Fuse ESB Enterprise

Configuring and Running Fuse ESB Enterprise

Version 7.1
December 2012

Integration Everywhere

Configuring and Running Fuse ESB Enterprise

Version 7.1

Updated: 22 Feb 2013

Trademark Disclaimer

Third Party Acknowledgements

One or more products in the Fuse ESB Enterprise release includes third party components covered by licenses that require that the following documentation notices be provided:

- JLine (http://jline.sourceforge.net) jline:jline:jar:1.0

  License: BSD (LICENSE.txt) - Copyright (c) 2002-2006, Marc Prud'hommeaux <mwp1@cornell.edu>

  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
  - Neither the name of JLine nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- Stax2 API (http://woodstox.codehaus.org/StAX2) org.codehaus.woodstox:stax2-api:jar:3.1.1

  License: The BSD License (http://www.opensource.org/licenses/bsd-license.php)

  Copyright (c) <YEAR>, <OWNER> All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- jibx-run - JiBX runtime (http://www.jibx.org/main-reactor/jibx-run) org.jibx:jibx-run:bundle:1.2.3

  License: BSD (http://jibx.sourceforge.net/jibx-license.html) Copyright (c) 2003-2010, Dennis M. Sosnoski.

  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
  - Neither the name of JiBX nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- JavaAssist (http://www.jboss.org/javassist) org.jboss.javassist:com.springsource.javassist:jar:3.9.0.GA:compile

  License: MPL (http://www.mozilla.org/MPL/MPL-1.1.html)

- HAPI-OSGI-Base Module (http://hl7api.sourceforge.net/hapi-osgi-base/) ca.uhn.hapi:hapi-osgi-base:bundle:1.2

  License: Mozilla Public License 1.1 (http://www.mozilla.org/MPL/MPL-1.1.txt)

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Configuring the Initial Features in a Standalone Container

If you are using a standalone container, you can change the features it automatically loads the first time it is started.

Overview

The *first* time you start a standalone container, the container looks in the `etc/org.apache.karaf.features.cfg` file to discover the feature URLs (feature repository locations) and to determine which features it will load. By default, Fuse ESB Enterprise loads a large number of features and you may not need all of them. You may also decide you need features that are not included in the default configuration.

Fabric Containers

The features loaded by a Fabric Container are controlled by the container's profiles. Changing the values as described below will have no effect on a Fabric container.

The values in `etc/org.apache.karaf.features.cfg` are only used the *first* time the container is started. On subsequent start-ups, the container uses the contents of the *`InstallDir`*`/data` directory to determine what to load. If you need to adjust the features loaded into a container, you can delete the `data` directory, but this will also destroy any state or persistence information stored by the container.

For more on features and how they are used in Fuse ESB Enterprise, see "Deploying Features" in *Deploying into the Container*.

Modifying the default installed features

By default, Fuse ESB Enterprise installs a large number of features, including some that you may not want to deploy.

You can change the initial set of installed features by editing the `featuresBoot` property.

Modifying the default set of feature URLs

Fuse ESB Enterprise registers a number of URLs that point to feature repositories on start-up. You can change the initial set of feature URLs by editing the `featureRepositories` property.

Chapter 2. Installing Fuse ESB Enterprise as a Service

The Fuse ESB Enterprise installer generates a service wrapper that can be easily configured to install Fuse ESB Enterprise as a system service.

Installing Fuse ESB Enterprise as a system service is a two step process:

1. Configure on page 16 the service wrapper for your system.
2. Install on page 20 the service wrapper as system service.

Configuring the Wrapper

Overview

The service wrapper is configured by the *ServiceName*-wrapper.conf file, which is located under the *InstallDir*/etc/ directory.

There are several settings you may want to change including:

- the default environment settings
- the properties passed to the JVM
- the classpath
- the JMX settings
- the logging settings

Specifying the Fuse ESB Enterprise's environment

A broker's environment is controlled by three environment variables:

- KARAF_HOME—the location of the Fuse ESB Enterprise install directory.
- KARAF_BASE—the root directory containing the configuration and OSGi data specific to the broker instance.

  The configuration for the broker instance is stored in the *KARAF_BASE*/conf directory. Other data relating to the OSGi runtime is also stored beneath the base directory.
- KARAF_DATA—the directory containing the logging and persistence data for the broker.

Example 2.1 on page 17 shows the default values.

Example 2.1. Default Environment Settings

```
set.default.KARAF_HOME=InstallDir
set.default.KARAF_BASE=InstallDir
set.default.KARAF_DATA=InstallDir\data
```

Passing parameters to the JVM

If you want to pass parameters to the JVM, you do so by setting wrapper properties using the form `wrapper.java.additional.<n>`. *<n>* is a sequence number that must be distinct for each parameter.

One of the most useful things you can do by passing additional parameters to the JVM is to set Java system properties. The syntax for setting a Java system property is `wrapper.java.additional.<n>=-DPropName=PropValue`.

