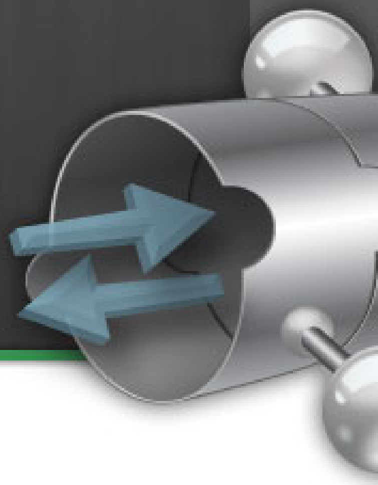


Talend ESB System Management Integration

User Guide

5.1

Talend ESB System Management Integration: User Guide

Publication date 3 May 2012

Copyleft

Notices

Table of Contents

Chapter 1. Introduction

This document looks at System Management Integration tools and their use within Talend ESB.

We look at Hyperic HQ integration support in Chapter 2, *Hyperic HQ Integration*, JMX configuration in Chapter 3, *JMX configuration* and Chapter 4, *Nagios Integration* deals with Nagios integration support.

Chapter 2. Hyperic HQ Integration

2.1. Hyperic HQ Integration

These sections describe the Hyperic HQ integration support in Talend ESB. This consists of two plugins files, one for Camel support and one for CXF support, which ship with the release and corresponding examples.

This consists of a number of stages:

1. Configure Tomcat and Karaf for JMX configuration, in Section 2.3.1, “Pre-requisites for deploying the plugins” . This enables the gathering of the basic metrics information.
2. Deploy the Hyperic HQ plugins and use them to autodiscover resources, in Section 2.3.2, “How to deploy Hyperic plugins”. This also shows how to display the metrics, and this output is updated as the examples run.
3. Build and run the examples to generate metrics, in README files for examples
4. Look at the updated metrics
5. To see the complete list of available CXF and Camel metrics, we also look at using jconsole - (which is additional information to that of the Hyperic HQ plugins), in Section 3.2, “Camel route and CXF service metrics”.

2.2. Introduction to Hyperic HQ

2.2.1. Hyperic HQ overview

The smooth running of the computer infrastructure is a critical part of any business. This requires constant system monitoring of network resources, to be aware of what is happening and of any problems that may arise, for example, in services being unavailable, or generating faults. In the event of emergencies, the system can be configured to notify key personnel about the problem and can help resolve it.

Hyperic is a cross platform monitoring system which is designed to monitor and control server resources. The system implements four general functions:

- **Discovery**: HQ Agents that run on the machines in your environment automatically detect, or *auto-discover*, the software resources running on the machine. When HQ discovers a software resource, it collects key facts about it, including its type, vendor, version, and location
- **Monitoring**: HQ agents track the current state of services and servers in real time, and automatic detect abnormalities
- **Alerts**: this subsystem will notify you about problems at the resources that are monitored. Alerts can be sent to administrator using e-mail, a mobile phone or pager
- **Control**: You can use HQ for remote control and administration of your software resources. Available control actions vary by resource type

You can find further information about Hyperic at the "Hyperic" site http://www.hyperic.com/. For more information about installing Hyperic, please see : http://support.hyperic.com/display/DOC/QuickStart+Installation.

2.2.2. System Requirements

Camel and CXF management demo examples can be installed, built, and run on Windows or Linux for which Java 6 is available. The following software is required for installing, building, and running the samples:

- JDK 1.6.0 from Oracle should be installed, and the environment variable `JAVA_HOME` should be set to the JDK installation directory. Later JDKs from Oracle or JDKs from other providers may work but have not been tested.
- Maven 3.0.3 or later from Apache should be installed, and the **mvn** executable should be in your `PATH`. When running **mvn**, HTTP access to the internet is required. The local Maven repository is expected to be created in its default location, i.e. the Maven configuration should not have been modified.
- Tomcat 6.0 or later from Apache should be installed, and the environment variable `CATALINA_HOME` should be set to the Tomcat installation directory.
- The Talend OSGi container is included in the release installation in sub-directory `container`. The current version of Karaf is 2.2.5.
- Hyperic HQ 4.5.1 (CE or EE) Server and Agent from SpringSource should be installed. Later versions may work but have not been tested.

2.2.3. Release directory structure

First, you need to download and extract the Talend ESB software (please see **Talend ESB Getting Started User Guide** for more details). After you unpack the Talend ESB release, the plugin files are in a subdirectory `adapters`, which is structured as follows:

```
adapters/
    plugins/
       camel-plugin.xml
       cxf-plugin.xml
    LICENSE.txt
    README
```

2.3. Deploying the Hyperic HQ plugins

2.3.1. Pre-requisites for deploying the plugins

In order to provide metrics for monitoring, the Talend OSGi container or Tomcat web application container must be configured for remote JMX access.

Note

The Karaf container of the Talend ESB installation is already configured.

For a Tomcat web application server, you must set an environment variable when starting it:

```
CATALINA_OPTS="-Dcom.sun.management.jmxremote \
-Dcom.sun.management.jmxremote.port=6969 \
-Dcom.sun.management.jmxremote.ssl=false \
-Dcom.sun.management.jmxremote.authenticate=false"

export CATALINA_OPTS
```

This can be defined in a file:

`set CATALINA_OPTS=...`

On Linux, this is set at the beginning of `catalina.sh`

On Windows, it is set in `catalina.bat`.

Warning

To create MBeans that collect statistics about CXF services you first need to make an invocation on the CXF service, which registers the MBeans. Without that step Hyperic HQ won`t find metrics as MBeans don't exist.

2.3.2. How to deploy Hyperic plugins

Copy the Hyperic plugins to these folders:

- `<HypericServer>/hq-engine/hq-server/webapps/ROOT/WEB-INF/hq-plugins`
- `<HypericAgent>/bundles/agent-<version>/pdk/plugins`

The complete information about deploying Hyperic plugins is at http://support.hyperic.com/display/EVO/Deploy+Plugin

Tomcat and Karaf must be running to enable Hyperic to discover the servers.

1. Start the Hyperic server, if it wasn't running.
2. Start the Hyperic agent, if it wasn't running. Otherwise, restart it.

Information how to start/stop/restart Hyperic agent and Hyperic server is in the Hyperic documentation http://support.hyperic.com

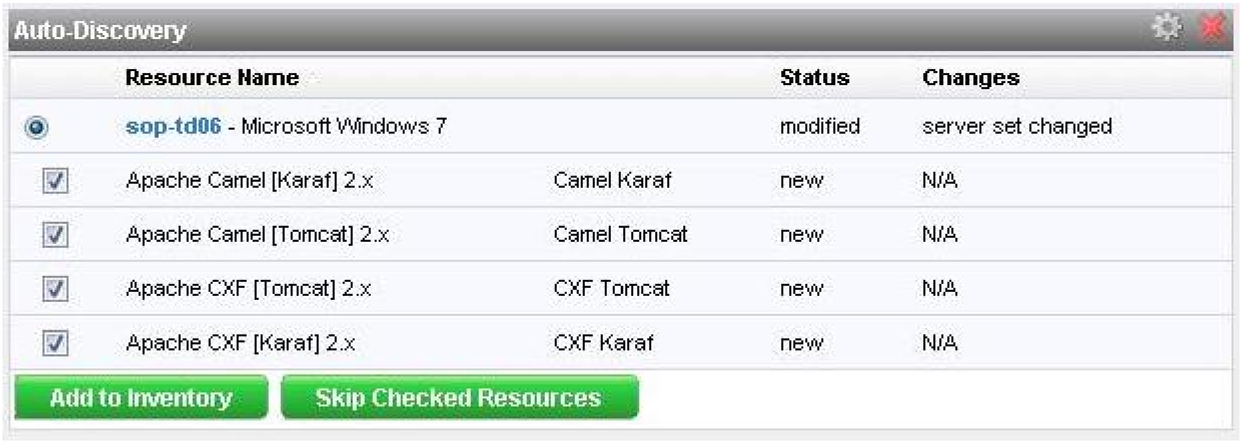
Open your browser and login to Hyperic server (default http://localhost:7080/). You will see new discovered servers in the **Autodiscovery**.

