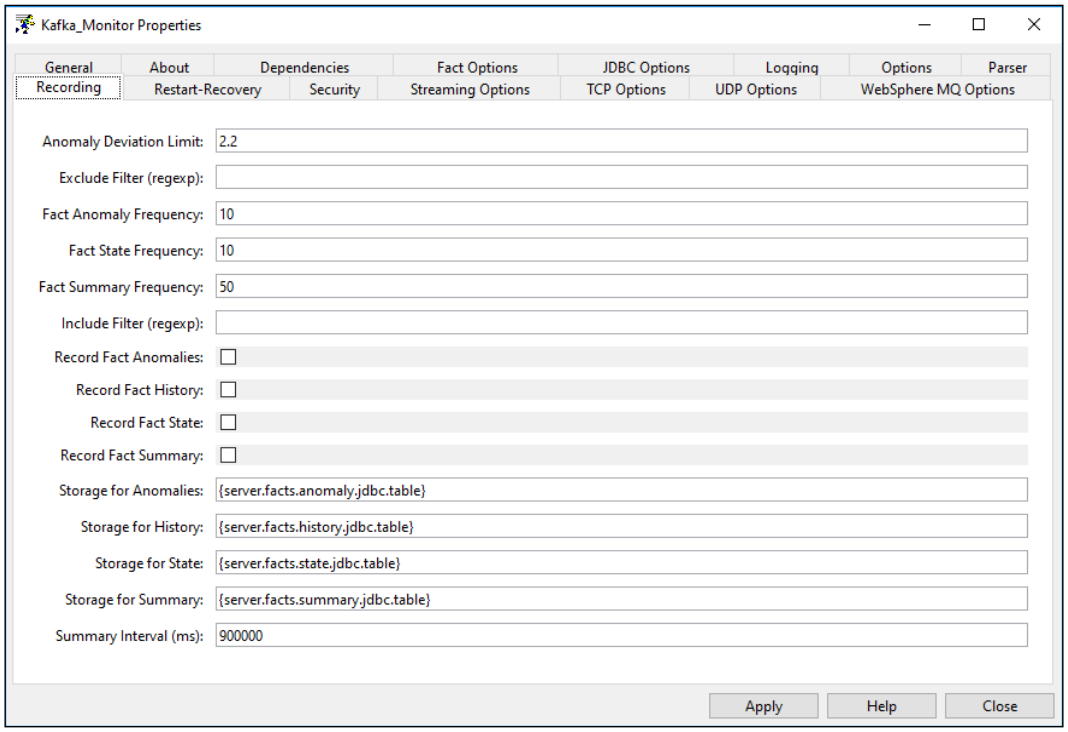


Figure 3-8. Create Kafka Monitor – Recording Tab

Table 3-4. Kafka Monitor – Recording Properties	
Property	**Description**
Anomaly Deviation Limit	The number of standard deviations from the mean at which the value is considered an anomaly. For example, a value of 2.2 is 2.2 standard deviations. Requires fact recording to be configured (although not actually recording).
Exclude Filter (regexp)	A regular expression filter to exclude certain facts from being written to the database. Facts have the format expert\class\instance\leaf=value such as in the example Servers\Linux\Serv7\processes=40.
Fact Anomaly Frequency	The frequency of fact updates at which anomaly calculation is done. A value of 10 indicates every 10th sample. A value of 1 would analyze every fact update to determine if it was an anomaly.
Fact State Frequency	If **Record Fact State** is enabled, the value entered here specifies how often the Fact State is updated.
Fact Summary Frequency	If **Record Fact Summary** is enabled, used to write an intermediate summary record every Xth update to the fact during the Summary Interval. In this example, every 50th update to the fact an intermediate summary record is recorded. This is done to avoid waiting 15 minutes for a summary record to appear in the summary table.
Include Filter (regexp)	A regular expression filter to include certain facts being written to the database. Same format as described for the exclude filter.
Record Fact Anomalies	If enabled, records every fact anomaly into the Anomaly database. The exclude/include filters are respected. Requires fact recording to be configured (although not actually recording).
Record Fact History	If enabled, records every fact change into the History database. The exclude/include filters are respected. To define database tables and set AutoPilot options, refer to *AutoPilot M6 User's Guide* section 4.5.4.1.

Table 3-4. Kafka Monitor – Recording Properties	
Property	**Description**
Record Fact State	If enabled, records the last value published (current state) into the state database and restores that value when the CEP Server is stopped and restarted. The exclude/include filters are respected. To define database tables and set AutoPilot options, refer to AutoPilot M6 User's Guide, section 4.5.4.1.
Record Fact Summary	If enabled, records summary record at the interval designated in the Summary Interval (ms) field into the Summary database. The exclude/include filters are respected. To define database tables and set AutoPilot options, refer to AutoPilot M6 User's Guide, section 4.5.4.1.
Storage for Anomalies	Database table where the Fact Anomalies data is stored.
Storage for History	Database table where the Fact History data is stored.
Storage for State	Database table where the Fact State data is stored.
Storage for Summary	Database table where the Fact Summary data is stored.
Summary Interval (ms)	If Record Fact Summary is enabled, designates the interval of time in ms for which baseline numbers for each numeric fact are computed. Summary Interval is only in affect when CEP instance is running in record mode (ATPNODE –record). Default 900000 is 15 minutes, which means maintain a baseline of statistics for each numeric fact for a period of 15 minutes and write a record to the database. At the end of interval fact statistics is reset and the baseline collection starts again.

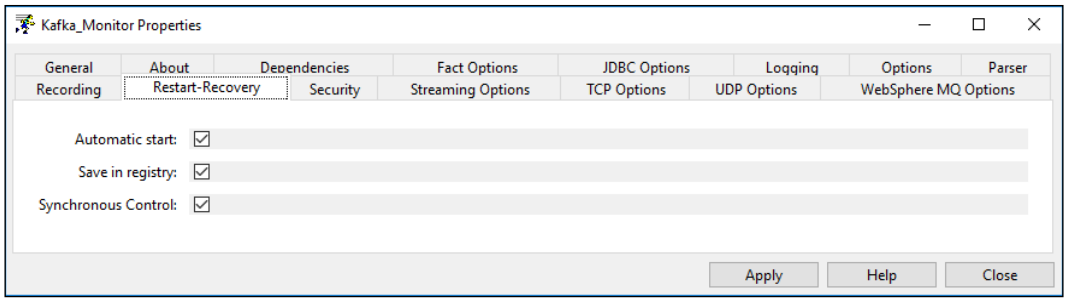


Figure 3-9. Create Kafka Monitor – Restart-Recovery Tab

Table 3-5. Kafka Monitor – Restart-Recovery Properties	
Property	**Description**
Automatic start	Enable/disable automatic start.
Save in registry	Persistent services are saved in Registry.xml file. Default is enabled.
Synchronous Control	Enable/Disable synchronous service initiation.

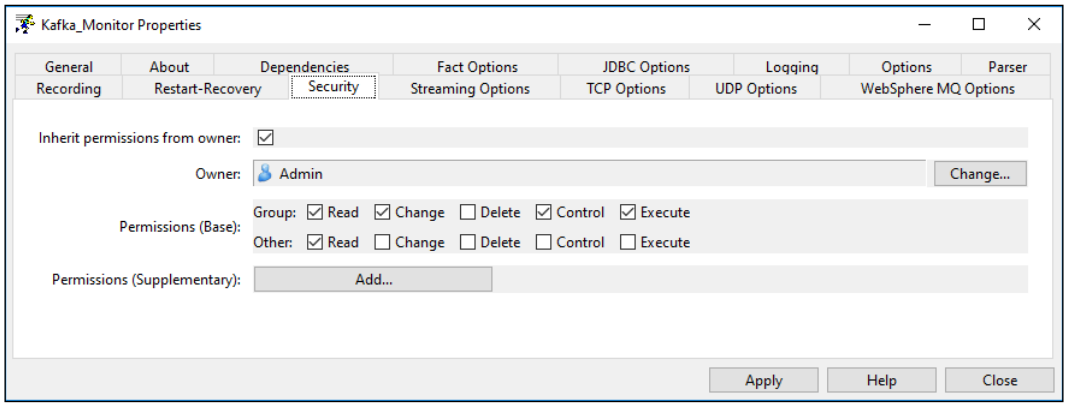


Figure 3-10. Create Kafka Monitor – Security Tab

Table 3-6. Kafka Monitor – Security Properties

Property	Description	
Inherit permissions from owner	Enable/disable inherit permission from owner's permission masks. Default is enabled.	
Owner	User that owns the object.	
Permissions	Permissions for users in the same group and users in other groups. Enable/disable as required.	
	Group:	**Others:**
Read	Group members may read/view attributes of an object.	Other users may read/view attributes of an object.
Change	Group members may change the attributes of an object.	Other users may change the attributes of an object.
Delete	Group members may delete the object.	Other users may delete the object.
Control	Group members may execute control actions such as start, stop, and disable.	Other users may execute control actions such as start, stop, and disable.
Execute	Group members may execute operational commands on the object.	Other users may execute operational commands on the object.

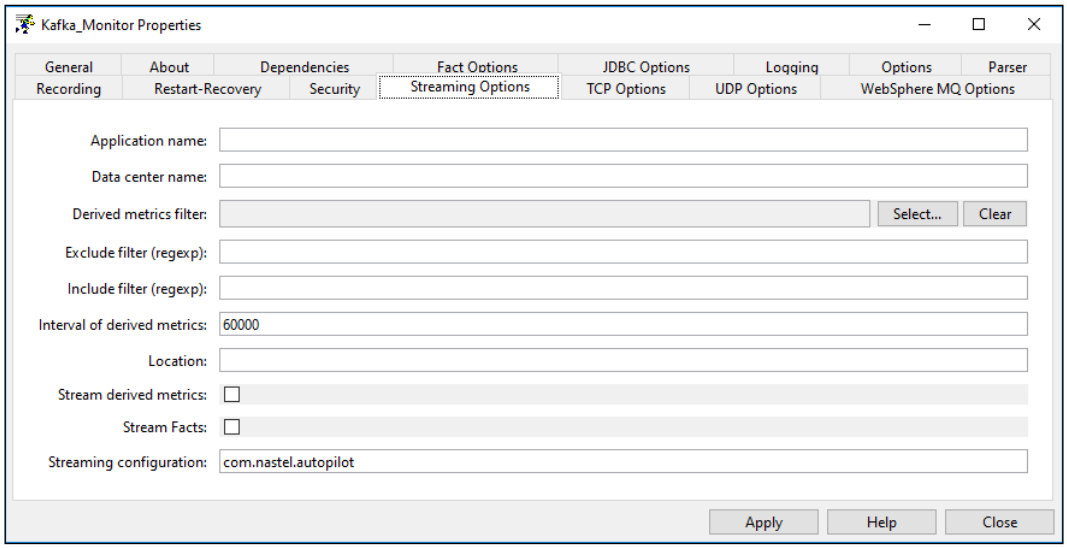


Figure 3-11. Create Kafka Monitor – Streaming Options Tab

Table 3-7. Kafka Monitor – Streaming Options Properties	
Application name	Sets application name if different from the default set in the **tnt4j.properties** file.
Data center name	Sets data center name if different from the default set in the **tnt4j.properties** file.
Derived metrics filter	Create or select filter.
Exclude filter (regexp)	Ignore facts that match specified regular expression; that is, do not stream facts that match the regexp.
Include filter (regexp)	Only stream the facts that match specified regular expression.
Interval of derived metrics	Set interval.
Location	Sets server location if different from the default set in the **tnt4j.properties** file.
Stream derived metrics	Enable/disable derived metrics streaming.
Stream Facts	Enable/disable fact streaming (requires TNT4J streaming framework).
Streaming configuration	Indicates where the data streams. This value must match a stanza in the **tnt4j.properties** file. The default is **com.meshIQ.autopilot**.

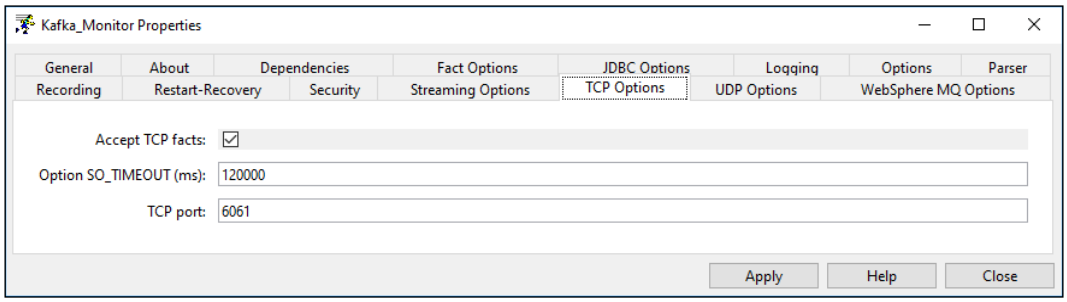


Figure 3-12. Create Kafka Monitor – TCP Options Tab

TCP is a reliable data connection to ensure facts will be published, but there is a slight performance hit due to the extra networking overhead required.