Example 2.2 on page 17 shows the default Java properties.

Example 2.2. Default Java System Properties

```
# JVM
# note that n is the parameter number starting from 1.
wrapper.java.additional.1=-Dkaraf.home="%KARAF_HOME%"
wrapper.java.additional.2=-Dkaraf.base="%KARAF_BASE%"
wrapper.java.additional.3=-Dkaraf.data="%KARAF_DATA%"
wrapper.java.additional.4=-Dcom.sun.managment.jmxremote
wrapper.java.additional.5=-Dkaraf.startLocalConsole=false
wrapper.java.additional.6=-Dkaraf.startRemoteShell=true
wrapper.java.additional.7=-Djava.endorsed.dirs="%JAVA_HOME%/jre/lib/en
dorsed;%JAVA_HOME%/lib/endorsed;%KARAF_HOME%/lib/endorsed"
wrapper.java.additional.8=-
Djava.ext.dirs="%JAVA_HOME%/jre/lib/ext;%JAVA_HOME%/lib/ext;%KARAF_HOME%/lib/ext"
```

Adding classpath entries

You add classpath entries using the syntax `wrapper.java.classpath.<n>`. *<n>* is a sequence number that must be distinct for each classpath entry.

Example 2.3 on page 17 shows the default classpath entries.

Example 2.3. Default Wrapper Classpath

```
wrapper.java.classpath.1=%KARAF_BASE%/lib/karaf-wrapper.jar
wrapper.java.classpath.2=%KARAF_HOME%/lib/karaf.jar
```

```
wrapper.java.classpath.3=%KARAF_HOME%/lib/karaf-jaas-boot.jar
wrapper.java.classpath.4=%KARAF_BASE%/lib/karaf-wrapper-main.jar
```

JMX configuration

The default service wrapper configuration does not enable JMX. It does, however, include template properties for enabling JMX. To enable JMX:

1. Locate the line `# Uncomment to enable jmx`.

   There are three properties, shown in Example 2.4 on page 18, that are used to configure JMX.

   Example 2.4. Wrapper JMX Properties

   ```
   # Uncomment to enable jmx
   #wrapper.java.additional.n=-Dcom.sun.management.jmxre
   mote.port=1616
   #wrapper.java.additional.n=-Dcom.sun.management.jmxre
   mote.authenticate=false
   #wrapper.java.additional.n=-Dcom.sun.management.jmxre
   mote.ssl=false
   ```

2. Remove the `#` from in front of each of the properties.

3. Replace the `n` in each property to a number that fits into the sequence of addition properties established in the configuration.

You can change the settings to use a different port or secure the JMX connection.

For more information about using JMX see "Configuring JMX" on page 83.

Configuring logging

The wrapper's logging in configured using the properties described in Table 2.1 on page 18.

Table 2.1. Wrapper Logging Properties

Property	Description
`wrapper.console.format`	Specifies how the logging information sent to the console is formated. The format consists of the following tokens: • `L`—log level

Property	Description
	• P—prefix • D—thread name • T—time • Z—time in milliseconds • U—approximate uptime in seconds (based on internal tick counter) • M—message
wrapper.console.loglevel	Specifies the logging level displayed on the console.
wrapper.logfile	Specifies the file used to store the log.
wrapper.logfile.format	Specifies how the logging information sent to the log file is formated.
wrapper.console.loglevel	Specifies the logging level sent to the log file.
wrapper.console.maxsize	Specifies the maximum size, in bytes, that the log file can grow to before the log is archived. The default value of 0 disables log rolling.
wrapper.console.maxfiles	Specifies the maximum number of archived log files which will be allowed before old files are deleted. The default value of 0 implies no limit.
wrapper.syslog.loglevel	Specifies the logging level for the sys/event log output.

For more information about Fuse ESB Enterprise logging see "Logging" on page 93.

Installing and Starting the Service

Overview

The operating system determines the exact steps using to complete the installation of Fuse ESB Enterprise as a service. The **wrapper:install** command provides basic instructions for your operating system.

Windows

To install the service run ***InstallDir*\bin*ServiceName*-service.bat install**. If you used the default start setting, the service will start when Windows is launched. If you specified `DEMAND_START`, you will need to start the service manually.

To start the service manually run **net start "*ServiceName*"**. You can also use the Windows service UI.

To manually stop the service run **net stop "*ServiceName*"** You can also use the Windows service UI.

You remove the installed the service by running ***InstallDir*\bin*ServiceName*-service.bat remove**.

Redhat Linux

To install the service and configure it to start when the machine boots, run the following commands:

```
# ln -s InstallDir\bin\ServiceName-service /etc/init.d/
# chkconfig ServiceName-service --add
# chkconfig ServiceName-service on
```

To start the service manually run **service *ServiceName*-service start**.

To manually stop the service run **service *ServiceName*-service stop**.