Note

Sometimes servers are not detected correctly. In this case you need to restart Hyperic HQ Agent. From the `<HypericAgent>/bin` directory restart agent:

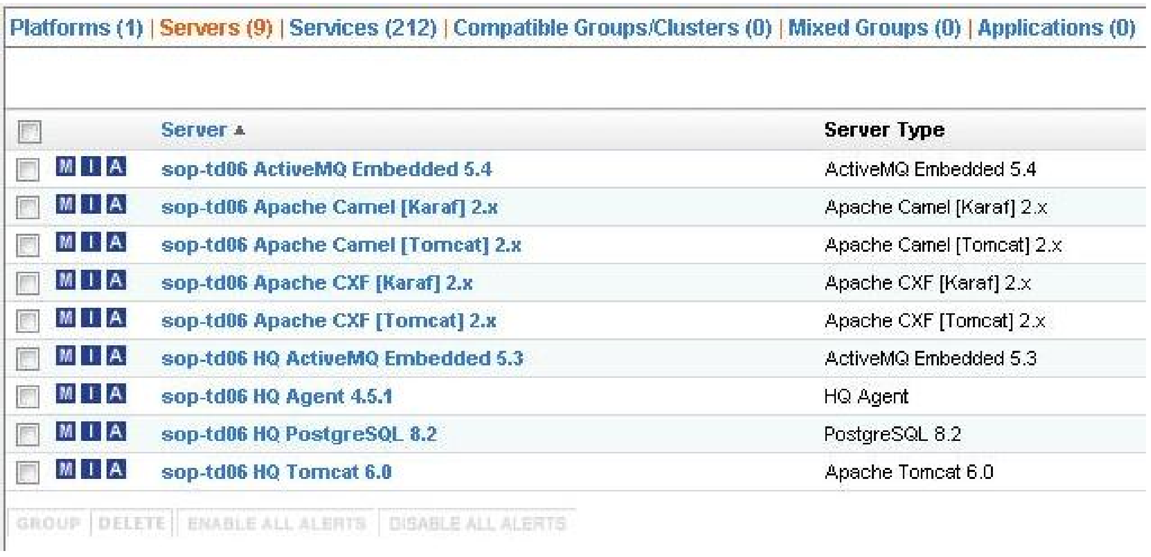
`hq-agent.bat restart` (on Windows)

`sh hq-agent.sh restart` (on Linux)


Auto-Discovery

	Resource Name		Status	Changes
●	sop-td06 - Microsoft Windows 7		modified	server set changed
✓	Apache Camel [Karaf] 2.x	Camel Karaf	new	N/A
✓	Apache Camel [Tomcat] 2.x	Camel Tomcat	new	N/A
✓	Apache CXF [Tomcat] 2.x	CXF Tomcat	new	N/A
✓	Apache CXF [Karaf] 2.x	CXF Karaf	new	N/A

1. Click on **Add to Inventory**.
2. Choose **Resources** Tab
3. Find and choose **Apache Camel(CXF) [Tomcat]([Karaf]) 2.x** from the list of servers, depending on which combination you wish to monitor. For example, you may wish to monitor Camel/Tomcat and CXF/Tomcat.

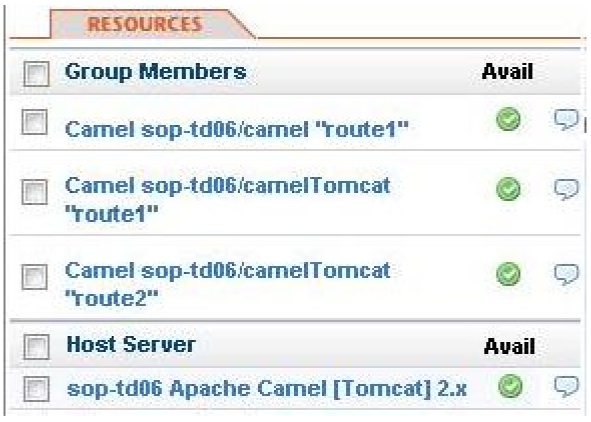


Platforms (1) | Servers (9) | Services (212) | Compatible Groups/Clusters (0) | Mixed Groups (0) | Applications (0)

		Server ▲	Server Type
☐	M I A	sop-td06 ActiveMQ Embedded 5.4	ActiveMQ Embedded 5.4
☐	M I A	sop-td06 Apache Camel [Karaf] 2.x	Apache Camel [Karaf] 2.x
☐	M I A	sop-td06 Apache Camel [Tomcat] 2.x	Apache Camel [Tomcat] 2.x
☐	M I A	sop-td06 Apache CXF [Karaf] 2.x	Apache CXF [Karaf] 2.x
☐	M I A	sop-td06 Apache CXF [Tomcat] 2.x	Apache CXF [Tomcat] 2.x
☐	M I A	sop-td06 HQ ActiveMQ Embedded 5.3	ActiveMQ Embedded 5.3
☐	M I A	sop-td06 HQ Agent 4.5.1	HQ Agent
☐	M I A	sop-td06 HQ PostgreSQL 8.2	PostgreSQL 8.2
☐	M I A	sop-td06 HQ Tomcat 6.0	Apache Tomcat 6.0

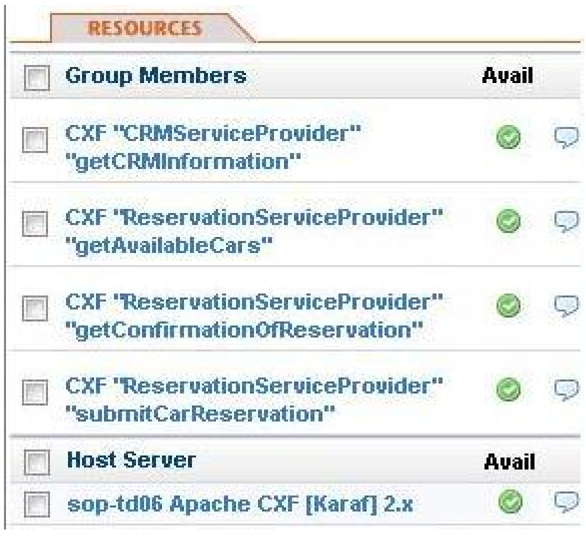
4. Choose a service group.

   For Camel you should see the group of the routes specified by context and name.



Group Members	Avail
Camel sop-td06/camel "route1"	✓
Camel sop-td06/camelTomcat "route1"	✓
Camel sop-td06/camelTomcat "route2"	✓
Host Server	**Avail**
sop-td06 Apache Camel [Tomcat] 2.x	✓

   For CXF you should see the group of the services specified by port and operation.



Group Members	Avail
CXF "CRMServiceProvider" "getCRMInformation"	✓
CXF "ReservationServiceProvider" "getAvailableCars"	✓
CXF "ReservationServiceProvider" "getConfirmationOfReservation"	✓
CXF "ReservationServiceProvider" "submitCarReservation"	✓
Host Server	**Avail**
sop-td06 Apache CXF [Karaf] 2.x	✓

5. Hyperic will collect the metrics for these groups, and the initial results will be displayed.

Note

These will subsequently updated as the examples are run.

Chapter 3. JMX configuration

3.1. Configuration

You can take the default JMX configuration, which facilitates monitoring and management of Java applications. This section discusses how you can override these defaults, and perform more advanced configuration.