1. Click *TCP Options* tab, to enable an M6 Process Wrapper to receive TCP data.
2. Check *Accept TCP Facts* checkbox ☑ , and then enter a port. The port will also have to be specified in the application sending the fact data.

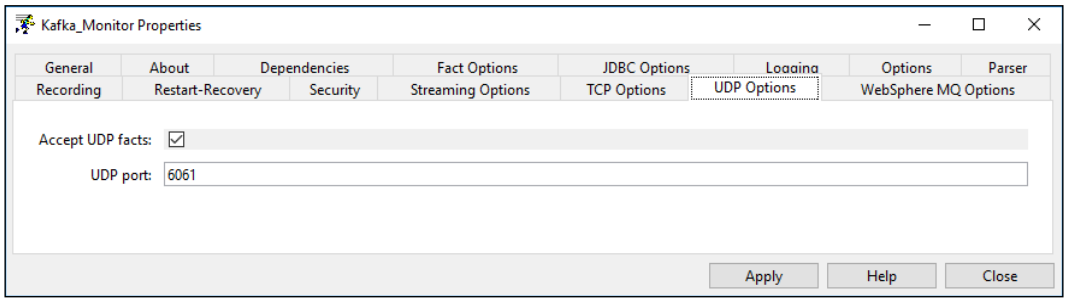


Figure 3-13. Create Kafka Monitor – UDP Options Tab

UDP is a less reliable data connection used mainly for speed and application-to-application decoupling. While this is the fastest protocol with the least amount of network overhead, there is potential that data could be lost since there is no acknowledgment/hand shaking between the sender and receiver.

1. Click *UDP Options* tab to enable an M6 process wrapper to receive UDP data.
2. Check *Accept UDP Facts* ☑ checkbox and then enter a port. The port will also have to be specified in the application sending the fact data.

3.3 Installation

This section provides instructions for installing Kafka Expert on the compatible platforms. Review all pre-requisites materials prior to commencing installation procedures. Reviewing materials will allow installers to familiarize themselves with associated requirements.

Installation Steps:

1. Download the Kafka monitoring package from the meshIQ ftp site.
2. On the VM, navigate to $APIN_HOME (for example, /opt/meshIQ) and create a new directory named tnt4j-stream-jmx.
3. Switch to the newly created directory and extract the monitoring package to this location - tar -xvf PackageName.tar.
4. Extract the policies shipped in the package to $AUTOPILOT_HOME/naming/policies directory on Domain servers - tar -xvf Kafka_Policies.tar & tar -xvf Confluent_Policies.tar.
5. Under CEP, deploy process wrappers as mentioned in *section 3.2.3*.
6. Create new policy managers under CEP to deploy the policies extracted on domain servers. Please refer to sections *5.1* & *5.2* for more details.
7. Finally, update configuration files with JMX connection strings and process wrapper details. The configuration files are located at $APIN_HOME/tnt4j-stream-jmx/current/config and the sample config file names are: connectionsStanzaZK_CONFLUENT.cfg,connectionsStanzaZK_KAFKA.cfg,tnt4j.CONFLUENT.properties & tnt4j.KAFKA.properties. Please refer to *section 3.4* for more information.

3.4 Configuration: Securing Kafka Server Components

Configuring Apache Kafka_2.13-3.5.1(Kraft mode) with SSL involves several steps to ensure secure communication between clients and brokers. Here's a concise guide:

1. In order to setup Kafka in the *Kraft mode* click here. (KRaft (Kafka Raft) mode in Apache Kafka is a self-managed metadata system that replaces the traditional dependency on **Apache ZooKeeper)**
2. To establish a secure connection between the broker and client, SSL certificates must be created on both sides.
3. Creating SSL certificates on the broker side involves the following steps
4. Create a directory to store all certificates using the CLI. Example: mkdir ssl-certs.
5. Create a file called **san.conf** using the command touch san.conf under the ssl-certs directory and enter the following details in it:

   [req_ext]
   subjectAltName = @alt_names
   [alt_names]
   DNS.0 = nastelin.com
   IP.0 = 172.16.31.72

Mention the required IP addresses.

6. Generate the certificates in sequence:

   - **Run the following command to generate the keystore: (keystore name and password can be set accordingly)**
     keytool -keystore kafka.server.keystore.jks -alias localhost -keyalg RSA -validity 1000 -genkey -storepass nastel -keypass nastel -ext SAN=DNS:nastelin.com,IP:172.16.31.72

   - **Generate the Certificate Authority (CA) certificate:**
     openssl req -new -x509 -keyout ca-key.key -out ca-cert.crt -days 1000

   - **Import the CA certificate into the truststore:**
     keytool -keystore kafka.server.truststore.jks -alias CARoot -importcert -file ca-cert.crt

   - **Generate the certificate signing request (CSR) for the server:**
     keytool -keystore kafka.server.keystore.jks -alias localhost -certreq -file cert-file-server.csr -ext SAN=DNS:nastelin.com,IP:172.16.31.72

   - **Sign the CSR with the CA certificate:**
     openssl x509 -req -CA ca-cert.crt -CAkey ca-key.key -in cert-file-server.csr -out cert-signed-server.crt -days 1000 -CAcreateserial -passin pass:nastel -extfile san.conf -extensions req_ext

   - **Import the CA certificate into the keystore:**
     keytool -keystore kafka.server.keystore.jks -alias CARoot -importcert -file ca-cert.crt

- **Import the signed server certificate into the keystore:**
  keytool -keystore kafka.server.keystore.jks -alias localhost -importcert -file cert-signed-server.crt

7. All the above certificates will appear as shown in the image below.

```
[nastel@QA-xray-1-5 kafka]$ cd ssl-certs
[nastel@QA-xray-1-5 ssl-certs]$ ll
total 32
-rw-r--r--. 1 nastel nastel 1411 Feb 18 13:23 ca-cert.crt
-rw-------. 1 nastel nastel 1854 Feb 18 13:22 ca-key.key
-rw-r--r--. 1 nastel nastel 1129 Feb 18 13:23 cert-file-server.csr
-rw-r--r--. 1 nastel nastel 1387 Feb 18 13:23 cert-signed-server.crt
-rw-r--r--. 1 nastel nastel 5230 Feb 18 13:24 kafka.server.keystore.jks
-rw-r--r--. 1 nastel nastel 1382 Feb 18 13:23 kafka.server.truststore.jks
-rw-r--r--. 1 nastel nastel   91 Feb 18 13:21 san.conf
[nastel@QA-xray-1-5 ssl-certs]$
```

Figure 3-14 Configured Certificates

8. Add the following details to the **server. properties** file present in **/opt/meshiq/kafka/kafka-2.13-3.5.1/config/kraft**

   listeners=SSL://:9093
   advertised.listeners=SSL://172.16.31.72:9093
   inter.broker.listener.name=SSL
   ssl.keystore.location=/opt/meshiq/kafka/kafka_2.13-3.5.1/ssl-certs/kafka.server.keystore.jks
   ssl.keystore.password=nastel
   ssl.key.password=nastel
   ssl.truststore.location=/opt/meshiq/kafka/kafka_2.13-3.5.1/ssl-certs/kafka.server.truststore.jks
   ssl.truststore.password=Nastel

NOTE These are some the important properties that should be mentioned in the server.properties file since we are configuring it with SSL.

9. In the scripts folder provided by meshiq, modify the **KAFKA_START.sh** script by adding the following lines:

   - WORKING_DIR=/opt/meshiq/kafka/kafka_2.13-3.5.1
     export JAVA_HOME=/opt/meshiq/java/current
     export JRE_HOME=/opt/meshiq/java/current
     export PATH=/opt/meshiq/java/current/bin:$PATH
   - export KAFKA_JMX_OPTS="-Dcom.sun.management.jmxremote=true -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=true -Djava.rmi.server.hostname=172.16.31.72 -Dcom.sun.management.jmxremote.authenticate=false -Djavax.net.ssl.keyStore=/opt/meshiq/kafka/kafka_2.13-3.5.1/ssl-certs/kafka.server.keystore.jks -Djavax.net.ssl.keyStorePassword=nastel -Djavax.net.ssl.trustStore=/opt/meshiq/kafka/kafka_2.13-3.5.1/ssl-certs/kafka.server.truststore.jks -Djavax.net.ssl.trustStorePassword=nastel -Dcom.sun.management.jmxremote.rmi.port=9993"
   - export JMX_PORT=9992

> **IMPORTANT!**
>
> After Ensure that the script has the correct working directory and Java path

10. After adding the above lines quickly save it and run the shell command:

    ./KAFKA_START.sh

11. To verify if the kafka has started and is running, execute the following command:
    ps -ef | grep properties
12. A series of lines can be seen, where SSL is enabled. Keystore paths and passwords are mentioned as shown in the image below.

Figure 3-15 SSL Enabled for Kafka Running

    These lines in the image above indicate that SSL is enabled for Kafka running in Kraft mode.
13. Once the broker side setup is done and is up and running, the client side properties can be set.

3.5 Configuration: Using TNT4J Stream-JMX for Kafka Monitoring

This section discusses the configuration files that need to be edited and the properties they include.

3.5.1 Configuring Client-Side Properties using SSL

Setting up a Kafka client with SSL involves configuring the client to securely communicate with the Kafka brokers. Follow the steps below:

1. Navigate to the ssl-certs directory which was created earlier. Since, we already have the san.conf file, we do not need to create another one.
2. One important thing to note is that the same ca-cert.crt and ca-key.key files are used to create the client keystore and truststore certificates.

   - **Generate the client keystore:**
     keytool -keystore kafka.client.keystore.jks -alias clientKeystore -validity 365 -genkey -keyalg RSA -ext SAN=DNS:nastelin.com,IP:172.16.31.72

   - **Create the client truststore by importing the CA certificate:**
     keytool -keystore kafka.client.truststore.jks -alias CARoot -importcert -file ca-cert.crt

   - **Generate the certificate request for the client:**
     keytool -keystore kafka.client.keystore.jks -alias clientKeystore -certreq -file cert-file

   - **Sign the certificate request with the CA:**
     openssl x509 -req -CA ca-cert.crt -CAkey ca-key.key -in cert-file -out cert-signed -days 365 -CAcreateserial -passin pass:meshiq -extfile san.conf -extensions req_ext
   - **Import the CA certificate into the keystore:**
     keytool -keystore kafka.client.keystore.jks -alias CARoot -importcert -file ca-cert

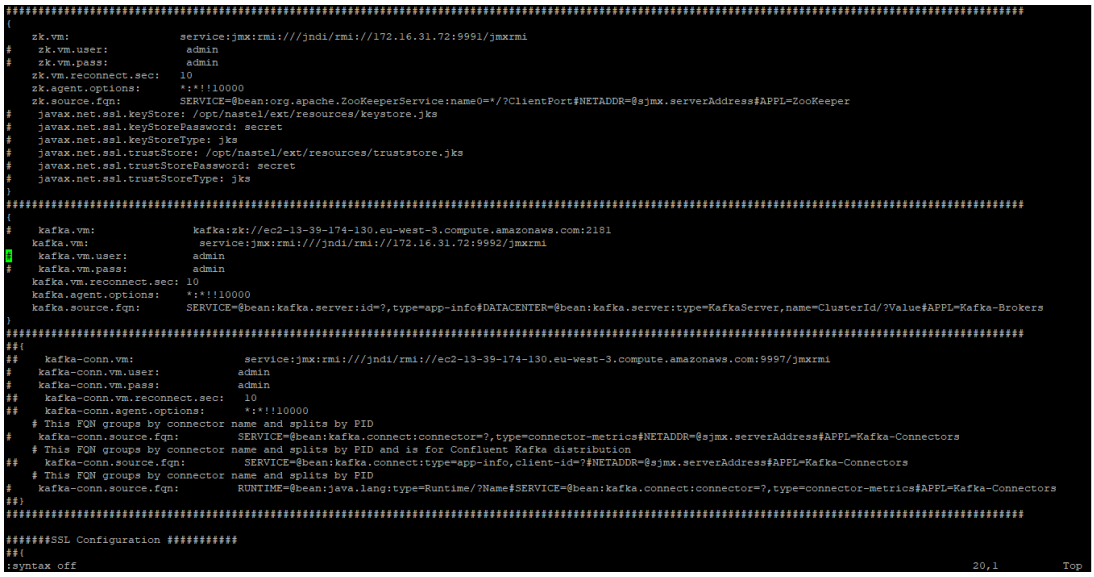
   - **Import the signed certificate into the keystore:**
     keytool -keystore kafka.client.keystore.jks -alias clientKeystore -importcert -file cert-signed