You remove the installed the service by running the following commands:

```
#service ServiceName-service stop
# chkconfig ServiceName-service --del
# rm /etc/init.d/ServiceName-service
```

Ubuntu Linux

To install the service and configure it to start when the machine boots, run the following commands:

```
# ln -s InstallDir\bin\ServiceName-service /etc/init.d/
# update-rc.d ServiceName-service defaults
```

To start the service manually run **/etc/init.d/*ServiceName*-service start**.

To manually stop the service run **/etc/init.d/*ServiceName*-service stop**.

You remove the installed the service by running the following commands:

```
#/etc/init.d/ServiceName-service stop
# rm /etc/init.d/ServiceName-service
```

Chapter 3. Basic Security

This chapter describes the basic steps to configure security before you start Fuse ESB Enterprise for the first time. By default, Fuse ESB Enterprise is secure, but none of its services are remotely accessible. This chapter explains how to enable secure access to the ports exposed by Fuse ESB Enterprise.

Configuring Basic Security

Overview

The Fuse ESB Enterprise runtime is secured against network attack by default, because all of its exposed ports require user authentication and no users are defined initially. In other words, the Fuse ESB Enterprise runtime is remotely inaccessible by default.

If you want to access the runtime remotely, you must first customize the security configuration, as described here.

Before you start the container

If you want to enable remote access to the Fuse ESB Enterprise container, perform the following configuration steps before starting the container:

1. "Create a secure JAAS user" .
2. "Configure the Apache ActiveMQ Web console (optional)" .

Create a secure JAAS user

By default, no JAAS users are defined for the container, which effectively disables remote access (it is impossible to log on).

To create a secure JAAS user, edit the *InstallDir*/etc/users.properties file and add a new user field, as follows:

```
Username=Password,admin
```

Where *Username* and *Password* are the new user credentials. The admin role gives this user the privileges to access all administration and management functions of the container. For more details about JAAS, see "Configuring JAAS Security" on page 85.

Warning

It is strongly recommended that you define custom user credentials with a strong password.

Configure the Apache ActiveMQ Web console (optional)

If you want to access the Apache ActiveMQ Web console (for remote administration of JMS messaging), you must provide the Web console servlet with the credentials it needs to login to the JMS broker.

Edit the *InstallDir*/etc/system.properties file and modify the webconsole.jmx.* and webconsole.jms.* properties as follows:

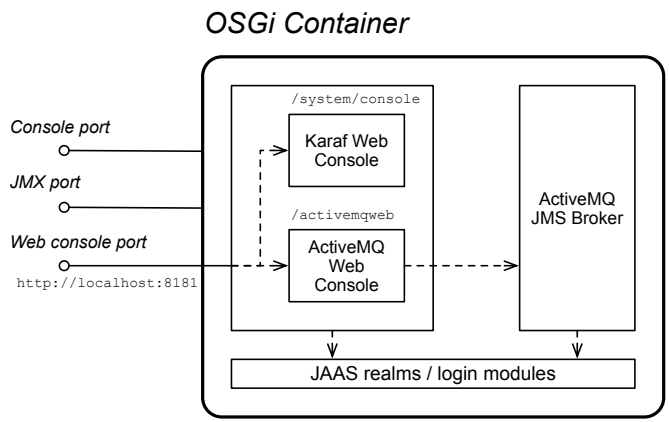
```
webconsole.jmx.user=Username
webconsole.jmx.password=Password
webconsole.jms.user=Username
webconsole.jms.password=Password
```

Where the *Username* and *Password* are the credentials of a JAAS user with admin privileges.

Ports exposed by the Fuse ESB Enterprise container

Figure 3.1 on page 25 shows the ports exposed by the Fuse ESB Enterprise container by default.

Figure 3.1. Ports Exposed by the Fuse ESB Enterprise Container



The following ports are exposed by the container:

- *Console port*—enables remote control of a container instance, through Apache Karaf shell commands. This port is enabled by default and is secured both by JAAS authentication and by SSL.
- *JMX port*—enables management of the container through the JMX protocol. This port is enabled by default and is secured by JAAS authentication.

- *Web console port*—provides access to an embedded Jetty container that can host Web console servlets. By default, the Web consoles are *not* installed in the container.

Enabling the remote console port

You can access the remote console port whenever both of the following conditions are true:

- JAAS is configured with at least one set of login credentials.
- The Fuse ESB Enterprise runtime has *not* been started in client mode (client mode disables the remote console port completely).

For example, to log on to the remote console port from the same machine where the container is running, enter the following command:

```
./client -u Username -p Password
```

Where the `Username` and `Password` are the credentials of a JAAS user with `admin` privileges. For more details, see "Using Remote Connections to Manage a Container" on page 43.

Strengthening security on the remote console port

You can employ the following measures to strengthen security on the remote console port:

- Make sure that the JAAS user credentials have strong passwords.
- Customize the X.509 certificate (replace the Java keystore file, `InstallDir/etc/host.key`, with a custom key pair).