3.1.1. How to make advanced JMX configuration for Camel routes and CXF services

3.1.1.1. JMX configuration

Plugin section, in `camel-plugin.xml` and `cxf-plugin.xml`:

For the Karaf server, by default, in the plugin, there is the following JMX configuration properties:

Property Name	**Description**	**Default**
jmx.url	the JMX Service URL	service:jmx:rmi://localhost:44444/jndi/rmi://localhost:1099/karaf-trun
jmx.username	Username authentication	tadmin
jmx.password	Password authentication	tadmin

For the Tomcat server, by default, in the plugin, there is the following JMX configuration properties:

Property Name	Description	Default
jmx.url	the JMX Service URL	service:jmx:rmi:///jndi/rmi://localhost:6969/jmxrmi
jmx.username	Username authentication	""
jmx.password	Password authentication	""

3.1.1.2. JMX configuration for Camel routes

Apache Camel has support for JMX and allows you to monitor a Camel managed object (for example, routes). By default, a JMX agent is enabled in Camel which means that the Camel runtime creates and registers MBean management objects with a MBeanServer instance in the VM. But if you would like to configure a JMX agent (for example to use a custom port in JMX URL) the best way to do it is adding a `jmxAgent` element inside the camelContext element in Spring configuration:

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <jmxAgent id="agent" mbeanObjectDomainName="your.domain.name">
    ...
<camelContext>
```

The default JMX configuration is used in both examples, but you can also configure it:

```
<jmxAgent id="agent" registryPort="port number" createConnector="true">
```

`createConnector` means that we should create a JMX connector (to allow remote management) for the MBeanServer.*`registryPort`* is the port for JMX.

You can set *`hostName`* and *`domainName`* for `DefaultManagementNamingStrategy`. As a default, *`hostName`* is the computer name and *`domainName`* is `org.apache.camel`

```
<bean id="naming"
        class="org.apache.camel.management.DefaultManagementNamingStrategy">
    <property name="hostName" value="localhost"/>
    <property name="domainName" value="org.apache.camel">
</bean>
```

To configure specific definitions for the Camel route object use the properties:

Property Name	Description
org.apache.camel	domain name
routes	Camel routes type
context	Camel context name
name	route name

You can find further information about configuring Camel JMX agent at the "Camel" site http://camel.apache.org/camel-jmx.html.

3.1.1.3. JMX configuration for CXF services

Each server type defines several service types such as EJBs, Connection Pools, JMS Queues, and so on. The plugin defines additional service types to provide management of CXF via custom MBeans. The service element defines a service type, for example:

Plugin object section:

```
<service name="CXF all services monitoring">
    <property name="OBJECT_NAME"
  value='org.apache.cxf:bus.id=*,type=Performance.Counter.Server,service=*,
  port=*,operation=*'/>
    <metrics include="cxf"/>
    <plugin type="autoinventory" />
</service>
```

In order to access custom MBeans, the plugin must define its JMX ObjectName to be used with various MBeanServer interface methods. Only one ObjectName is defined per service type using the property tag within the service tag.

To configure specific definitions for the CXF service object use properties:

Property Name	**Description**
org.apache.cxf:bus.id	id of specific CXF bus
service	service endpoint name
port	service port name
operation	service operation name

To enable JMX integration for CXF you need to declare the following bean in service Spring configuration:

```
<bean id="org.apache.cxf.management.InstrumentationManager"
  class="org.apache.cxf.management.jmx.InstrumentationManagerImpl">
  <property name="bus" ref="cxf" />
  <property name="usePlatformMBeanServer" value="true" />
  <property name="createMBServerConnectorFactory" value="false" />
  <property name="enabled" value="true" />
</bean>
```

To avoid any unnecessary runtime overhead, the performance counters measuring response time are disabled by default. To collect statistics for running sevices define the following bean:

```
<bean id="CounterRepository"
            class="org.apache.cxf.management.counters.CounterRepository">
        <property name="bus" ref="cxf" />
</bean>
```

For further information about configuring JMX in CXF you can find at Apache CXF.

3.1.2. Advanced configuration for CXF and Camel plugins

Note

We use the term `<Talend.runtime.dir>` for the directory where Talend Runtime is installed. This is typically the full path of either `Runtime_ESBSE` or `Talend-ESB-V5.1.x`, depending on the version of the software that is being used. Please substitute appropriately.

If you wish to make advanced configuration of the plugin, you need to modify the plugin files that are shipped with the release (all these configurations are optional):

1. Remove all the metrics you don't need to observe.

2. If you have several instances of Karaf or Tomcat, Hyperic will try to assign discovered Camel routes or CXF services to each Tomcat or Karaf instance. So you can set INSTALLPATH where the instance with Camel routes (CXF services) is actually situated. This will prevent assigning Camel routes (CXF services) to each Tomcat or Karaf running instance.

   For Tomcat, add to Apache Camel Tomcat 2.x server in the `camel-plugin.xml`

   ```
   <property name="TOMCAT_HOME" value="C:\tomcat"/>
   <property name="INSTALLPATH_MATCH" value="${TOMCAT_HOME}"/>
   ```

   Set INSTALLPATH to Tomcat instead of `C:\tomcat`

   For Tomcat, add to Apache CXF [Tomcat] 2.x server in the `cxf-plugin.xml`

   ```
   <property name="TOMCAT_HOME" value="C:\tomcat"/>
   <property name="INSTALLPATH_MATCH" value="${TOMCAT_HOME}"/>
   ```

   Set INSTALLPATH to Tomcat instead of `C:\tomcat`

   For Karaf, add to Apache Camel [Karaf] 2.x server in the `camel-plugin.xml`

   ```
   <property name="KARAF_HOME" value="<Talend.runtime.dir>\container\bin"/>
   <property name="INSTALLPATH_MATCH" value="${KARAF_HOME}"/>
   ```

   Set INSTALLPATH to bin directory of [Karaf] container instead of `<Talend.runtime.dir>\container\bin`

   For Karaf, add to Apache CXF [Karaf] 2.x server in the `cxf-plugin.xml`

   ```
   <property name="KARAF_HOME" value="<Talend.runtime.dir>\container\bin"/>
   <property name="INSTALLPATH_MATCH" value="${KARAF_HOME}"/>
   ```

   Set INSTALLPATH to bin directory of Karaf container instead of `<Talend.runtime.dir>\container\bin`

   Another variant, even easier, to solve this problem is to override `INSTALLPATH` property. As an example for Karaf and Tomcat:

   ```
   <property name="INSTALLPATH" value="Camel Karaf" />
   <property name="INSTALLPATH" value="Camel Tomcat" />
   <property name="INSTALLPATH" value="CXF Karaf" />
   <property name="INSTALLPATH" value="CXF Tomcat" />
   ```

3. Set `jmx.url` value, which is defined for Tomcat and Karaf. As a default, Tomcat uses: `service:jmx:rmi:///jndi/rmi://localhost:6969/jmxrmi/`

   and Karaf uses:

   ```
   service:jmx:rmi://localhost:44444/jndi/rmi://localhost:1099/karaf-trun
   ```

4. Change the property `AUTOINVENTORY_NAME` for the defined services in the plugin.

5. For Camel you can set `domain` option, (the default is `org.apache.camel`).

That's all you need. The plugin is ready to be used. Additional information about developing Hyperic JMX plugins can be found at http://support.hyperic.com/display/EVO/JMX+Plugin+Tutorial

3.2. Camel route and CXF service metrics

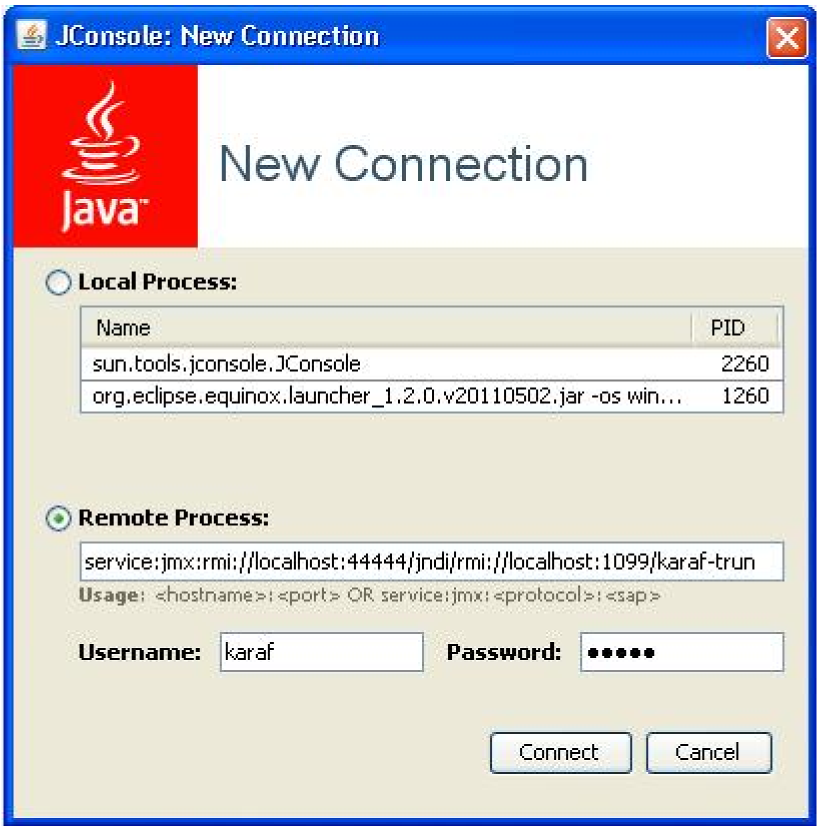
3.2.1. Hyperic metrics

In the case of JMX plugins, Hyperic monitors attributes of MBeans. The main metric attribute is `name`. The `name` or `alias` of the metric should be the same as MBean attribute name. To see the complete list of available CXF and Camel metrics, in addition to the Hyperic information, let's see how we can find Camel routes MBean attribute names using **jconsole** tool

You can find information about metrics at http://support.hyperic.com/display/EVO/Metric+Parameters

3.2.2. Defining metrics

1. Type **`jconsole`** from Command Line.



Name	PID
sun.tools.jconsole.JConsole	2260
org.eclipse.equinox.launcher_1.2.0.v20110502.jar -os win...	1260

Remote Process: service:jmx:rmi://localhost:44444/jndi/rmi://localhost:1099/karaf-trun

Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap>

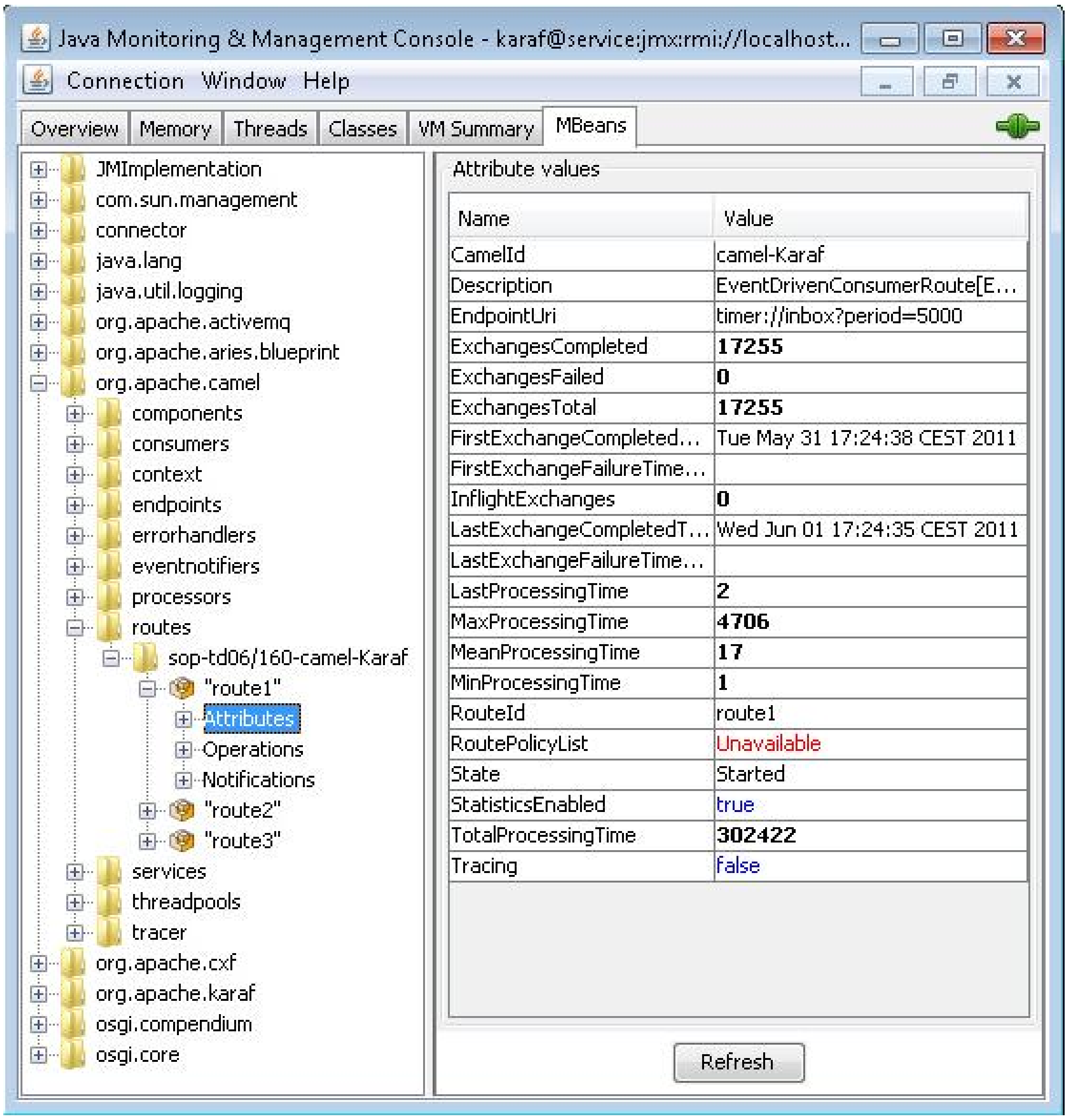
Username: karaf

Password: •••••

2. Set remote JMX URL by selecting remote process and click on **Connect**. This opens the Java Monitoring & Management console.

3. After connection, choose **MBeans** tab.

   For Camel you should see:

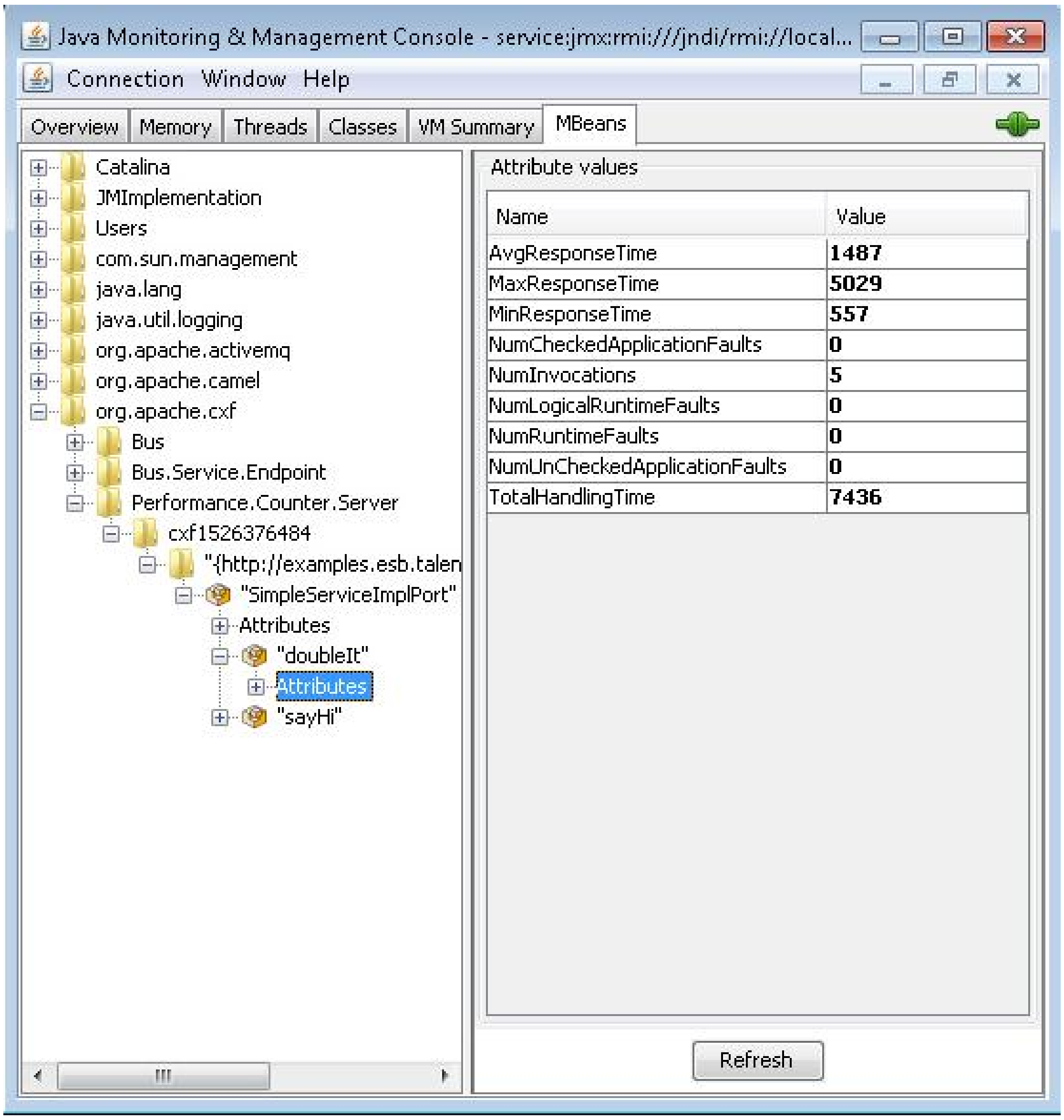


Name	Value
CamelId	camel-Karaf
Description	EventDrivenConsumerRoute[E...