3. All the above created client certificates will appear as shown in the image below.

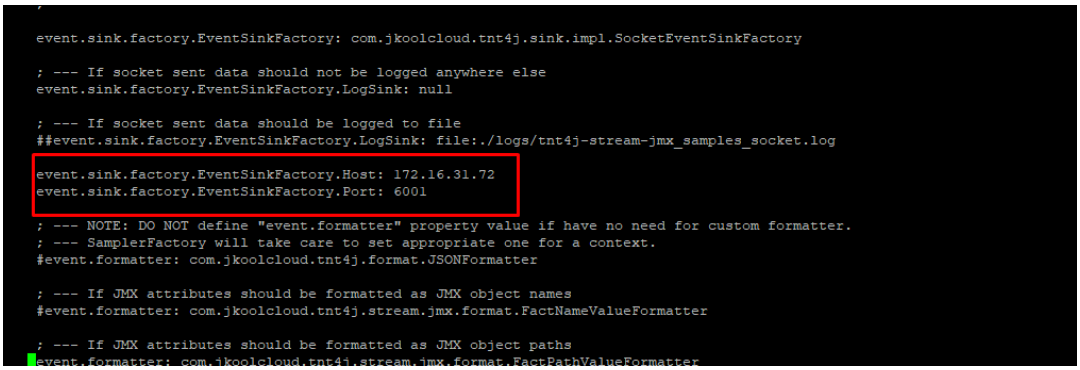
```
[nastel@QA-xray-1-5 confluent-6.1.1]$ cd clientssl
[nastel@QA-xray-1-5 clientssl]$ ll
total 32
-rw-r--r--. 1 nastel nastel 1411 Feb 20 14:56 ca-cert.crt
-rw-------. 1 nastel nastel 1854 Feb 20 14:59 ca-key.key
-rw-r--r--. 1 nastel nastel 1077 Feb 20 14:57 cert-file
-rw-r--r--. 1 nastel nastel 1379 Feb 20 14:59 cert-signed
-rw-r--r--. 1 nastel nastel 5240 Feb 20 15:00 kafka.client.keystore.jks
-rw-r--r--. 1 nastel nastel 1382 Feb 20 14:57 kafka.client.truststore.jks
-rw-r--r--. 1 nastel nastel   91 Feb 20 13:51 san.conf
[nastel@QA-xray-1-5 clientssl]$
```

Figure 3-16 Created Client Certificates

4. Ensure that the **connectionsStanzaZK_KAFKA.cfg** file has the correct configuration located in **/opt/meshiq/kafka/tnt4j-stream-jmx/current/config**



5. In **tnt4j.KAFKA.properties** located in **/opt/meshiq/kafka/tnt4j-stream-jmx/current/config**, make sure to provide the correct Host and Port.



6. **stream-jmx-connect-file-config.sh** file should be executed in order to collect the JMX metrics.

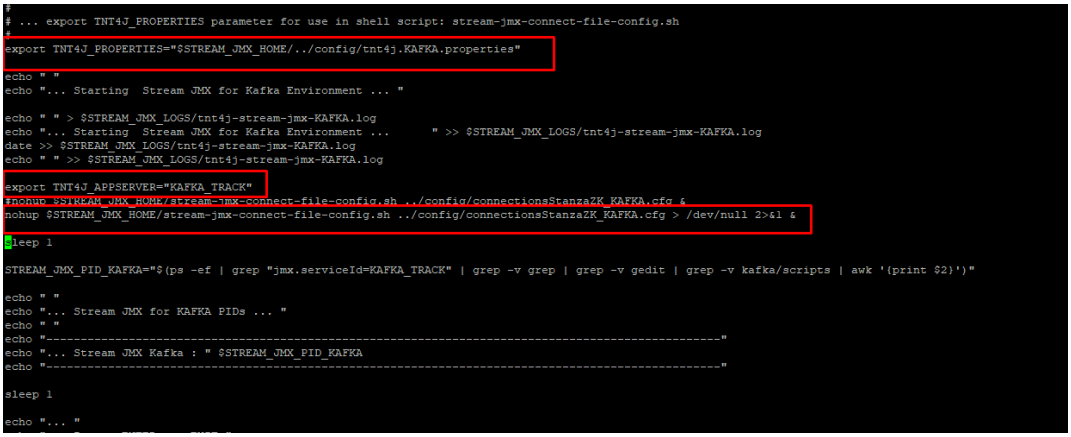
7. This file can be found under tnt4j-stream-jmx-VERSION-SNAPSHOT\run folder, also download the STREAM_JMX_SCRIPTS from the ftp site.
8. In order to execute the stream jmx config file firstly, add the following details to the **stream-jmx-connect-file-config.sh** file present in tnt4j-stream-jmx-VERSION-SNAPSHOT\run:
   TNT4JOPTS="$TNT4JOPTS
   TNT4JOPTS="$TNT4JOPTS
   -Djavax.net.ssl.keyStore=/opt/meshiq/kafka/Kafka-2.13-3.5.1/clientssl/kafka.client.keystore.jks /
   -Djavax.net.ssl.keyStorePassword=nastel /
   -Djavax.net.ssl.trustStore=/opt/meshiq/kafka/ Kafka-2.13-3.5.1/clientssl/kafka.client.truststore.jks /
   -Djavax.net.ssl.trustStorePassword=nastel /
   -Dcom.sun.management.jmxremote.port=9098 /
   -Dcom.sun.management.jmxremote.rmi.port=9099 /
   -Dcom.sun.management.jmxremote.authenticate=false /
   -Dcom.sun.management.jmxremote.ssl=false"

9. Ensure that the following paths are exported.

   - export TNT4J_PROPERTIES="$STREAM_JMX_HOME/../config/tnt4j.KAFKA.properties"
   - export TNT4J_APPSERVER="KAFKA_TRACK"

   - nohup $STREAM_JMX_HOME/stream-jmx-connect-file-config.sh ../config/connectionsStanzaZK_KAFKA.cfg &

     Add the following lines as shown in the screenshot below.



10. Since the kafka server is configured with RMI Registry SSL, add the following properties as well:
    - ### --- Uncomment if server side has enabled RMI registry SSL communication
      CONN_OPTIONS="$CONN_OPTIONS -cp:com.sun.jndi.rmi.factory.socket=javax.rmi.ssl.SslRMIClientSocketFactory"
      ### ----------
    - ### --- Uncomment when following required if client has Java v11 and server has java v17
      TNT4JOPTS="$TNT4JOPTS \
      -Djdk.tls.client.protocols=TLSv1.2,TLSv1.3\
      -Djdk.tls.namedGroups="secp256r1,secp384r1"\
      -Dcom.sun.net.ssl.checkRevocation=false
11. After entering all the details correctly, you can start the stream using one of the following methods, depending on your setup:
    Option 1: Run the following shell command to start streaming data with the standard stream-jmx-connect-file-config.sh script:
    nohup $STREAM_JMX_HOME/stream-jmx-connect-file-config.sh ../config/connectionsStanzaZK_KAFKA.cfg &
    This method uses the standard streams package setup. Ensure that all paths, properties, and certificate references are configured correctly before running this command.
    Option 2: Alternatively, if you're using custom build startup shell scripts provided in the STREAM_JMX_SCRIPTS folder, run:
    ./KAFKA_STREAM_JMX_START.sh
    This script starts the stream using a custom configuration, typically used in deployments where additional startup logic or environment setup is handled within KAFKA_STREAM_JMX_START.sh.

NOTE

Use only one of the above methods based on how your environment is structured.

12. Once the script is executed, check whether the JMX is running by using the command: ps -ef | grep jmx.

```
nastel     12165   12150  4 Mar13 pts/0    05:53:19 /opt/meshiq/java/current/bin/java -Dtnt4j.dump.on.vm.shutdown=true -Dtnt4j.dump.on.exception=true -Dtnt4j.dump.provi
der.default=true -Dtnt4j.config=/opt/meshiq/kafka/tnt4j-stream-jmx/current/run/../config/tnt4j.KAFKA.properties -Dlog4j2.configurationFile=file:/opt/meshiq/kafka/tnt4j-
stream-jmx/tnt4j-stream-jmx-1.18.6/run/../config/log4j2.xml -Dfile.encoding=UTF-8 -Dsjmx.serviceId=KAFKA_TRACK -Djavax.net.ssl.keyStore=/opt/meshiq/kafka/kafka_2.13-3.5
.1/ssl-certs/kafka.client.keystore.jks -Djavax.net.ssl.keyStorePassword=nastel -Djavax.net.ssl.trustStore=/opt/meshiq/kafka/kafka_2.13-3.5.1/ssl-certs/kafka.client.trus
tstore.jks -Djavax.net.ssl.trustStorePassword=nastel -classpath /opt/meshiq/kafka/tnt4j-stream-jmx/tnt4j-stream-jmx-1.18.6/run/../opt/tnt4j-stream-jmx-zk-1.18.6-all.jar
:/opt/meshiq/java/current/lib/tools.jar com.jkoolcloud.tnt4j.stream.jmx.SamplingAgent -connect -f:../config/connectionsStanzaZK_KAFKA.cfg -slp:compositeDelimiter=_
nastel   4064678    3748  0 18:35 pts/1    00:00:00 grep --color=auto jmx
```

13. Kafka JMX with SSL metrics can now be monitored via the meshIQ platform.

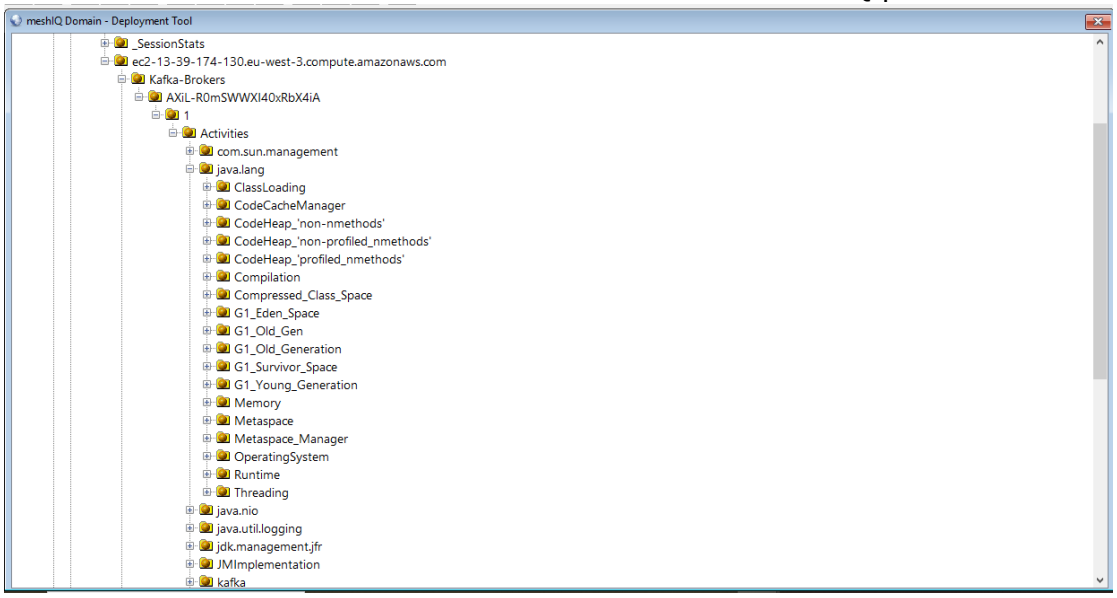


Figure 3-17 Kafka JMX with SSL Metrics

3.5.2Configure tnt4j.properties

This property file is used to define or store

- **Process wrapper information**: the host on which the process wrapper is deployed and the port it uses.
- **Root FQN**: the FQN to use when creating the fact tree (the parent directory created under the process wrapper).
- **Retry interval and attempts**: the retry interval and the number of attempts to retrieve the required MBean if the value is not received on the first try.

Update the properties listed below in the tnt4j*.properties file.