Enabling the JMX port

The JMX port is enabled by default and secured by JAAS authentication. In order to access the JMX port, you must have configured JAAS with at least one set of login credentials. To connect to the JMX port, open a JMX client (for example, `jconsole`) and connect to the following JMX URI:

```
service:jmx:rmi:///jndi/rmi://localhost:1099/karaf-root
```

You must also provide valid JAAS credentials to the JMX client in order to connect.

Note

In general, the tail of the JMX URI has the format `/karaf-ContainerName`. If you change the container name from `root` to some other name, you must modify the JMX URI accordingly.

Enabling the Web console port

All of the Web consoles are installed as servlets in the container's embedded Jetty container. The Web consoles share the same HTTP server port, which is powered by Jetty[1]. You can optionally enable the following Web consoles in Fuse ESB Enterprise:

- *Karaf Web console*—is not installed by default. To enable the Karaf Web console, perform the following steps:

  1. In a running Fuse ESB Enterprise instance (see "Starting Fuse ESB Enterprise" on page 32), enter the following console command:

     ```
     karaf@root> features:install webconsole
     ```

  2. In a Web browser, navigate to the following URL:

     ```
     http://localhost:8181/system/console
     ```

  3. The browser will prompt you to log on. Enter valid JAAS user credentials to access the console.

- *Apache ActiveMQ Web console*—is not installed by default. To enable the Apache ActiveMQ Web console, perform the following steps:

  1. Make sure that you have already configured the container's Java system properties, as described in "Configure the Apache ActiveMQ Web console (optional)" on page 24.

     ## Note

     The Apache ActiveMQ Web console has a three tier architecture, as shown in Figure 3.1 on page 25. The credentials provided in this step enable the middle tier (the Web console servlet) to log on to the back-end tier (the Apache ActiveMQ broker).

[1] http://jetty.codehaus.org/jetty/

2. In a running Fuse ESB Enterprise instance, enter the following console command:

```
karaf@root> features:install mq-web-console
```

3. In a Web browser, navigate to the following URL:

```
http://localhost:8181/activemqweb
```

4. The browser will prompt you to log on. Enter valid JAAS user credentials to access the console.

Strengthening security on the Web console port

The Karaf Web console is already secured by JAAS authentication. To add SSL security, see "Securing the Web Console" in *Security Guide*.

Disabling Broker Security

Overview

Prior to Fuse ESB Enterprise version 7.0.2, the Apache ActiveMQ broker was insecure (JAAS authentication not enabled). This section explains how to revert the Apache ActiveMQ broker to an insecure mode of operation, so that it is unnecessary to provide credentials when connecting to the broker.

Warning

After performing the steps outlined in this section, the broker has no protection against hostile clients. This type of configuration is suitable only for use on internal, trusted networks.

Standalone server

These instructions assume that you are running Fuse ESB Enterprise in standalone mode (that is, running in an OSGi container, but not using Fuse Fabric). In your installation of Fuse ESB Enterprise, open the *InstallDir*`/etc/activemq.xml` file using a text editor and look for the following lines:

```
...
<plugins>
    <jaasAuthenticationPlugin configuration="karaf" />
</plugins>
...
```

To disable JAAS authentication, delete (or comment out) the `jaasAuthenticationPlugin` element. The next time you start up the Fuse ESB Enterprise container (using the *InstallDir*`/bin/fusemq` script), the broker will run with unsecured ports.

Chapter 4. Starting and Stopping Fuse ESB Enterprise

Fuse ESB Enterprise provides simple command-line tools for starting and stopping the server.

Starting Fuse ESB Enterprise

Overview

The default way for deploying the Fuse ESB Enterprise runtime is to deploy it as a standalone server with an active console. You can also deploy the runtime to run as a background process without a console.

Setting up your environment

You can start the Fuse ESB Enterprise runtime from the installation directory without doing any work. However, if you want to start it in a different folder you will need to add the `bin` directory of your Fuse ESB Enterprise installation to the `PATH` environment variable, as follows:

Windows

```
set PATH=%PATH%;InstallDir\bin
```

***NIX**

```
export PATH=$PATH,InstallDir/bin
```

Launching the runtime

If you are launching the Fuse ESB Enterprise runtime from the installation directory use the following command:

Windows

```
bin\fuseesb.bat
```

***NIX**

```
bin/fuseesb
```

If Fuse ESB Enterprise starts up correctly you should see the following on the console:

```
|  ___|                 |  ___|/  ___|| ___ \
| |_  _   _  ___   ___  | |__  \ `--. | |_/ /
|  _|| | | |/ __| / _ \ |  __|  `--. \|  __ \
| |  | |_| |\__ \|  __/ | |___ /\__/ /| |_/ /
\_|   \__,_||___/ \___| \____/ \____/ \____/

  Fuse ESB (7.1.0.fuse-047)
  http://fusesource.com/products/fuse-esb-enterprise/

Hit '<tab>' for a list of available commands
```

```
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown Fuse ESB.

FuseESB:karaf@root>
```

Launching the runtime in server mode

Launching in server mode runs Fuse ESB Enterprise in the background, without a local console. You would then connect to the running instance using a remote console. See " Connecting and Disconnecting Remotely" on page 45 for details.