EndpointUri	timer://inbox?period=5000
ExchangesCompleted	**17255**
ExchangesFailed	**0**
ExchangesTotal	**17255**
FirstExchangeCompleted...	Tue May 31 17:24:38 CEST 2011
FirstExchangeFailureTime...	
InflightExchanges	**0**
LastExchangeCompletedT...	Wed Jun 01 17:24:35 CEST 2011
LastExchangeFailureTime...	
LastProcessingTime	**2**
MaxProcessingTime	**4706**
MeanProcessingTime	**17**
MinProcessingTime	**1**
RouteId	route1
RoutePolicyList	Unavailable
State	Started
StatisticsEnabled	true
TotalProcessingTime	**302422**
Tracing	false

The attributes provided by Camel routes include:

- Exchanges Completed
- Exchanges Failed
- Exchanges Total
- Last Processing Time
- Max Processing Time
- Mean Processing Time
- Min Processing Time
- Total Processing Time

For CXF you should see:



Name	Value
AvgResponseTime	1487
MaxResponseTime	5029
MinResponseTime	557
NumCheckedApplicationFaults	0
NumInvocations	5
NumLogicalRuntimeFaults	0
NumRuntimeFaults	0
NumUnCheckedApplicationFaults	0
TotalHandlingTime	7436

The attributes provided by CXF services include:

- Availability
- Number of Invocations
- Total Handling Time
- Num Checked Application Faults
- Num Logical Runtime Faults
- Num Runtime Faults
- Num Unchecked Application Faults

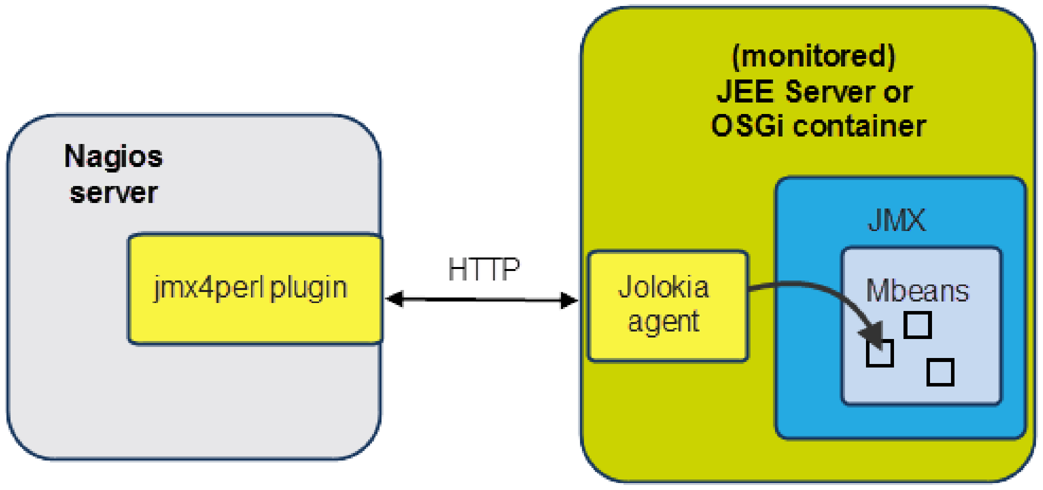
Chapter 4. Nagios Integration

Nagios is an Open Source monitoring application which allows users to identify infrastructure problems before they impact on important business processes. Nagios monitors the entire IT infrastructure to ensure servises, applications and business processes are working as expected.

In the case of critical changes in the infrastructure Nagios, can alert the IT department of the problem. That allows them to start fixing any issues as early as possible, before they affect the business processes.

In this chapter we describe how to monitor Talend ESB infrastructure using Nagios.

4.1. Architecture overview of Nagios and Talend ESB



Jmx4Perl provides an alternative way of accessing JMX (Java Management Extensions) on JEE Servers or OSGi containers. It uses an agent-based approach where a small Java Web application (Jolokia), is deployed on the application server, and provides HTTP/JSON-based access to JMX MBeans registered within the application server. Jolokia is based on a set of Perl modules, and does not need a local Java installation.

Component	Description
jmx4perl	This installed on the same machine as Nagios; it has a plugin which is integrated with the Talend ESB plugins. It also has a Perl library, JMX::Jmx4Perl, for programmatic JMX access.
check_jmx4perl	A command utility on the Nagios server that can be used to get the monitored data - it is part of the jmx4perl distribution.
Jolokia	Jolokia is an HTTP/JSON bridge for efficent remote JMX access. It is a separate agent that resides on the monitored server, and works with the jmx4perl plugin.

For convenience, Jolokia has been integrated into the Talend Runtime container as an OSGi agent and also integrated into ActiveMQ as a WAR agent.

More Information

For more information, see:

- Nagios: `http://www.nagios.com/documentation`
- Jolokia: `http://www.jolokia.org`
- Jmx4perl: `http://labs.consol.de/jmx4perl`

4.2. Installing jmx4perl plugin to Nagios

In this section we describe how to download and install the jmx4perl plugin to a Nagios server.

4.2.1. Prerequisites

Nagios Open Source version or Nagios XI 2011 version should be installed into Linux platform (or VM). To download Nagios, please go to `http://www.nagios.org/download`.

Note: Nagios Open Source 3.3.1 and Nagios XI 2011 version have been tested, but previous versions of Nagios may also work with Talend Runtime.

4.2.2. Downloading the jmx4perl plugin

Download the jmx4perl plugin from `http://search.cpan.org/CPAN/authors/id/R/RO/ROLAND/jmx4perl-1.04.tar.gz`.

4.2.3. Building the jmx4perl plugin

We use the Perl CPAN (Comprehensive Perl Archive Network) shell here to download missing dependencies.

1. Extract the jmx4perl distribution:

   ```
   $ cd /usr/local/src
   $ tar zxvf /tmp/jmx4perl-1.04.tar.gz
   $ ln -s -f jmx4perl-1.04 jmx4perl
   ```

2. Install the build module:

   ```
   $ cpan
   cpan[1]> install Module::Build
   cpan[1]> exit
   ```

3. There are two ways of doing this step:

   run './Build installdeps' (make sure you have ROOT rights)

   alternatively, you can explicitly install missing dependencies:

   ```
   $ cpan
   cpan[1]> install Config::General
   cpan[2]> install Crypt::Blowfish_PP
   cpan[3]> install File::SearchPath
   cpan[4]> install JSON
   cpan[5]> install Module::Find
   ```