##

;Stanza used for Stream-JMX sources

##

{ source: com.jkoolcloud.tnt4j.stream.jmx

source.factory: com.jkoolcloud.tnt4j.stream.jmx.source.JMXSourceFactoryImpl

source.factory.GEOADDR: New York

source.factory.DATACENTER: HQDC

source.factory.SERVICE: $sjmx.serviceId ⇦ example1 for Kafka processID/service name

source.factory.SERVICE: @bean:kafka.server:type=app-info,id=? <- example2 for Kafka BrokerID

source.factory.SERVER: @sjmx.serverName or @sjmx.serverAddress ⇦example1 for Kafka node/server name

source.factory.SERVER: @bean:java.lang:type=Runtime/?Name ⇦example2 for Kafka node/server name

source.factory.SERVER: ClusterName ⇦ example3 for Kafka node/server name

; This RootFQN definition is for single VM monitoring

source.factory.RootFQN: SERVER=?

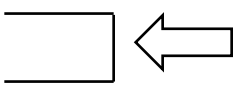
; This RootFQN definition is for multi VM monitoring: SERVICE token shall be defined next to VM

#source.factory.RootFQN: SERVER=?#DATACENTER=?

source.factory.RootSSN: tnt4j-stream-jmx

source.factory.RetryIntervalSec: 1

source.factory.MaxRetryAttempts: 60



Retry interval to request mbean

##

; Event Sink configuration for streaming name=value pairs over socket

##

event.sink.factory.EventSinkFactory: com.jkoolcloud.tnt4j.sink.impl.SocketEventSinkFactory

;--- If socket sent data should no be logged anywhere else

event.sink.factory.EventSinkFactory.LogSink: null

;--- If socket sent data should be logged to file

##event.sink.factory.EventSinkFactory.LogSink: file:./logs/tnt4j-stream-jmx_samples_socket.log

event.sink.factory.EventSinkFactory.Host: localhost ⇦ hostname of AutoPilot Kafka_Expert

event.sink.factory.EventSinkFactory.Port: 6061 ⇦ TCP port of AutoPilot Kafka_Expert

3.5.3ConnectionsStanzaZK_*.cfg

In the connectionStanzaZK_*.cfg file, JMX connection string information, and additional options that are required when publishing facts are included. Each connection stanza has a common set of properties, and these are differentiated by adding the component name before them.

- **vm**: Property vm is used to define the JMX connection string. As each Kafka component publishes metrics on its own JMX port, this should be defined separately for all of them.
- **vm.user**: Property vm.user is required, if a basic authentication is set to true when JMX is enabled on the Kafka component.
- **vm.pass**: This property is for defining the basic auth user password
- **vm.reconnect.sec**: vm.reconnect.sec property defines the time interval in which, the Stream-JMX process retries connection to the Kafka component if a previous attempt failed.
- **agent.options**: Through agent options, Stream-JMX allows users to define filtering, initial delay, sampling rate, and batch size. Please see *section 3.4.2.1* agent options for more information on syntax and how each of them is used.
- **source.fqn**: Source FQN is used to define the fact tree structure for each Kafka component. This property is read from right to left and the delimiter is #.

Additionally, if stream JMX is enabled with SSL=true, then the following properties should also be added and the values of each of them need to be defined.

- **javax.net.ssl.keyStore**: Path to the SSL keystore certificate.
- **javax.net.ssl.keyStorePassword**: SSL keystore password.
- **javax.net.ssl.keyStoreType**: SSL Keystore type (JKS or PEM or so on).
- **javax.net.ssl.trustStore**: Path to the SSL trust store certificate.
- **javax.net.ssl.trustStorePassword**: SSL trust store password.
- **javax.net.ssl.trustStoreType**: SSL trust store type (JKS or PEM or so on).

In the package, 2 sample files are already present at $STREAM_JMX_HOME/config/ directory and they are connectionsStanzaZK_KAFKA.cfg, connectionsStanzaZK_CONFLUENT.cfg. To have efficient processing, it is preferred to have separate config files for each cluster & separate config files for components belonging to each vendor.

3.5.3.1 JMX Sampling Agent (sampler) options

As defined in the previous section, the agent option allows applying filters and other required parameters, which are useful when the goal is to optimize the collection & publication processes. Agent options are defined using format is mbean-filter!exclude-filter!sample-ms!init-delay-ms!batch-size

- **mbean-filter:** MBean includes a name filter defined using object name pattern: domainName:keysSet.

NOTE

Multiple filters can be defined using; as delimiter (domainName1:keysSet1;domainName2:keysSet2;domainName3:keysSet3) or grouping using () and | notation. Grouping like java.lang:type=(Threading|Memory|OperatingSystem|Runtime) is equal to java.lang:type=Threading;java.lang:type=Memory;java.lang:type=OperatingSystem;java.lang:type=Runtime;

- **Exclude-filter:** MBean exclude name filter defined using object name pattern: domainName:keysSet.

NOTE

Multiple filters can be defined using; as delimiter (domainName1:keysSet1;domainName2:keysSet2;domainName3:keysSet3) or grouping using () and | notation. Grouping like java.lang:type=(Threading|Memory|OperatingSystem|Runtime) is equal to java.lang:type=Threading;java.lang:type=Memory;java.lang:type=OperatingSystem;java.lang:type=Runtime;

- **Sample-ms:** MBeans sampling rate in milliseconds.

NOTE

If MBean sampling duration gets longer than the configured sample-ms value, it will dynamically reschedule sampling from a fixed rate to a fixed delay in sample-ms between samples.

- **Init-delay-ms:** MBeans sampling initial delay in milliseconds. Optional, by default it is equal to sample-ms value.
- **Batch-size:** number of sampled MBeans to post over a single package. Optional, default is -1 (unlimited).

NOTE

If the current MBean sampling iteration takes longer than the system property com.jkoolcloud.tnt4j.stream.jmx.sampler.batch.period.sec defined value or used memory consumption percentage gets higher than system property com.jkoolcloud.tnt4j.stream.jmx.sampler.batch.used.memory.percent defined value, the current batch is also considered to be complete.

Default sampling agent options value is *:*!!30000!30000!-1

Sample usage of using agent options, so that Stream JMX only collects metrics required by policies is mentioned below.

- **Zookeeper Agent Options Sample:**

This example includes the mbeans filter for publishing only the Zookeeper metrics required by sample policies.

zk.agent.options: org.apache.ZooKeeperService:*!!60000!!10

- **Kafka Broker Agent Options Sample:**

This example includes the mbeans filter for publishing only the Kafka broker metrics required by sample policies.

kafka.agent.options: kafka.server:type=app-info,id=*;kafka.server:type=KafkaServer,name=BrokerState;kafka.server:type=KafkaServer,name=yammer-metrics-count;kafka.server:type=KafkaServer,name=ClusterId;kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec;kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec;kafka.server:type=BrokerTopicMetrics,name=BytesRejectedPerSec;kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec;kafka.server:type=BrokerTopicMetrics,name=TotalFetchRequestsPerSec;kafka.server:type=BrokerTopicMetrics,name=TotalProduceRequestsPerSec;kafka.server:type=BrokerTopicMetrics,name=FailedFetchRequestsPerSec;kafka.server:type=BrokerTopicMetrics,name=FailedProduceRequestsPerSec;kafka.server:type=socket-server-metrics,listener=*,networkProcessor=*;kafka.server:type=Fetch;kafka.server:type=Produce;kafka.server:type=ReplicaManager,name=LeaderCount;kafka.server:type=ReplicaManager,name=PartitionCount;kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions;kafka.server:type=ReplicaFetcherManager,name=MaxLag,clientId=*;kafka.server:type=ReplicaManager,name=IsrExpandsPerSec;kafka.server:type=ReplicaManager,name=IsrShrinksPerSec;kafka.server:type=ReplicaManager,name=ReassigningPartitions;kafka.server:type=ReplicaManager,name=OfflineReplicaCount;kafka.server:type=LeaderReplication;kafka.server:type=FetcherLagMetrics;kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Fetch;kafka.server:type=DelayedOperationPurgatory,name=NumDelayedOperations,delayedOperation=Fetch;kafka.server:type=SessionExpireListener,name=ZooKeeperAuthFailuresPerSec;kafka.server:type=SessionExpireListener,name=ZooKeeperDisconnectsPerSec;kafka.server:type=SessionExpireListener,name=ZooKeeperExpiresPerSec;kafka.server:type=SessionExpireListener,name=ZooKeeperReadOnlyConnectsPerSec;kafka.server:type=SessionExpireListener,name=ZooKeeperSaslAuthenticationsPerSec;kafka.server:type=SessionExpireListener,name=ZooKeeperSyncConnectsPerSec;kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent;kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent;kafka.network:type=Processor,name=IdlePercent,networkProcessor=0;kafka.network:type=RequestMetrics,name=*,request=Heartbeat;kafka.network:type=RequestMetrics,name=*,request=Heartbeat;kafka.network:type=RequestMetrics,name=*,request=Produce;kafka.network:type=RequestMetrics,name=*,request=Fetch;kafka.network:type=RequestMetrics,name=*,request=FetchConsumer;kafka.network:type=RequestMetrics,name=*,request=FetchFollower;kafka.network:type=RequestMetrics,name=*,request=LeaderAndIsr;kafka.network:type=RequestMetrics,name=*,request=GroupCoordinator;kafka.network:type=RequestMetrics,name=*,request=SyncGroup;kafka.network:type=RequestMetrics,name=*,request=JoinGroup;kafka.network:type=RequestMetrics,name=*,request=LeaveGroup;java.lang:type=Runtime;java.lang:type=OperatingSystem;java.lang:type=Memory;java.lang:name=G1 Young Generation,type=GarbageCollector;java.lang:type=Threading;kafka.controller:type=KafkaController,name=ActiveControllerCount;kafka.controller:type=KafkaController,name=OfflinePartitionsCount;kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs;kafka.controller:type=ControllerStats,name=UncleanLeaderElectionsPerSec;kafka.cluster:type=Partition,name=UnderReplicated,topic=*,partition=*;kafka.server:type=kafka-metrics-count;kafka.log:type=LogFlushStats,name=LogFlushRateAndTimeMs;kafka.log:type=LogManager,name=OfflineLogDirectoryCount;kafka.log:type=LogCleaner,name=cleaner-recopy-percent;kafka.log:type=LogCleaner,name=max-buffer-utilization-

percent;kafka.log:type=LogCleaner,name=max-clean-time-secs;kafka.log:type=LogCleanerManager,name=max-dirty-percent;kafka.log:type=LogCleanerManager,name=time-since-last-run-ms!!60000!!10

- **Kafka Connect Agent Option Sample:**

  This example includes the mbeans filter for publishing only the connect metrics required by sample policies.

  kafka-conn.agent.options: kafka.connect:type=connect-metrics,client-id=*;kafka.connect:type=connector-metrics,connector=*;kafka.connect:type=connector-task-metrics,connector=*,task=*;kafka.connect:type=source-task-metrics,connector=*,task=*;kafka.connect:type=sink-task-metrics,connector=*,task=*;kafka.connect:type=connect-worker-metrics,connector=*;kafka.connect:type=connect-worker-metrics;kafka.connect:type=connect-worker-rebalance-metrics;java.lang:type=Runtime;java.lang:type=OperatingSystem;java.lang:type=Memory;java.lang:name=G1 Young Generation,type=GarbageCollector;java.lang:type=Threading!!60000!!10

- **Schema Registry Agent Option Sample:**
  This example includes mbeans filter for publishing only the schema registry metrics required by sample policies.

  confluent-reg.agent.options: kafka.schema.registry:type=app-info,client-id=*;kafka.schema.registry:type=master-slave-role;kafka.schema.registry:type=api-failure-count;kafka.schema.registry:type=avro-schemas-created;kafka.schema.registry:type=avro-schemas-deleted;kafka.schema.registry:type=jetty-metrics;kafka.schema.registry:type=jersey-metrics;kafka.schema.registry:type=kafka.schema.registry-metrics,client-id=*;java.lang:type=Runtime;java.lang:type=OperatingSystem;java.lang:type=Memory;java.lang:name=G1 Young Generation,type=GarbageCollector;java.lang:type=Threading!!60000!!10

- **KSQL Agent Option Sample:**
  This example includes mbeans filter for publishing only the ksql metrics required by sample policies.

  confluent-ksql.agent.options: java.lang:type=Runtime;java.lang:type=OperatingSystem;java.lang:type=Memory;java.lang:name=G1 Young Generation,type=GarbageCollector;java.lang:type=Threading!!60000!!10