To launch Fuse ESB Enterprise in server mode, run the following

Windows

```
bin\fuseesb.bat server
```

***NIX**

```
bin/fuseesb server
```

Alternatively, you can launch Fuse ESB Enterprise in server mode using the `start` script in the *`InstallDir`*`/bin` directory.

Launching the runtime in client mode

In production environments you may want to have a runtime instance accessible using only a local console. In other words, *you cannot connect to the runtime remotely*. You can do this by launching the runtime in client mode using the following command:

Windows

```
bin\fuseesb.bat client
```

***NIX**

```
bin/fuseesb client
```

Stopping Fuse ESB Enterprise

You can stop an instance of Fuse ESB Enterprise either from within a console, or using a `stop` script.

Stopping an instance from a local console

If you launched Fuse ESB Enterprise by running **`fuseesb`** or **`fuseesb client`**, you can stop it by doing one of the following at the `karaf>` prompt:

- Type simply **`shutdown -f`**
- Press **Ctrl**+**D**

Stopping an instance running in server mode

If you launched Fuse ESB Enterprise by running **`fuseesb server`** or by running the `start` script, you can stop it remotely, as described in "Stopping a Remote Container" on page 57.

Alternatively, you can log on to the host where the instance is running and run one of the following from the *`InstallDir`*`/bin` directory:

- **`./admin stop`** *`instanceName`*
- **`./stop`**

Note

If the `sshHost` property in `etc/org.apache.karaf.shell.cfg` is set to the default value of `0.0.0.0`, you can run the `stop` script without any arguments. However, if you have configured a different hostname, you must run **`stop -h`** ***`hostname`***.

Chapter 5. Creating a New Fabric

When there are no existing fabric's to join, or you want to start a new fabric, you can create a new one from a standalone container.

Overview

A fabric consists of two different kinds of container:

Fabric Server
: A Fabric Server has a special status in the fabric, because it is responsible for maintaining a replica of the fabric registry. In each Fabric Server, a registry service is installed. The registry service (based on Apache ZooKeeper) maintains a replica of the registry database and provides a ZooKeeper server, which ordinary agents can connect to in order to retrieve registry data.

Fabric Container
: A Fabric Container is aware of the locations of all of the Fabric Servers, and it can retrieve registry data from any Fabric Server in the Fabric Ensemble. A *Fabric Agent* is installed in each Fabric Container. The Fabric Agent actively monitors the fabric registry, and whenever a relevant modification is made to the registry, it immediately updates its container to keep the container consistent with the registry settings.

Fabric Ensemble

A *Fabric Ensemble* is a collection of Fabric Servers and Fabric Containers that collectively maintain the state of the fabric registry. The Fabric Ensemble implements a replicated database and uses a quorum-based voting system[1] to ensure that data in the fabric registry remains consistent across all of the fabric's containers. To guard against network splits in a quorum-based system, it is a requirement that *the number of Fabric Servers in a Fabric Ensemble is always an odd number*.

The number of Fabric Servers in a fabric is typically 1, 3, or 5. A fabric with just one Fabric Server is suitable for experimentation only. A live production system should have at least 3 or 5 Fabric Servers, installed on separate hosts, to provide fault tolerance.

Procedure

To create a new fabric from a standalone container:

[1] http://en.wikipedia.org/wiki/Quorum_(distributed_computing)

1. Connect to the standalone container's command console.

2. Any existing users in the *InstallDir*/etc/users.properties file are automatically used to initialize the fabric's user data, when you create the fabric. This provides a convenient way to initialize the fabric's user data.

   If you have not already done so, it is recommended that you populate the users.properties file, by adding one or more lines of the following form:

   ```
   Username=Password[,RoleA][,RoleB]...
   ```

   At least one of the users must have the admin role, to enable administration of the fabric. For example:

   ```
   admin=secretpassword,admin
   ```

3. Assuming that some users are already defined in the users.properties file, you can create a new fabric by entering the following command:

   ```
   FuseESB:karaf@root> fabric:create --zookeeper-password admin
   ```

   The current container (named root by default) becomes a Fabric Server with a registry service installed. Initially, this is the only container in the fabric. The Zookeeper password is used to protect sensitive data in the Fabric registry service (all of the nodes under /fabric).

   Tip

   If you want to import a predefined set of profiles, use the **-p *import-dir*** option to specify the set of profiles to import.

   For more details on **fabric:create** see fabric:create in *Console Reference*.

4. *(Alternative)* If no users are predefined in the users.properties file, an alternative approach is to define a new user at the same time the fabric is created, by supplying the --new-user and --new-user-password options, as follows:

   ```
   FuseESB:karaf@root> fabric:create --new-user jdoe --new-
   user-password secretpassword --zookeeper-password admin
   ```

The new user, `jdoe`, is automatically assigned the `admin` role, which gives the user full administration privileges.