```
cpan[6]> install Nagios::Plugin

cpan[7]> install Term::Clui

cpan[8]> install Term::ReadKey

cpan[9]> install Term::ReadLine::Perl

cpan[10]> install Term::ShellUI

cpan[11]> install Term::Size

cpan[12]> exit
```

4. Start the Build.PL script:

   ```
   # cd /usr/local/src/jmx4perl

   # perl Build.PL
   ```

   Running this script includes accepting the following defaults:

   ```
   Install 'jmx4perl' ? (y/n) [y ]y
   Install 'check_jmx4perl' ? (y/n) [y ]y
   Install 'cacti_jmx4perl' ? (y/n) [y ]y
   Install 'j4psh' ? (y/n) [y ]y
   Install Term::ReadLine::Gnu ? (y/n) [n ]n
   Install 'jolokia' ? (y/n) [y ]n
   ```

5. Run Build comand:

   ```
   # ./Build install
   ```

When all these steps finished successfully, the jmx4perl plugin should be installed onto Nagios. Please check this by running:

```
$ check_jmx4perl -u http://jolokia_host:8040/jolokia --alias MEMORY_HEAP_USED --base ME
```

In this example `jolokia_host` means the host where the Talend Runtime container was started and which has the Jolokia agent enabled and running.

4.3. Installing the Jolokia agent to a container

Jolokia is a HTTP/JSON bridge for efficent remote JMX access, and is a separate agent which was created as part of the evolution of jmx4perl.

For more information about Jolokia, see `http://www.Jolokia.org/`.

For convenience, Jolokia has been integrated into the Talend Runtime container as an OSGi agent and also integrated into ActiveMQ using a JAR file (see Section 4.1, “Architecture overview of Nagios and Talend ESB ”).

4.3.1. Installing the Jolokia OSGi agent to a Talend Runtime container

To install a Jolokia agent to a Talend Runtime container is simple; execute this command after starting the container:

```
features:install tesb-jmx-http-agent
```

Then, Jolokia agent bundle will be installed to the container. Run the **`list`** command, and the output should look like this:

```
[ 191] [Active] [       ] [       ] [   60] Jolokia Agent (1.0.2)
```

In addition, if you access the URL `http://localhost:8040/Jolokia/version`, you will see a JSON output line about version info, which indicates the Jolokia agent is running correctly.

4.3.2. The Jolokia agent and ActiveMQ

The Jolokia agent (JAR file) has been already integrated into the ActiveMQ distribution (included in Talend ESB). It's ready to use out of the box.

Go to <Talend.runtime.dir>/activemq/bin, and start ActiveMQ.

Then, if you access the URL `http://localhost:8161/Jolokia/version`, you will see a JSON output line about version info, which indicates the Jolokia agent is running correctly.

4.4. Nagios configuration templates

Now we look at configuring Nagios to select the metrics you wish to monitor.

4.4.1. Adding metrics for monitoring

Add the metrics for monitoring in three steps:

1. You should determine "check definitions" for jmx4perl plugin - these are the metrics that need to be monitored.
2. Define one or more commands for Nagios, that make use of the `check_jmx4perl` command from the plugin.
3. Describe a host and service definition for Nagios; the service definition needs to use the command defined in the previous step.

Here, we will see an example of using these steps - here we will define a metric for monitoring Active MQ:

1. First we describe a "check definition" for ActiveMQ in the configuration file `activemq.cfg`:

   ```
   # Define server connection parameters
   <Server tesb_activemq>
      Url = http://jolokia_host:8161/jolokia
   </Server>

   # checks for ActiveMQ metrics
   <Check Broker_TotalConsumerCount>
      MBean = org.apache.activemq:BrokerName=$0,Type=Broker
      Attribute =  TotalConsumerCount
   ```

```
   Name = TotalConsumerCount
   Warning 1000000
</Check>
```

In this sample, `jolokia_host` is the host that has the Jolokia agent installed, and is being monitored by the jmx4perl plugin.

2. Here is an example of a command definition, which is saved, for example, in jmx_commands.cfg:

```
# Define a command to monitor ActiveMQ using Nagios
# $USER5$ - user macros defining folder with check_jmx4perl
# $USER6$ - user macros defining folder with command configuration file
# $ARG1$ - MBean property name to be monitored
# $ARG2$ - set broker name for activemq to be monitored
# $ARG3$ - set destination for queue to be monitored
# $ARG4$ - set destination for topic to be monitored
define command {
   command_name check_jmx4perl_activemq
   command_line $USER5$/check_jmx4perl \
                --config $USER6$/activemq.cfg \
                --server $HOSTNAME$ \
                --check  $ARG1$ $ARG2$ $ARG3$ $ARG4$
}
```

   Note that the command definition uses the configuration file `activemq.cfg` which contains all the check definitions you created earlier.

   Several arguments are used in this command; their values are set later.

3. In following configuration example you can see how to describe the host and service definition for Nagios:

```
# Define a host
define host{
        use     activemq-host     ; Name of host template to use.
                                  ; This host definition will inherit
                                  ; all the variables that are defined
                                  ; in (or inherited by) the linux-server
                                  ; host template definition.
        host_name        tesb_activemq
        alias            tesb_activemq
        }