- **Rest Proxy Agent Option Sample**
  This example includes mbeans filter for publishing only the rest proxy metrics required by sample policies

  confluent-rest.agent.options: java.lang:type=Runtime;java.lang:type=OperatingSystem;java.lang:type=Memory;java.lang:name=G1 Young Generation,type=GarbageCollector;java.lang:type=Threading;io.confluent.rest:*!!60000!!10

3.5.3.2 Using Zookeeper Orchestrated Approach

Stream JMX can resolve Broker's JMX connection string information through Zookeeper. When you define a broker's JMX connection string, include the ZooKeeper. Stream JMX will not only retrieve the list of brokers connected to this Zookeeper instance but also resolve their JMX connection strings to obtain broker metrics. Below is a sample configuration for the connectionStanzaZK*.cfg file for the broker component. The property used is Kafka.vm.

```
{
   kafka.vm:              kafka:zk://172.16.31.19:2181
#   kafka.vm.user:         admin
#   kafka.vm.pass:         admin
   kafka.vm.reconnect.sec: 10
   kafka.agent.options:   *:*!!60000
   kafka.source.fqn:      SERVICE=@bean:kafka.server:id=?,type=app-
info#NETADDR=@sjmx.serverName#DATACENTER=@bean:kafka.server:type=KafkaServer,n
ame=ClusterId/?Value#APPL=Kafka-Brokers
}
```

If ZooKeeper is configured with SSL and can only be connected over a secure port, the client certificate should be specified using the TNT4JOPTS environment variable in the startup script.

```
-Dzookeeper.client.secure=true \
   -Dzookeeper.ssl.keyStore.location=/path/to/client/keystore.jks \
   -Dzookeeper.ssl.keyStore.password=your_keystore_password \
   -Dzookeeper.ssl.trustStore.location=/path/to/client/truststore.jks \
   -Dzookeeper.ssl.trustStore.password=your_truststore_password \
```

3.5.4General Stream JMX Configuration

Stream-JMX provides configuration properties that let you configure the JMX sampler.

You can configure the JMX sampler using System Properties or Program Arguments. Depending on the sampling environment, one approach may be easier than the other.

When both definitions are available, the System Property value is assigned first, followed by the Program Argument value.

JMX sampler configuration properties are:

- **forceObjectName :** flag indicating to forcibly add objectName attribute, if such is not present for a MBean. Default value is false.
- **addStatisticMetadata :** flag indicating to add J2EE metadata entries as attributes for a MBean. Default value is true.
- **compositeDelimiter :** delimiter used to tokenize composite/tabular type MBean properties keys. Default value is \;
- **useObjectNameProperties :** flag indicating to copy MBean ObjectName contained properties into sample snapshot properties. Default value -is true.
- **excludeOnError :** flag indicating to auto-exclude failed to sample attributes. Default value is false.
- **excludedAttributes :** list of user chosen attribute names (may have wildcards * and ?) to exclude, pattern: attr1,attr2,...,attrN@MBean1_ObjectName;...;attr1,attr2,...,attrN@MBeanN_ObjectName` . Default value is ``.

  To define the Stream JMX sampler configuration property, use the program argument `-slp:`. One argument defines a single property. To define multiple properties, use as many argument definitions as there are required properties. For example:

cmd

- -slp:forceObjectName=true
- -slp:addStatisticMetadata=false
- -slp:compositeDelimiter=.
- -slp:useObjectNameProperties=false
- -slp:excludeOnError=true
- -slp:excludedAttributes=java*@WebSphere:mbeanIdentifier=cells*,*

3.5.4.1 System Properties Used

To define a system property for the application, you can use the common JVM argument -Dkey=value or the SamplingAgent program argument -sp:key=value.

General Use:

- **tnt4j.config :** defines TNT4J properties file path.
  Example: -Dtnt4j.config=".\config\tnt4j.properties"
- **log4j2.configurationFile :** defines stream-jmx logging used LOG4J properties file path.
  Example: -Dlog4j2.configurationFile="file:./config/log4j2.xml"
- **com.jkoolcloud.tnt4j.stream.jmx.agent.forceObjectName :** defines whether to forcibly add objectName attribute if such is not present for a MBean. Default value is false. Example: -Dcom.jkoolcloud.tnt4j.stream.jmx.agent.forceObjectName=true
- **com.jkoolcloud.tnt4j.stream.jmx.agent.addStatisticMetadata :** defines whether to add J2EE statistic metadata entries as attributes for a MBean. Default value is true.
  Example: -Dcom.jkoolcloud.tnt4j.stream.jmx.agent.addStatisticMetadata=false
- **com.jkoolcloud.tnt4j.stream.jmx.agent.compositeDelimiter :** defines delimiter used to tokenize composite/tabular type MBean properties keys. Default value is \. Example: -Dcom.jkoolcloud.tnt4j.stream.jmx.agent.compositeDelimiter=.
- **com.jkoolcloud.tnt4j.stream.jmx.agent.useObjectNameProperties :** defines whether to copy MBean ObjectName contained properties into sample snapshot properties. Default value is true.
  Example: `-Dcom.jkoolcloud.tnt4j.stream.jmx.agent.useObjectNameProperties=false`
- **sjmx.serviceId :** defines stream-jmx service identifier used by TNT4J SourceFQN to distinguish monitored application instance.
  Example: -Dsjmx.serviceId=broker-0 or -sp:sjmx.serviceId=broker-0
- **com.jkoolcloud.tnt4j.stream.jmx.agent.excludeOnError :** defines whether to auto-exclude failed to sample attributes. Default value is false.
  Example: - Dcom.jkoolcloud.tnt4j.stream.jmx.agent.excludeOnError=true
- **com.jkoolcloud.tnt4j.stream.jmx.agent.excludedAttributes :** defines list of user chosen attribute names (may have wildcards * and ?) to exclude, pattern: attr1,attr2,...,attrN@MBean1_ObjectName;...;attr1,attr2,...,attrN@MBeanN_ObjectName. Default value is ``.
  Example: -
  Dcom.jkoolcloud.tnt4j.stream.jmx.agent.excludedAttributes=javaVersion,javaVendor@WebSphere:mbeanIdentifier=cells*,*
- **com.jkoolcloud.tnt4j.stream.jmx.sampler.factory :** defines class name of SamplerFactory class to be used by stream. Default value is com.jkoolcloud.tnt4j.stream.jmx.factory.DefaultSamplerFactory.

  Example:-
  Dcom.jkoolcloud.tnt4j.stream.jmx.sampler.factory=com.jkoolcloud.tnt4j.stream.jmx.impl.WASSamplerFactory

- **com.jkoolcloud.tnt4j.stream.jmx.sampler.batch.period.sec :** defines current batch completion period in seconds.
  Default value is 30 sec.
- **com.jkoolcloud.tnt4j.stream.jmx.sampler.batch.used.memory.percent :** defines current batch completion by used memory percentage threshold.

  Default value is 80%.

- **tnt4j.stream.log.filename :** defines name of stream log file. Default value is ./logs/tnt4j-stream-jmx.log.
  Example: `-Dtnt4j.stream.log.filename=./logs/tnt4j-stream-jmx_broker0.log`
- **tnt4j.activities.log.filename :** defines name of streamed activities log file. Default value is ./logs/tnt4j-stream-jmx_samples.log.

  Example: -Dtnt4j.activities.log.filename=./logs/tnt4j-stream-jmx_broker0_samples.log

3.6 Starting Stream JMX

The Stream JMX process can be started in the following ways:

1. Run Stream-JMX as a `-javaagent`
2. Attach Stream-JMX as an agent to running JVM
3. Connect Stream-JMX over JMXConnector to locally running JVM or remote JMX service

Option 3 is preferred and widely used. Using this approach, Stream JMX connects to the Kafka component over the JMX port previously enabled with *section 3.2.2*. Please refer to sections *3.5.1* & *3.5.2* for more information on starting Stream JMX using -connect option.

3.6.1Start Using Default Scripts Shipped with Package

The package includes some sample scripts that demonstrate how the properties configured in the tnt4j properties file and connectionStanzaZK config file are sent to the shell script **stream-jmx-connect-file-config.sh** located in the $STREAM_JMX_HOME/run directory. This script starts Stream JMX in connect mode.

As per the recommendation, each cluster will have its own config & property file, so each one needs to start a separate process. This can be achieved using the sample scripts. The changes required are the names of property, config files & jmx.serviceId defined in each script.

Using the scripted method is recommended.

3.6.2 Start from CLI

The Stream JMX process can be started from the command line using stream-jmx-connect.bat or stream-jmx-connect.sh or directly using Java, by including class path. This section lists the available methods.

3.6.2.1 Using stream-jmx-connect utility available under executables ($STREAM_JMX_HOME/bin)

The following OS shell script files are provided for connecting Stream-JMX to a local or remote Kafka JMX service:

- bin/stream-jmx-connect.bat or
- bin/stream-jmx-connect.sh

Windows:

- rem using URL

  /bin/stream-jmx-connect.bat service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi

- rem using URL with connection parameters

  /bin/stream-jmx-connect.bat service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi -ul:admin -up:admin -cp:java.naming.security.authentication=simple -cp:java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory

- rem using process name part

/bin/stream-jmx-connect.bat server.properties

- rem using pid

  /bin/stream-jmx-connect.bat 1553

Unix/Linux:

- # using URL

  ./bin/stream-jmx-connect.sh service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi

- # using URL with connection parameters

  ./bin/stream-jmx-connect.sh service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi -

  ul:admin -up:admin -cp:java.naming.security.authentication=simple -

  cp:java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory

- # using process name part

  ./bin/stream-jmx-connect.sh server.properties

- # using pid

./bin/stream-jmx-connect.sh 1553

3.6.2.2 To connect to local JVM process

Command line to connect local JVM process JMX looks like this:

java -Dtnt4j.config=.\config\tnt4j.properties -Dcom.jkoolcloud.tnt4j.stream.jmx.agent.trace=true -classpath "tnt4j-stream-jmx-core0.7-all.jar" com.jkoolcloud.tnt4j.stream.jmx.SamplingAgent -connect -vm:server.properties -ao:*:*!10000

System properties –Dxxxxx defines Stream-JMX configuration. For details, refer to General StreamJMX Configuration (*Section 3.4.3*).

SamplingAgent arguments -connect -vm:server.properties -ao:*:*!10000 states:

- **-connect :** defines that SamplingAgent shall connect to running JVM process over JMXConnector (RMI) connection.
- **-vm:server.properties :** is JVM descriptor. In this case it is running JVM name fragment server.properties. But it also may be JVM process identifier - PID. Mandatory argument.
- **-ao:*:*!10000 :** is JMX sampler options stating to include all MBeans and schedule sampling every 10 seconds. Sampler options are optional. Default value = *:*!30000. Initial sampler delay can be configured by adding numeric parameter *:*!30000!1000 defining initial sampler delay as 1 second. Default sampler delay value is equal to sampling period value.
- **-slp:** – any JMX sampler configuration property. Refer to Program Arguments Used (*section 3.4.3.2*) for details

3.6.2.3 To Connect to JMX Service Over URL

Command line to connect remote JMX service looks like this:

java -Dtnt4j.config=.\config\tnt4j.properties -Dcom.jkoolcloud.tnt4j.stream.jmx.agent.trace=true -classpath " tnt4j-stream-jmxcore-0.7-all.jar" com.jkoolcloud.tnt4j.stream.jmx.SamplingAgent -connect -vm:service:jmx:[JMX_URL] -ul:admin -up:admin -ao:*:*!!10000 -cri:30 -

cp:java.naming.security.authentication=simple -
cp:java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory

System properties –Dxxxxx defines Stream-JMX configuration. For details, refer to General StreamJMX Configuration (0).

SamplingAgent arguments -connect -vm:service:jmx:[JMX_URL] -ul:admin -up:admin -ao:*:*!!10000 -cri:30 -cp:java.naming.security.authentication=simple -cp:java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory states:

- **-connect :** defines that SamplingAgent shall connect to running JMX service over JMXConnector (RMI) connection.
- **-vm:service:jmx:[JMX_URL] :** is JMX service URL to use for connection. Mandatory argument. Full URL may be like: service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi
- **-ul:admin :** is user login. In this case it is admin. User login argument is optional.
- **-up:admin :** is user password. In this case it is admin. User password argument is optional.
- **-ao:*:*!!10000 :** is JMX sampler options stating to include all MBeans and schedule sampling every 10 seconds. Sampler options are optional - default value is *:*!30000. Initial sampler delay can be configured by adding numeric parameter *:*!30000!1000 defining initial sampler delay as 1 second. Default sampler delay value is equal to sampling period value.
- **-cp:java.naming.security.authentication=simple :** cp:java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory - is JMX connector parameters definitions in properties format key=value. JMX connector parameters are optional and can be defined multiple times; as many as there are required JMX connector parameters. Refer to Java API Context class documentation for available properties naming.