Fabric creation process

Several things happen when a fabric is created from a standalone container:

1. The container installs the requisite OSGi bundles to become a Fabric Server.
2. The Fabric Server starts a registry service, which listens on IP port 2181 (which makes fabric configuration data available to all of the containers in the fabric).
3. The Fabric Server installs a new JAAS realm (based on the ZooKeeper login module), which overrides the default JAAS realm and stores its user data in the ZooKeeper registry.
4. The new Fabric Ensemble consists of a *single* Fabric Server (the current container).
5. A default set of profiles is imported from *`InstallDir`*`/fabric/import` (can optionally be overridden).
6. After the standalone container is converted into a Fabric Server, the previously installed OSGi bundles and Karaf features are completely cleared away and replaced by the default Fabric Server configuration. For example, some of the shell command sets that were available in the standalone container are no longer available in the Fabric Server.

Chapter 6. Joining a Fabric

Overview

Any standalone container can be joined to an existing fabric using the **fabric:join** console command. You need to supply the URL of one of the Fuse Servers in the fabric and the standalone container is then added to the fabric. The container can join the fabric as either a managed container or a non-managed container:

- A *managed container* is a full member of the fabric and is managed by a Fabric Agent. The agent configures the container based on information provided by the fabric's ensemble. The ensemble knows which profiles are associated with the container and the agent determines what to install based on the contents of the profiles.
- A *non-managed container* is not managed by a Fabric Agent. Its configuration remains intact after it joins the fabric and is controlled as if the container were a standalone container. Joining the fabric in this manner registers the container with the fabric's ensemble and allows clients to locate the services running in the container using the fabric's discovery mechanism.

Joining a fabric as a managed container

The default behavior of the **fabric:join** command is to wipe out the container's configuration and replace it with the `fabric` profile. If you want to preserve the previous configuration of the container, however, you must ensure that the fabric has an appropriately configured *profile*, which you can deploy into the container after it joins the fabric.

The **fabric:join** command's `-p` option enables you to specify a profile to install into the container once the agent is installed.

For details of how to create and edit a profile, see "Create Fabric Profiles" in *Getting Started*, fabric:profile-create in *Console Reference*, and fabric:profile-edit in *Console Reference*.

Joining a fabric as an non-managed container

When a container joins a fabric as a non-managed container, its deployment mechanisms continue to function like a standalone container (based on `osgi:install`, `features:install`, and hot deployment), because a Fabric Agent does *not* take control of its configuration. The agent only registers the container with the fabric's ensemble and keeps the registry entries for it up to date. This enables the newly joined container to discover services running

in the container (through Fabric's discovery mechanisms) and to administer these services.

Joining a fabric as an non-managed container is a convenient approach to take when you want to use your local container as a console to administer a fabric. For example, this is an approach that is typically taken with the Fuse Management Console (FMC).

How to join a fabric

To join a container to a fabric, perform the following steps:

1. Get the registry service URL for one of the Fabric Servers in the existing fabric. The registry service URL has the following format:

   ```
   Hostname[:IPPort]
   ```

   Normally, it is sufficient to specify just the hostname, `Hostname`, because the registry service uses the fixed port number, 2182, by default. In exceptional cases, you can discover the registry service port by following the instructions in "How to discover the URL of a Fabric Server" on page 41.

2. Get the ZooKeeper password for the fabric. An administrator can access the fabric's ZooKeeper password at any time, by entering the following console command (while logged into one of the Fabric Containers):

   ```
   karaf@root> fabric:ensemble-password
   ```

3. Connect to the standalone container's command console.

4. Join a container in one of the following ways:

   - *Join as a managed container, with a default profile*—uses the `fabric` profile.

     ```
     karaf@root> fabric:join --zookeeper-password ZooPass URL
      ContainerName
     ```

   - *Join as a managed container, specifying a custom profile*—uses a custom profile.

     ```
     karaf@root> fabric:join --zookeeper-password ZooPass -p
      Profile URL ContainerName
     ```

- *Join as a non-managed container*—preserves the existing container configuration.

```
karaf@root> fabric:join -n --zookeeper-password ZooPass
 URL ContainerName
```

Where you can specify the following values:

`ZooPass`

The existing fabric's ZooKeeper password.

`URL`

The URL for one of the fabric's registry services (usually just the hostname where a Fabric Server is running).

`ContainerName`

The new name of the container when it registers itself with the fabric.

Warning

If the container being added to the fabric has the same name as a container already registered with the fabric, both containers will be reset and will always share the same configuration.

`Profile`

The name of the custom profile to install into the container after it joins the fabric (managed container only).

5. If you joined the container as a *managed container*, you can subsequently deploy a different profile into the container using the `fabric:container-change-profile` console command (see ????).