define service {
    use                     generic-service
    service_description     Broker_TotalConsumerCount
    display_name            Broker_TotalConsumerCount:
    check_interval          1
    host_name               tesb_activemq
    check_command           check_jmx4perl_activemq!Broker_TotalConsumerCount!
localhost!example.A!ActiveMQ.Advisory.Consumer.Queue.example.A
    }
```

Finally, note that you need to specify the values of the `check_command` properties in a strict order:

1. the name of command to check the metric
2. the name of check you use from jmx4perl configuration
3. the arguments for the command

4.4.2. List of configuration template files

There are four pre-defined configuration template files which ship with Talend ESB. These can be used for monitoring metrics of CXF, Camel and Activemq resources.

File	Description
jmx_commands.cfg	defines three commands for Nagios monitoring, including **check_jmx4perl_cxf**, **check_jmx4perl_camel** and **check_jmx4perl_activemq** commands. Each command has several macros which need to be defined in the `etc/resources.cfg` of Nagios.
cxf.cfg	check definition for cxf metrics to be monitored.
camel.cfg	check definition for camel metrics to be monitored.
activemq.cfg	check definition for activemq metrics to be monitored.

Also, there are three sample xxx_host.cfg configuration files which provided most of the useful metrics monitoring for CXF, Camel and Activemq. You can define your own xxx_host.cfg for monitoring specific metrics and specific resources(CXF services, Camel routes, etc.):

File	Description
cxf_host.cfg	sample configuration of host and service definition for CXF monitoring.
camel_host.cfg	sample configuration of host and service definition for Camel monitoring.
activemq_host.cfg	sample configuration of host and service definition for Activemq monitoring.

4.4.3. Using the configuration template files

Note

In these example, the Nagios installion directory is assumed to be `/usr/local/nagios` - please update if it is installed elsewhere.

You can use commands defined in `jmx_commands.cfg` file to monitor CXF services, Camel Context and Routes, ActiveMQ Broker, Topics and Queues.

In order to do it, you do not need to change template files `jmx_commands.cfg`, `cxf.cfg`, `camel.cfg`, `activemq.cfg` which already contain all check definitions and commands for these entities. We suggest you add your own `new_host.cfg` to monitor your own cxf service, camel route, and so on, using `cxf_host.cfg`, `camel_host.cfg`, `activemq_host.cfg` as samples.

1. Define `jolokia_host` in `/etc/hosts` - this name is used in subsequent files, rather than hard-coding in the ip address.

   For configuration templates `jolokia_host` means the host that has the Jolokia agent installed and would be monitored by the jmx4perl plugin. For example:

   ```
   192.168.1.101 jolokia_host
   ```

2. Put the configuration files into the folder `/usr/local/nagios/etc/objects/`

   In Talend Enterprise ESB Studio or Talend Enterprise ESB, the configuration files are in `<Talend.runtime.dir>/add-ons/adapters/nagios`.

Copy template and sample configuration files from this directory into `/usr/local/nagios/etc/objects/`.

```
cp -f <Talend.runtime.dir>/add-ons/adapters/nagios/template/*.cfg /usr/local/nagios/etc/objects/
```

```
cp -f <Talend.runtime.dir>/add-ons/adapters/nagios/sample/*.cfg /usr/local/nagios/etc/objects/
```

3. Add the command configuration file to the `/usr/local/nagios/etc/nagios.cfg` file by adding this line in it:

```
cfg_file=/usr/local/nagios/etc/objects/jmx_commands.cfg
```

4. Create host definitions file, for example `new_host.cfg`. Add it to the `/usr/local/nagios/etc/nagios.cfg` file, file by adding this line in it:

```
cfg_file=/usr/local/nagios/etc/objects/new_host.cfg
```

5. Define macros which will be used by `jmx_commands.cfg` in the `/usr/local/nagios/etc/resource.cfg` file by adding this line in it:

```
# set the path which jmx4perl plugin installed
$USER5$=/usr/local/src/jmx4perl/scripts
# set the path to where to find configuration files
$USER6$=/usr/local/nagios/etc/objects
```

6. Then, restart Nagios for the changes to take effect.