NOTE

If you are If you are using some API extending JNDI, check documentation if it provides some additional connection configuration properties..

- **-cri:30** – is connection retry interval in seconds. In this case it is 30 seconds between connect retry attempts. Connection retry interval is optional. Default value = 10 sec. Special values are:
  - 0 indicates no delay between repeating connect attempts.
  - -1 indicates no repeating connect attempts shall be made at all and application must stop on first failed attempt to connect.
- **-slp:** - any JMX sampler configuration property. Refer to Program Arguments Used (*section 3.4.3.2*) for details

This Page Intentionally Left Blank

Chapter 4: Kafka JMX Metrics

This section describes some of the Kafka Expert metrics collected by the expert coming from the Kafka. The data presented is collected using JMX services and can be modified as discussed in the configuration section above. The facts produced are samples only.

4.1 Zookeeper Metrics

Sample Zookeeper metrics collected & published by Stream JMX.

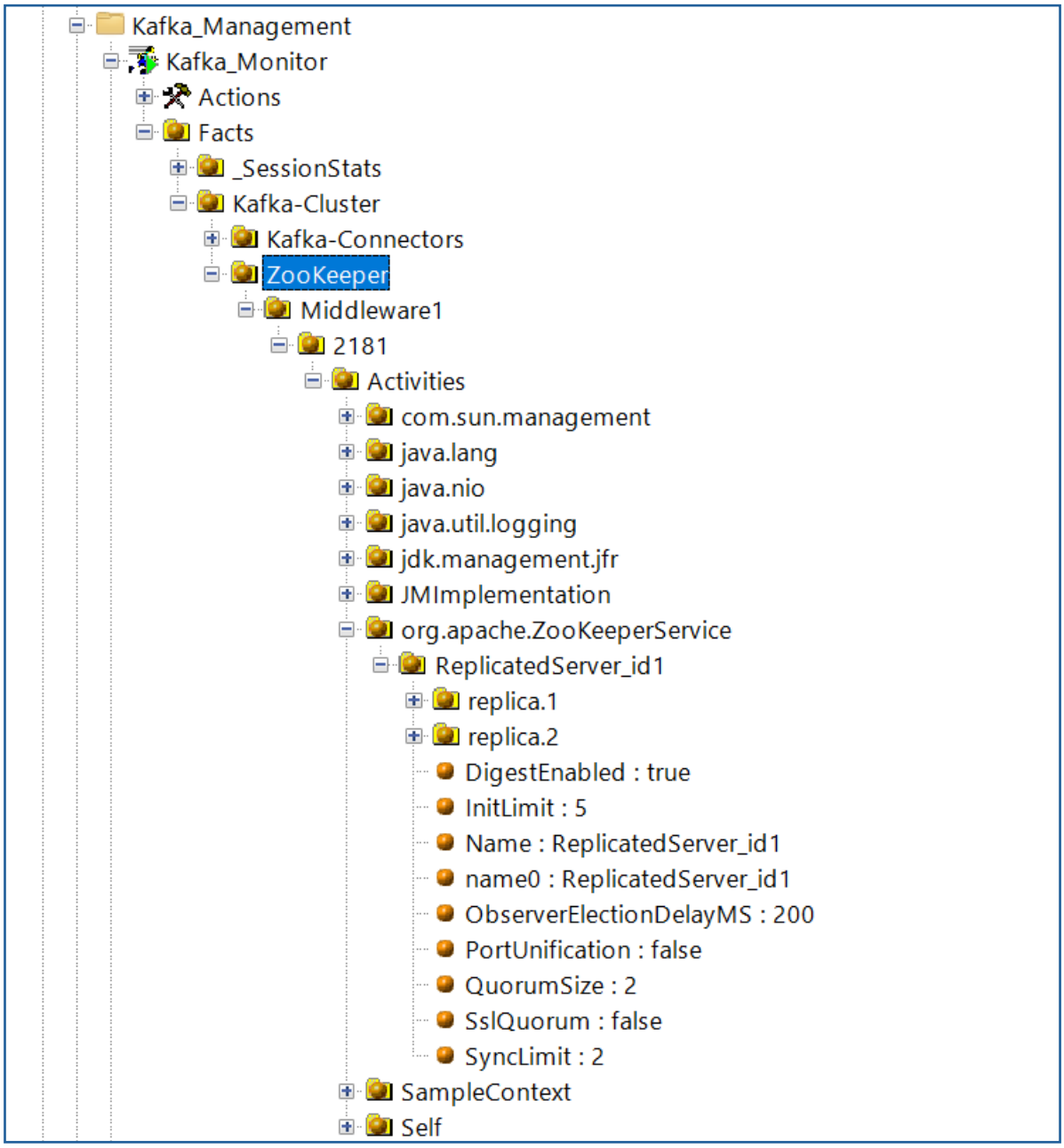


Figure 4-1. Zookeeper Metrics

4.2 Broker Metrics

Sample Kafka Broker metrics collected & published by Stream JMX.

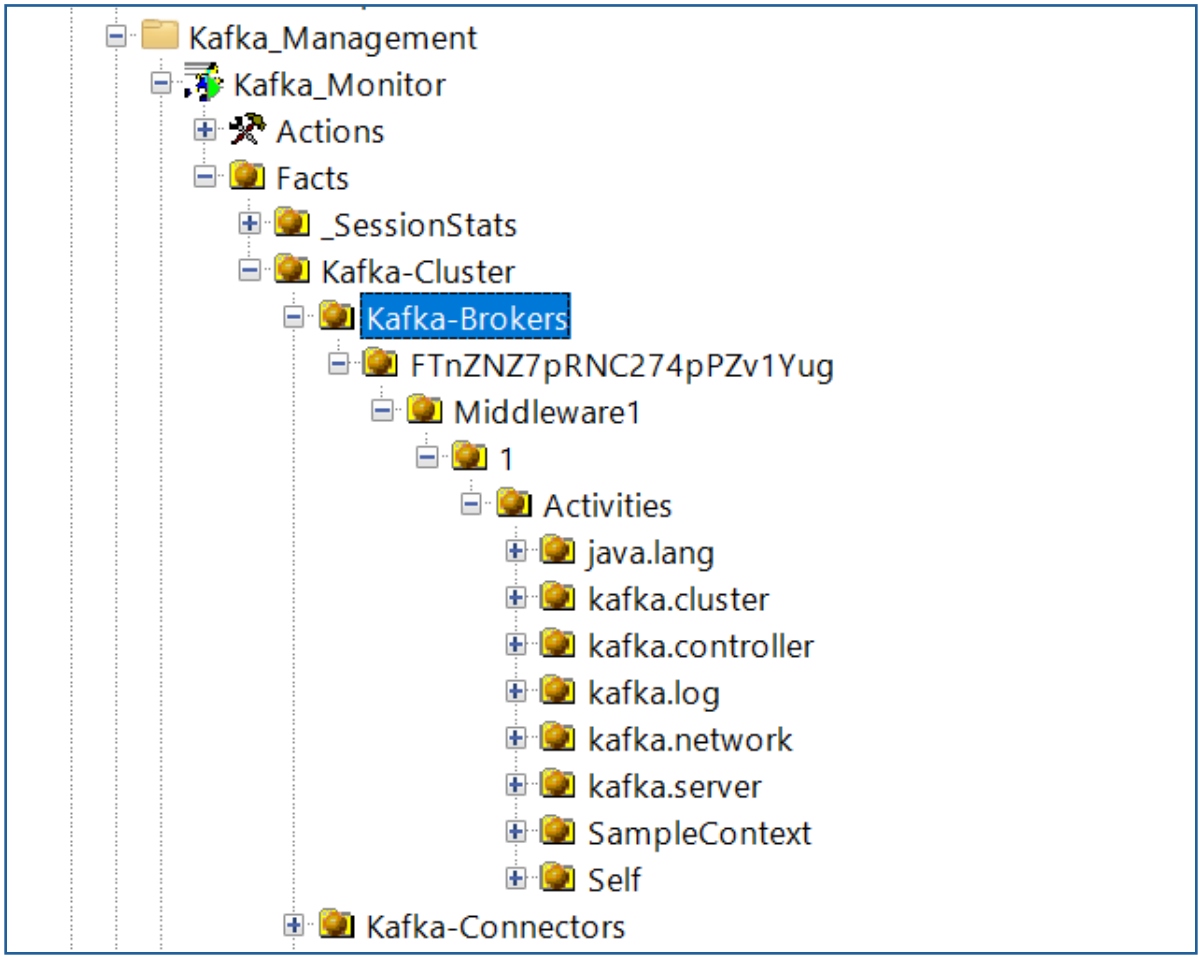


Figure 4-2. Broker Metrics

4.3 Connect Metrics

Sample Connect metrics collected & published by Stream JMX.

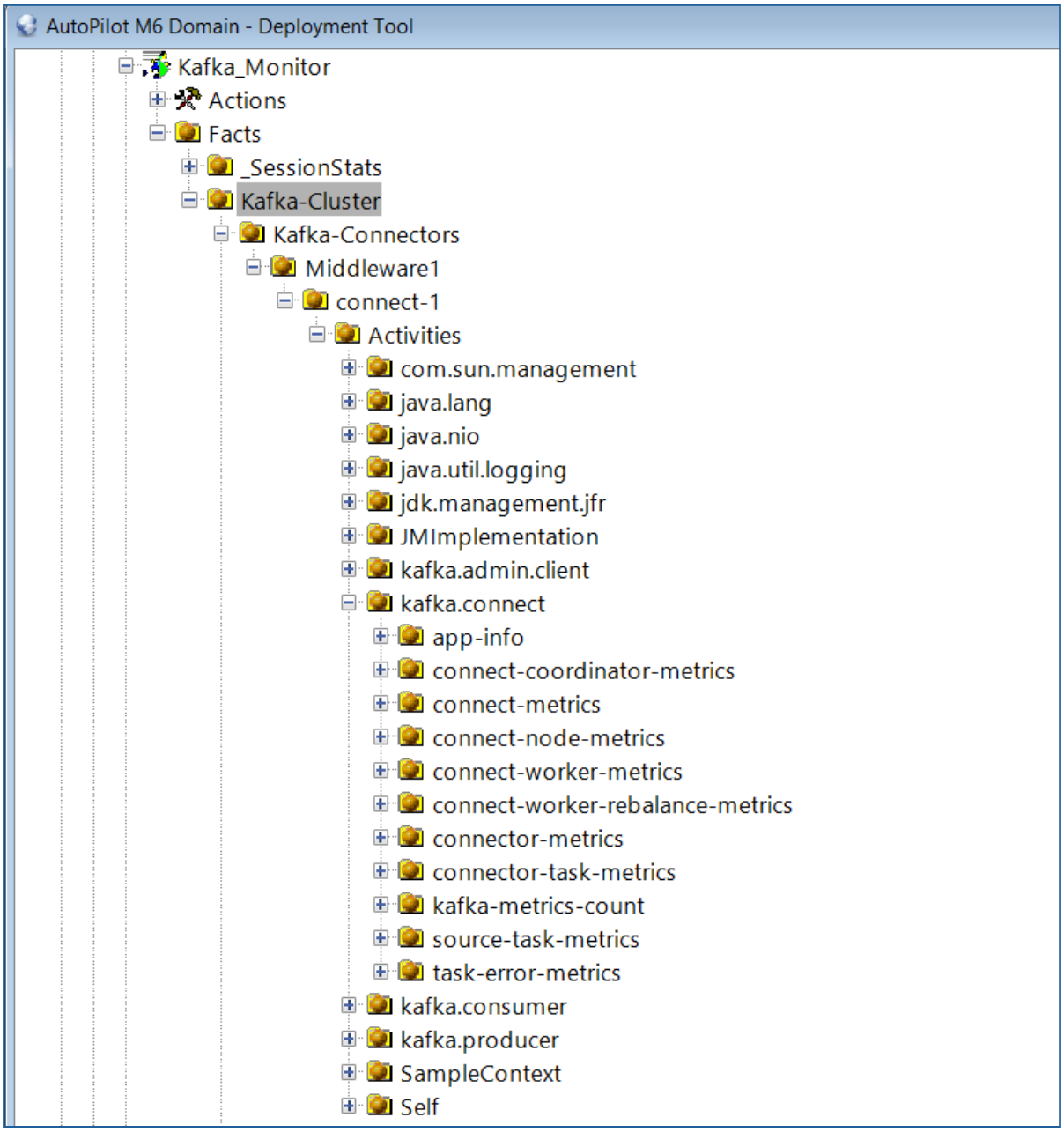


Figure 4-3. Connect Metrics

4.4 Schema Registry Metrics

Sample Schema Registry metrics collected & published by Stream JMX.