How to discover the URL of a Fabric Server

If you suspect that a Fabric Server is not using the default IP port, 2181, for its registry service, you can discover the port as follows:

1. Connect to the command console of one of the containers in the fabric.
2. Enter the following sequence of console commands:

```
FuseMQ:karaf@root> config:edit org.fusesource.fabric.zookeeper
FuseMQ:karaf@root> config:proplist
   service.pid = org.fusesource.fabric.zookeeper
   zookeeper.url = myhostA:2181,myhostB:2181,my
hostC:2181,myhostC:2182,myhostC:2183
   fabric.zookeeper.pid = org.fusesource.fabric.zookeeper
FuseMQ:karaf@root> config:cancel
```

The `zookeeper.url` property holds a comma-separated list of Fabric Server URLs. You can use any one of these URLs to join the fabric.

Chapter 7. Using Remote Connections to Manage a Container

It does not always make sense to use a local console to manage a container. Fuse ESB Enterprise has a number of ways of remotely managing a container. You can use a remote container's command console or start a remote client.

Configuring a Container for Remote Access

Overview

When you start the Fuse ESB Enterprise runtime in default mode or in server mode on page 33, it enables a remote console that can be accessed over SSH from any other Fuse ESB Enterprise console. The remote console provides all of the functionality of the local console and allows a remote user complete control over the container and the services running inside of it.

Note

When run in client mode on page 33 the Fuse ESB Enterprise runtime disables the remote console.

Configuring a standalone container for remote access

The SSH hostname and port number are configured in the *InstallDir*/etc/org.apache.karaf.shell.cfg configuration file. Example 7.1 on page 44 shows a sample configuration that changes the port used to 8102.

Example 7.1. Changing the Port for Remote Access

```
sshPort=8102
sshHost=0.0.0.0
```

Configuring a fabric container for remote access

The SSH hostname and port number are set at the time a container is created. It is *not* possible to change the SSH address of a fuse container after the container has been created.

Connecting and Disconnecting Remotely

Connecting to a Standalone Container from a Remote Container

Overview

Any container's command console can be used to access a remote container. Using SSH, the local container's console connects to the remote container and functions as a command console for the remote container.

Using the ssh:ssh console command

You connect to a remote container's console using the **ssh:ssh** console command.

Example 7.2. ssh:ssh Command Syntax

`ssh:ssh` {-l *`username`*} {-P *`password`*} {-p *`port`*} { *`hostname`* }

`-l` *`username`*

The username used to connect to the remote container. Use valid JAAS login credentials that have `admin` privileges (see "Configuring JAAS Security" on page 85).

`-P` *`password`*

The password used to connect to the remote container.

`-p` *`port`*

The SSH port used to access the desired container's remote console.

By default this value is `8101`. See "Configuring a standalone container for remote access" on page 44 for details on changing the port number.

`hostname`

The hostname of the machine that the remote container is running on. See "Configuring a standalone container for remote access" on page 44 for details on changing the hostname.

Warning

We recommend that you customize the username and password in the `etc/users.properties` file. See "Configuring JAAS Security" on page 85for details.

Example 7.3. Connecting to a Remote Console

```
karaf@root>ssh:ssh -l smx -P smx -p 8108 hostname
```

To confirm that you have connected to the correct container, type **`shell:info`** at the prompt. Information about the currently connected instance is returned, as shown in Example 7.4 on page 47.

Example 7.4. Output of the shell:info Command

```
Karaf
  Karaf version              2.2.5.fuse-beta-7-052
  Karaf home                 /Volumes/ESB/fuse-esb-7.1.0.fuse-047
  Karaf base                 /Volumes/ESB/fuse-esb-7.1.0.fuse-047
  OSGi Framework           org.apache.felix.framework - 4.0.3.fuse-
beta-7-052

JVM
  Java Virtual Machine       Java HotSpot(TM) 64-Bit Server VM
version 20.6-b01-415
  Version                    1.6.0_31
  Vendor                     Apple Inc.
  Uptime                     6 minutes
  Total compile time         24.048 seconds
Threads
  Live threads               62
  Daemon threads             43
  Peak                       287
  Total started              313
Memory
  Current heap size          78,981 kbytes
  Maximum heap size          466,048 kbytes
  Committed heap size        241,920 kbytes
  Pending objects            0
  Garbage collector          Name = 'PS Scavenge', Collections =
11, Time = 0.271 seconds
  Garbage collector          Name = 'PS MarkSweep', Collections =
1, Time = 0.117 seconds
Classes
  Current classes loaded     5,720
  Total classes loaded       5,720
  Total classes unloaded     0
Operating system
  Name                       Mac OS X version 10.7.3
  Architecture               x86_64
  Processors                 2
```

Disconnecting from a remote console

To disconnect from a remote console, enter `logout` or press **Ctrl**+**D** at the prompt.

You will be disconnected from the remote container and the console will once again manage the local container.

Connecting to a Fabric Container From another Fabric Container

Overview

When containers are deployed into a fabric, they are all connected to each other. You can easily connect to any container's command console from any of its peers. When connecting using fabric, you do not need to know any of the location details for the container you want to connect to. The fabric's runtime registry stores all of the location details needed to establish the remote connection.