```
service nagios restart
```

4.5. Examples: monitoring cxf-jmx and camel-jmx sample applications

There are two sample applications that ship with the TESB release: `cxf-jmx` and `camel-jmx`.

These can be built and installed on Windows or Linux.

4.5.1. Build and install the cxf-jmx sample applications

This sample consists of four parts:

Directory	Description
service/	the CXF web service provider packaged as an OSGi bundle
client/	a sample client application that uses the CXF JAX-WS API to create a SOAP client and make several calls with it.

Directory	Description
common/	code that is common for both the client and the server.
war/	A WAR archive containing code from common and service modules. This is for Servlet container use only, not used in OSGi deployment.

1. From the example parent directory (<Talend.runtime.dir>/examples/talend/tesb), run the following command to install the example parent `pom.xml` file into the local maven repo

   `mvn install --non-recursive`

2. From the base directory of the sample, the maven `pom.xml` file can be used to build and run the demo

   `mvn install`

3. Start Talend Runtime container:

   `trun.sh` (on Linux)

   `trun.bat` (on Windows)

4. Add cxf-jmx example features URL. Type this command in Talend Runtime container:

   `features:addurl mvn:org.talend.esb.examples/cxf-jmx-feature/5.1.0/xml`

5. Install cxf-jmx example feature into the Talend Runtime container

   `features:install cxf-jmx-service`

6. You can find wsdl at `http://localhost:8040/services/simpleService?wsdl`

7. Now run the client; from cxf-jmx folder run:

   `mvn exec:java -pl client`

After SOAP calls on the web service have been done, you'll see the `Performance.Counter.Server` folder, where CXF MBeans with their attributes will be listed (see Section 3.2.2, “Defining metrics” for more details).

4.5.2. Build and install the camel-jmx sample applications

This sample consists of two parts:

1. service/ - This is the CXF service packaged as an OSGi bundle.
2. war/ - This module creates a WAR archive containing the service module. This is for Servlet container use only, not used in OSGi deployment.

From the base directory of this sample, the Maven pom.xml file can be used to build and run the demo

1. **`mvn clean install`**

   Running this command will build the demo and create a WAR archive and an OSGi bundle for deploying the service either to servlet or OSGi containers

2. Start Talend Runtime container

   `trun.sh` (on Linux)

   `trun.bat` (on Windows)

3. Add camel-jmx example features URL. Type this command in Talend Runtime container:

   ```
   features:addurl mvn:org.talend.esb.examples/camel-jmx-feature/5.1.0/xml
   ```

4. To install example ferture type command in Talend Runtime container:

   ```
   features:install camel-jmx-service
   ```

After deploying the samples you can see the Camel MBeans and their attributes which can be monitored using the JDK's JConsole. Attributes are also included in the metrics that we will monitor with help of Nagios. (See Section 3.2.2, “Defining metrics”)

4.5.3. Configure the plugin to monitor the sample applications

Note: in Talend Enterprise ESB Studio or Talend Enterprise ESB, the configuration files are in `<Talend.runtime.dir>/add-ons/adapters/nagios`.

1. Define jolokia_host in `/etc/hosts` - this name is used in subsequent files, rather than hard-coding in the ip address. For example, add the line:

   ```
   192.168.1.101 jolokia_host
   ```

2. Copy template and sample configuration files into Nagios etc folder

   ```
   cp -f <Talend.runtime.dir>/add-ons/adapters/nagios/template/*.cfg /usr/local/nagios/etc/objects/
   ```

   ```
   cp -f <Talend.runtime.dir>/add-ons/adapters/nagios/sample/*.cfg /usr/local/nagios/etc/objects/
   ```

3. Add template configuration file to the Nagios configuration file /usr/local/nagios/etc/nagios.cfg

   ```
   cfg_file=/usr/local/nagios/etc/objects/jmx_commands.cfg.cfg
   ```

4. Add sample configuration files to the Nagios configuration file /usr/local/nagios/etc/nagios.cfg

   ```
   cfg_file=/usr/local/nagios/etc/objects/cxf_host.cfg
   cfg_file=/usr/local/nagios/etc/objects/activemq_host.cfg
   cfg_file=/usr/local/nagios/etc/objects/camel_host.cfg
   ```

5. Define macros which will be used by `jmx_commands.cfg` in the `/usr/local/nagios/etc/resource.cfg`. Edit `/usr/local/nagios/etc/resource.cfg`

   ```
   # set the path which jmx4perl plugin installed
   $USER5$=/usr/local/src/jmx4perl/scripts
   # set the path to where to find configuration files
   $USER6$=/usr/local/nagios/etc/objects
   ```

6. Restart Nagios

   ```
   service nagios restart
   ```

4.5.4. Monitoring with Nagios

Login to the Nagios Web Interface `http://<nagios_host>/nagios/`

There, you will find the status of metrics for cxf-jmx and camel-jmx examples.

4.6. Resources and metrics that are being monitored

Here is a complete list of the default metrics for CXF, Camel and Activemq that are being monitored in Talend ESB. More detailed definitions can be found in `template/cxf.cfg`, `template/camel.cfg` and `template/activemq.cfg`.

4.6.1. CXF services metrics

Name	**MBean**	**Attribute**
NumInvocations	org.apache.cxf:bus.id= *,type=Performance.Counter.Server, service="$1",port="$0"	NumInvocations
TotalHandlingTime	org.apache.cxf:bus.id= *,type=Performance.Counter.Server, service="$1",port="$0"	TotalHandlingTime
NumCheckedApplicationFaults	org.apache.cxf:bus.id=*,type= Performance.Counter.Server, service="$1",port="$0"	NumCheckedApplicationFaults
NumLogicalRuntimeFaults	org.apache.cxf:bus.id=*,type= Performance.Counter.Server, service="$1",port="$0"	NumLogicalRuntimeFaults
NumRuntimeFaults	org.apache.cxf:bus.id=*,type= Performance.Counter.Server, service="$1",port="$0"	NumRuntimeFaults
NumUnCheckedApplicationFaults	org.apache.cxf:bus.id=*,type= Performance.Counter.Server, service="$1",port="$0"	NumUnCheckedApplicationFaults

4.6.2. Camel routes/contexts metrics

Name	**MBean**	**Attribute**
Context_InflightExchanges	org.apache.camel:context=*,type=context,name="$0"	InflightExchanges
Context_Uptime	org.apache.camel:context=*,type=context,name="$0"	Uptime
Route_ExchangesCompleted	org.apache.camel:context=*,type=routes,name="$1"	ExchangesCompleted
Route_ExchangesFailed	org.apache.camel:context=*,type=routes,name="$1"	ExchangesFailed
Route_ExchangesTotal	org.apache.camel:context=*,type=routes,name="$1"	ExchangesTotal

Name	MBean	Attribute
Route_LastProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	LastProcessingTime
Route_MaxProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	MaxProcessingTime
Route_MinProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	MinProcessingTime
Route_MeanProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	MeanProcessingTime
Route_TotalProcessingTime	org.apache.camel:context=*,type=routes,name="$1"	TotalProcessingTime

4.6.3. ActiveMQ queues/topics metrics

Name	MBean	Attribute
Broker_TotalConsumerCount	org.apache.activemq:BrokerName= $0,Type=Broker	TotalConsumerCount
Broker_TotalDequeueCount	org.apache.activemq:BrokerName= $0,Type=Broker	TotalDequeueCount
Broker_TotalEnqueueCount	org.apache.activemq:BrokerName= $0,Type=Broker	TotalEnqueueCount
Broker_Uptime	org.apache.activemq:BrokerName= $0,Type=Broker	Uptime
Broker_TotalMessageCount	org.apache.activemq:BrokerName= $0,Type=Broker	TotalMessageCount
Broker_MemoryPercentUsage	org.apache.activemq:BrokerName= $0,Type=Broker	MemoryPercentUsage
Broker_StorePercentUsage	org.apache.activemq:BrokerName= $0,Type=Broker	StorePercentUsage
Broker_TempPercentUsage	org.apache.activemq:BrokerName= $0,Type=Broker	TempPercentUsage
Queue_ConsumerCount	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	ConsumerCount
Queue_DequeueCount	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	DequeueCount
Queue_DispatchCount	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	DispatchCount
Queue_EnqueueCount	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	EnqueueCount
Queue_ExpiredCount	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	ExpiredCount
Queue_InFlightCount	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	InFlightCount
Queue_MaxEnqueueTime	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	MaxEnqueueTime
Queue_MemoryPercentUsage	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	MemoryPercentUsage
Queue_QueueSize	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	QueueSize
Queue_ProducerCount	org.apache.activemq:BrokerName= $0,Type=Queue,Destination=$1	ProducerCount

Name	MBean	Attribute
Topic_AverageEnqueueTime	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	AverageEnqueueTime
Topic_ConsumerCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	ConsumerCount
Topic_DequeueCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	DequeueCount
Topic_DispatchCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	DispatchCount
Topic_EnqueueCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	EnqueueCount
Topic_ExpiredCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	ExpiredCount
Topic_InFlightCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	InFlightCount
Topic_MaxEnqueueTime	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	MaxEnqueueTime
Topic_MemoryPercentUsage	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	MemoryPercentUsage
Topic_MinEnqueueTime	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	MinEnqueueTime
Topic_ProducerCount	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	ProducerCount
Topic_QueueSize	org.apache.activemq:BrokerName=$0,Type=Topic,Destination=$2	QueueSize

4.7. Metric criteria

This topic explain a metric criteria for CXF, Camel and ActiveMQ in Talend ESB. All states for Nagios checks are divided on OK, WARNING, CRITICAL and UNKNOWN.For additional information about Nagios states read the State types in Nagios documentation.

All Fault metrics throwing state WARNING if 1 fault occured and CRITICAL state if 100 faults occured.Multicheck AnyFaults is used for fault status indication.

All countable metrics have informational nature and throwing warning if 1000000 reached. It can be tuned for specific needs.

All memory usage metrics throwing Warning when 80 percents of memory used and Critical if 90 percents.

Here is a complete metric criteria table:

4.7.1. CXF services metrics criteria

Name	Criterias
NumInvocations	Critical 6000, Warning 5000
TotalHandlingTime	Critical 6000000, Warning 5000000

Name	**Criterias**
NumCheckedApplicationFaults	Critical 100, Warning 1
NumLogicalRuntimeFaults	Critical 100, Warning 1
NumRuntimeFaults	Critical 100, Warning 1
NumUnCheckedApplicationFaults	Critical 100, Warning 1

4.7.2. Camel routes/contexts metrics criteria

Name	**Criterias**
Context_InflightExchanges	Critical 20, Warning 10
Context_Uptime	Critical 6000000, Warning 5000000
Route_ExchangesCompleted	Critical 6000000, Warning 5000000
Route_ExchangesFailed	Critical 100, Warning 1
Route_ExchangesTotal	Critical 6000000, Warning 5000000
Route_LastProcessingTime	Critical 100, Warning 20
Route_MaxProcessingTime	Critical 100, Warning 20
Route_MinProcessingTime	Critical 20, Warning 10
Route_MeanProcessingTime	Critical 100, Warning 20
Route_TotalProcessingTime	Critical 6000000, Warning 5000000

4.7.3. ActiveMQ queues/topics metrics criteria

Name	**Criterias**
Broker_TotalConsumerCount	Warning 1000000
Broker_TotalDequeueCount	Warning 1000000
Broker_TotalEnqueueCount	Warning 1000000
Broker_Uptime	Warning 1000000
Broker_TotalMessageCount	Warning 1000000
Broker_MemoryPercentUsage	Critical 90, Warning 80
Broker_StorePercentUsage	Critical 90, Warning 80
Broker_TempPercentUsage	Critical 90, Warning 80
Queue_ConsumerCount	Warning 1000000
Queue_DequeueCount	Warning 1000000
Queue_DispatchCount	Warning 1000000
Queue_EnqueueCount	Warning 1000000
Queue_ExpiredCount	Critical 20, Warning 10
Queue_InFlightCount	Critical 20, Warning 10
Queue_MaxEnqueueTime	Critical 400, Warning 200
Queue_MemoryPercentUsage	Critical 100, Warning 80
Queue_QueueSize	Warning 80000

Name	**Criterias**
Queue_ProducerCount	Warning 1000000
Topic_AverageEnqueueTime	Warning 180, Critical 400
Topic_ConsumerCount	Warning 1000000
Topic_DequeueCount	Warning 1000000
Topic_DispatchCount	Warning 1000000
Topic_EnqueueCount	Warning 1000000
Topic_ExpiredCount	Critical 40, Warning 10
Topic_InFlightCount	Warning 1000
Topic_MaxEnqueueTime	Critical 100,Warning 40
Topic_MemoryPercentUsage	Critical 90, Warning 80
Topic_MinEnqueueTime	Warning 200
Topic_ProducerCount	Warning 1000000
Topic_QueueSize	Warning 1000