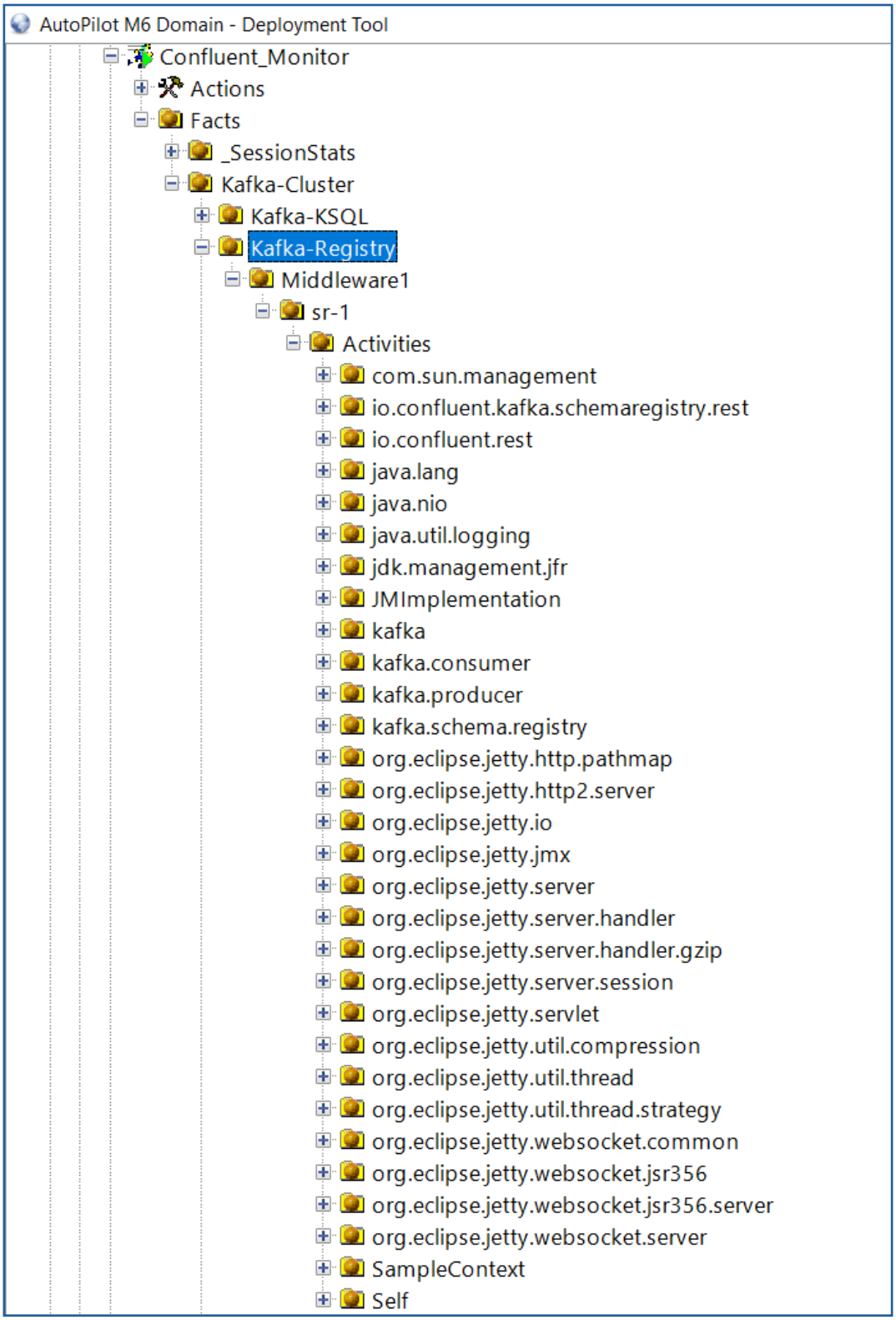


Figure 4-4. Schema Registry Metrics

4.5 KSQL Metrics

Sample KSQL metrics collected & published by Stream JMX.

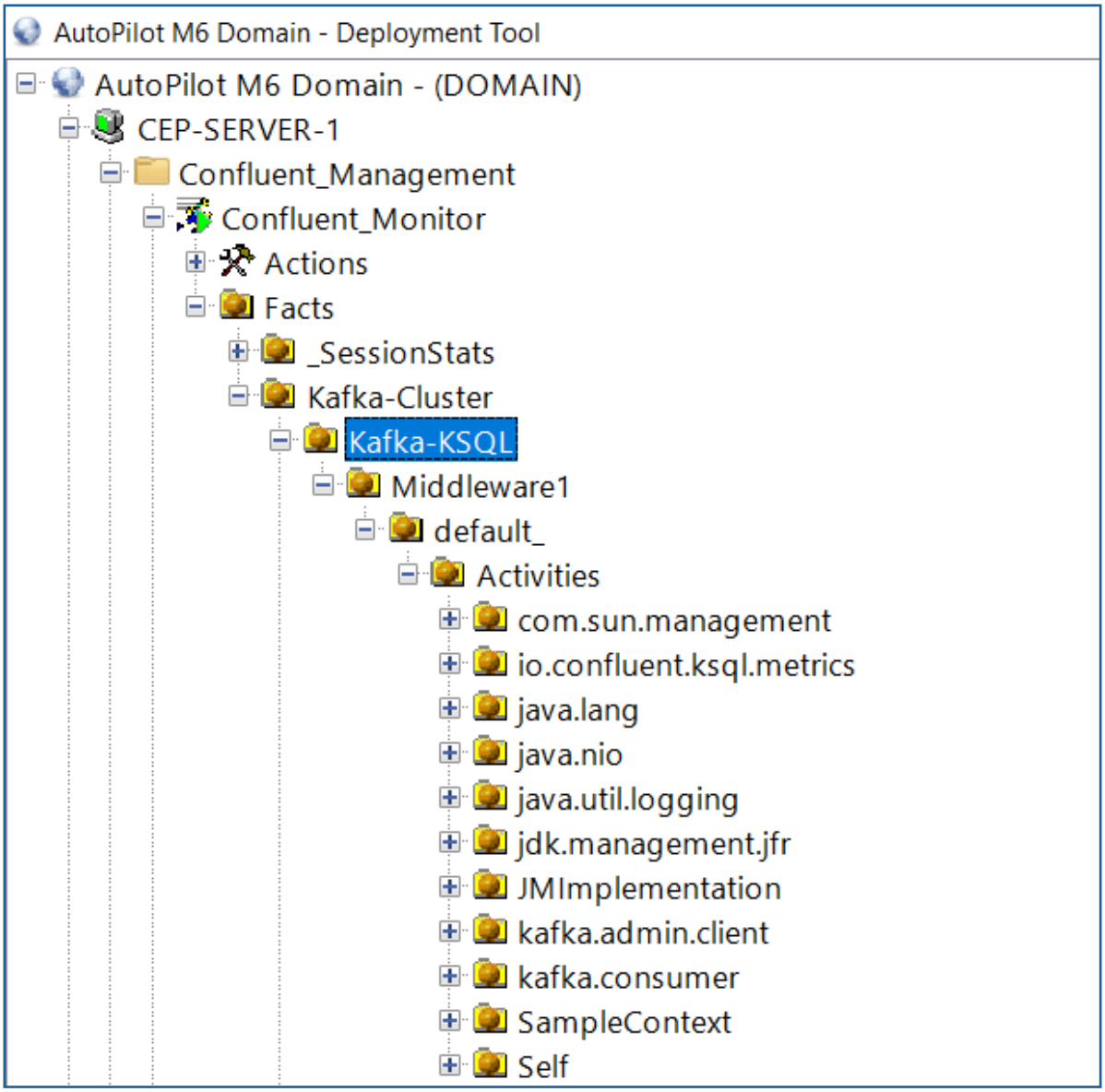


Figure 4-5. KSQL Metrics

4.6 Kafka Rest Proxy Metrics

Sample Kafka Rest Proxy metrics collected & published by Stream JMX.

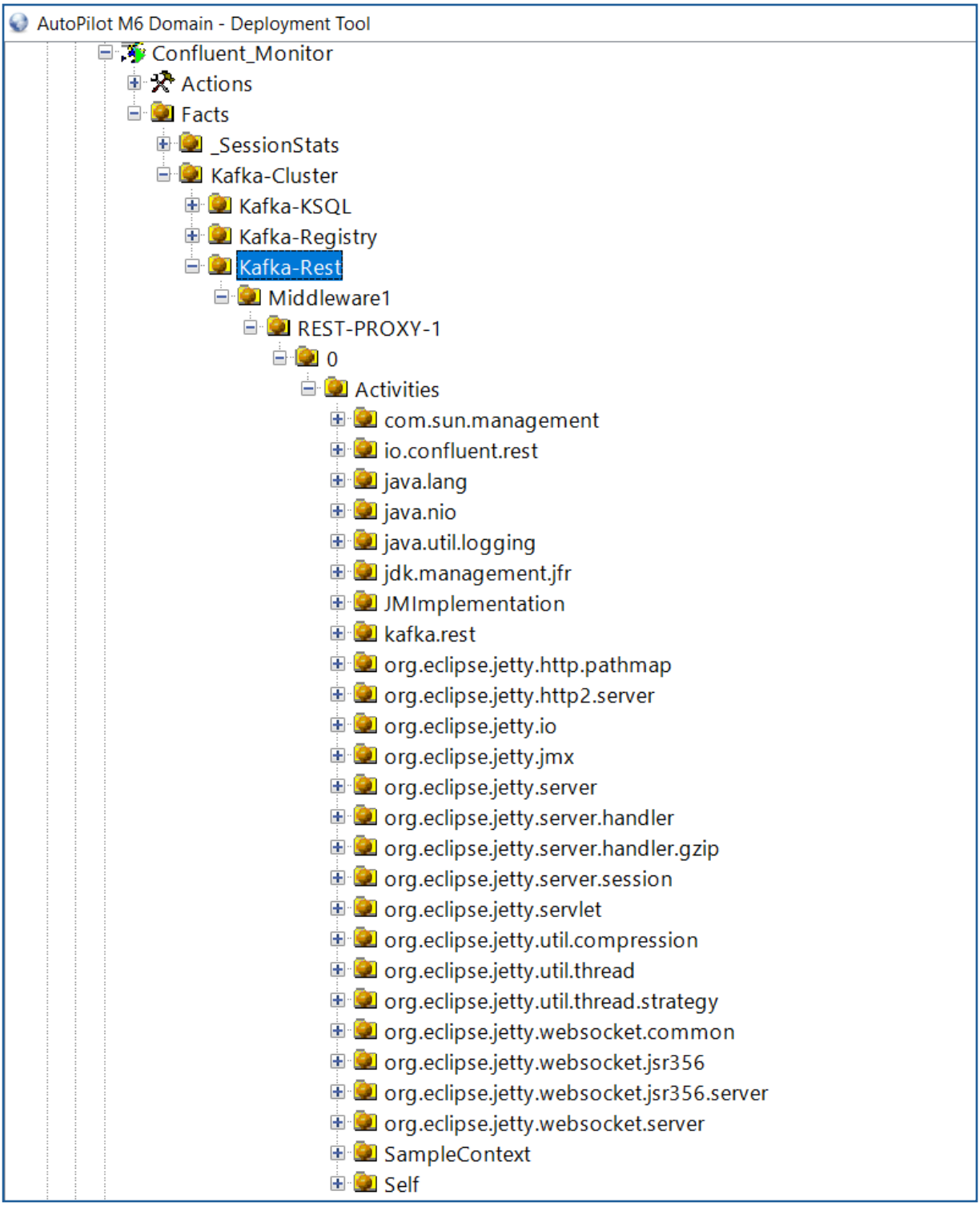


Figure 4-6. Kafka Rest Proxy Metrics

This Page Intentionally Left Blank

Chapter 5: Kafka JMX Sample Policies

Stream JMX package provides a set of out-of-the-box policies that analyse the data collected by the Kafka Monitor & Confluent Monitor. This section outlines the procedure to deploying policy managers, policies and key policies. Additional out-of-the-box policies are also included.

5.1 Create & Deploy Policy Manager

Policies are deployed under a policy manager, which is responsible for managing the deployed policies (starting, stopping, and setting auto-start)

To Deploy a Policy Manager:

- Right-click on CEP and choose Deploy Manager -> Default -> Policy Manager.
- In the create window under General tab, set Context to Kafka_Management & Name to Kafka_Policies.
- Deploy the policy manager, which will appear in the deployment view under CEP.

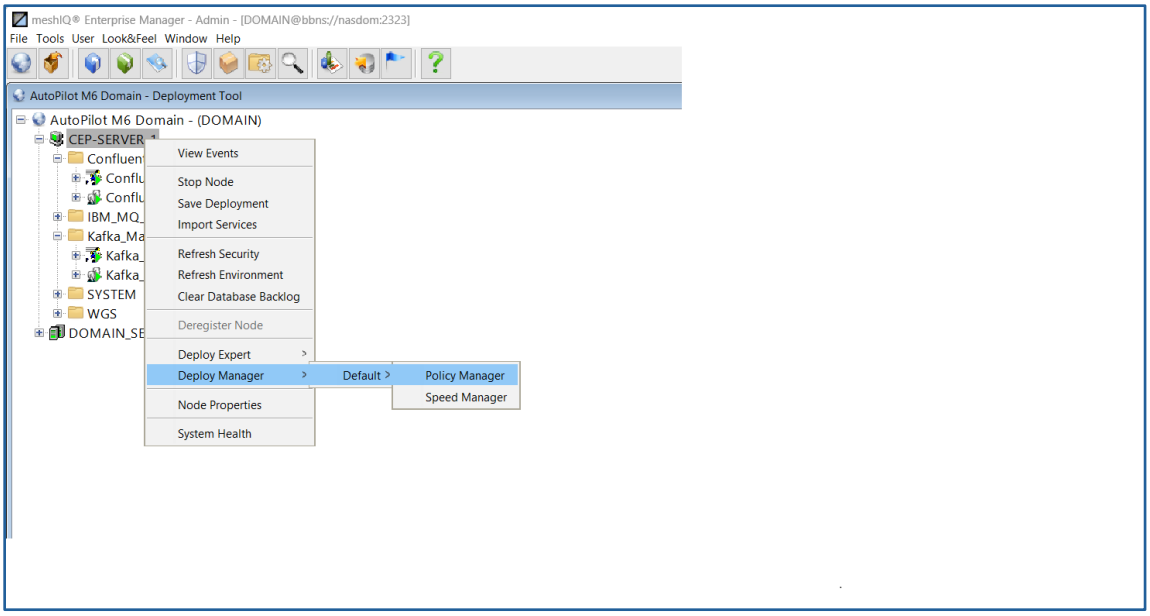


Figure 5-1. Deploy Policy Manager

Follow the same procedure to deploy the Confluent_Policies policy manager.

5.2 Deploy Policies Under Created Policy Manager

Policies copied to Domain VMs/server under naming/policies directory (see *section 3.3*) at the time of installation should be deployed under policy manager.

To Deploy a Policy :-

- Open **Business View Explorer** and expand Kafka_Policies directory.
- Right-click on a policy and choose **Deploy As Policy** option.
- In the pop-up window, choose the policy manager and click **Deploy**.
- Repeat the process to deploy all the required policies.
- Lastly, go to deployment tool and start all of them manually.

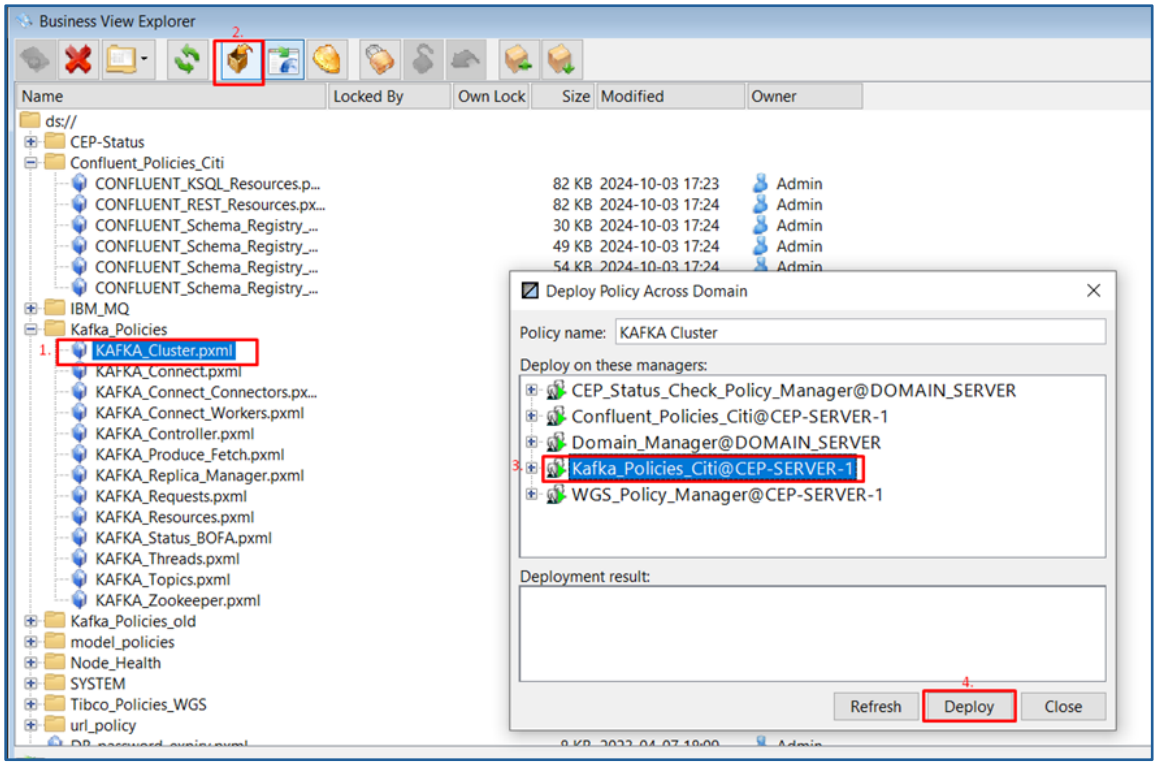


Figure 5-2. Deploy Policy Under Policy Manager

All the policies use environment variables named KAFKA_MON and CONFLUENT_MON. These variables should be added to $AUTOPILOT_HOME/localhost/node.properties and assign appropriate values (the name of the process wrapper deployed under CEP).

5.3 Kafka Resources

This example analyses the tracking of JVM memory over time.

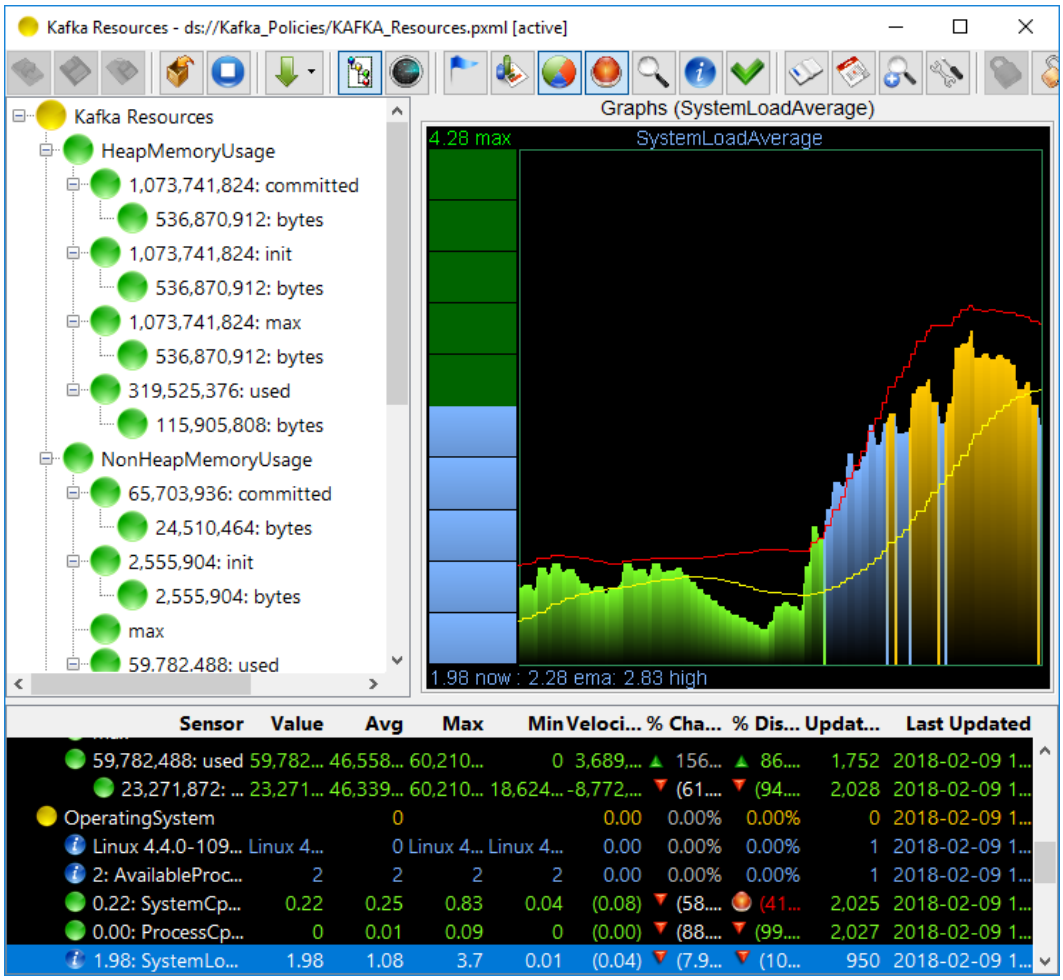


Figure 5-3. Kafka Resources

This Page Intentionally Left Blank

Appendix A: References

A.1 meshIQ Documentation

Table A-1. meshIQ Documentation

Document Number (or higher)	Title
M6/INS 625.001	*AutoPilot M6 Installation Guide*
M6/USR 625.001	*AutoPilot M6 User's Guide*
M6WMQ 600.004	*AutoPilot M6 Plug-in for WebSphere MQ*
M6WMQ/ADM 658.002	*AutoPilot M6 for WebSphere MQ Administrator's Guide*
M6WMQ/INS 658.002	*AutoPilot M6 for WebSphere MQ Installation Guide*
M6/OSM 600.002	*AutoPilot M6 Operating System Monitors Installation and User's Guide*
AP/TEMS 110.004	*AutoPilot Plug-in for TIBCO EMS*
AP/OR 100.006	*AutoPilot/Oracle Plug-in Guide*
AP/IT JMX 430.001	*AutoPilot/JMX Plug-in Guide*

This Page Intentionally Left Blank

Appendix B: Conventions

B.1 Typographical Conventions

Table B-1. Typographical Conventions

Convention	Description
Blue/Underlined	Used to identify links to referenced material or websites. Example: *support@meshiq.com*
Bold Print	Used to identify topical headings and to identify toggle or buttons used in procedural steps. Example: Click **EXIT**.
Italic Print	Used to place emphasis on a title, menu, screen name, or other categories.
Monospaced Bold	Used to identify keystrokes/data entries, file names, directory name etc.
Monospaced italic	Used to identify variables in an address location. Example: [*C:\AutoPilot_Home*]\documents. Where the portion of the address in the brackets [ ] is variable.
Monospaced Text	Used to identify addresses, commands, scripts etc.
Normal Text	Typically used for general text throughout the document.
Table Text	Table text is generally a smaller size to conserve space. 10, 9, and 8 point type is used in tables throughout the AutoPilot product family of documents

B.2 Naming Conventions

Naming conventions have been adjusted to accommodate IBM's re-naming of MQSeries products to WebSphere MQ.

meshIQ has adapted AutoPilot products to reflect IBM's product naming changes. In the redesign of AutoPilot, we have also better defined many elements within the AutoPilot product line.

Table B-2. AutoPilot Related Naming Conventions

Old Name	New Name
AutoPilot/MQSI	AutoPilot for WebSphere MQI
MQSeries Plug-in for AutoPilot	WebSphere MQ Plug-in for AutoPilot
MQControl	AutoPilot for WebSphere MQ
MQSeries	WebSphere MQ (IBM)