Using the fabric:container-connect command

In the context of a fabric, you should connect to a remote runtime's console using the **fabric:container-connect** command.

Example 7.5. fabric:container-connect Command Syntax

```
fabric:container-connect {-u username} {-p password}
{containerName}
```

`-u username`

The username used to connect to the remote console. The default value is `admin`.

`-p password`

The password used to connect to the remote console. The default value is `admin`.

containerName

The name of the container.

Warning

We recommend that you change the default administrator username and password. See "Configuring JAAS Security" on page 85 for details.

Example 7.6. Connecting to a Remote Container

```
karaf@root>fabric:container-connect -u admin -p admin containerName
```

To confirm that you have connected to the correct container, type **`shell:info`** at the prompt. Information about the currently connected instance is returned, as shown in Example 7.7 on page 50.

Example 7.7. Output of the shell:info Command

```
Karaf
  Karaf version               2.3.0.fuse-71-044
  Karaf home                  /Volumes/SAMSUNG/Programs/ESB/fuse-
esb-7.1.0.fuse-044
  Karaf base                  /Volumes/SAMSUNG/Programs/ESB/fuse-
esb-7.1.0.fuse-044/instances/child1
  OSGi Framework            org.apache.felix.framework - 4.0.3.fuse-
71-044

JVM
  Java Virtual Machine        Java HotSpot(TM) 64-Bit Server VM
version 20.8-b03-424
  Version                     1.6.0_33
  Vendor                      Apple Inc.
  Uptime                      7 minutes
  Total compile time          5.336 seconds
Threads
  Live threads                42
  Daemon threads              31
  Peak                        96
  Total started               123
Memory
  Current heap size           32,832 kbytes
  Maximum heap size           466,048 kbytes
  Committed heap size         104,960 kbytes
  Pending objects             0
  Garbage collector           Name = 'PS Scavenge', Collections =
7, Time = 0.063 seconds
  Garbage collector           Name = 'PS MarkSweep', Collections =
1, Time = 0.060 seconds
Classes
  Current classes loaded      4,019
  Total classes loaded        4,019
  Total classes unloaded      0
Operating system
  Name                        Mac OS X version 10.7.4
  Architecture                x86_64
  Processors                  2
```

Disconnecting from a remote console

To disconnect from a remote console, enter `logout` or press **Ctrl**+**D** at the prompt.

You will be disconnected from the remote container and the console will once again manage the local container.

Connecting to a Container Using the Client Command-Line Utility

Using the remote client

The remote client allows you to securely connect to a remote Fuse ESB Enterprise container without having to launch a full Fuse ESB Enterprise container locally.

For example, to quickly connect to a Fuse ESB Enterprise instance running in server mode on the same machine, open a command prompt and run the **client[.bat]** script (which is located in the *InstallDir*/bin directory), as follows:

```
client
```

More usually, you would provide a hostname, port, username, and password to connect to a remote instance. If you were using the client within a larger script, for example in a test suite, you could append console commands as follows:

```
client -a 8101 -h hostname -u username -p password shell:info
```

Alternatively, if you omit the -p option, you will be prompted to enter a password.

For a standalone container, use any valid JAAS user credentials that have admin privileges.

For a container in a fabric, the default username and password is admin and admin.

To display the available options for the client, type:

```
client --help
```

Example 7.8. Karaf Client Help

```
Apache Felix Karaf client
  -a [port]     specify the port to connect to
  -h [host]     specify the host to connect to
  -u [user]     specify the user name
  -p [password] specify the password
  --help        shows this help message
  -v            raise verbosity
  -r [attempts] retry connection establishment (up to attempts times)

  -d [delay]    intra-retry delay (defaults to 2 seconds)
  [commands]    commands to run
```

```
If no commands are specified, the client will be put in an interactive
 mode
```

Disconnecting from a remote client console

If you used the remote client to open a remote console, as opposed to using it to pass a command, you will need to disconnect from it. To disconnect from the remote client's console, enter **logout** or press **Ctrl**+**D** at the prompt.

The client will disconnect and exit.

Connecting to a Container Using the SSH Command-Line Utility

Overview

You can also use the `ssh` command-line utility (a standard utility on UNIX-like operating systems) to log in to the Fuse ESB Enterprise container, where the authentication mechanism is based on public key encryption (the public key must first be installed in the container). For example, given that the container is configured to listen on IP port 8101, you could log in as follows:

```
ssh -p 8101 jdoe@localhost
```

Important

Key-based login is currently supported only on standalone containers, not on Fabric containers.

Prerequisites

To use key-based SSH login, the following prerequisites must be satisfied:

- The container must be standalone (Fabric is not supported) with the `PublickeyLoginModule` installed.
- You must have created an SSH key pair (see "Creating a new SSH key pair" on page 54).
- You must install the public key from the SSH key pair into the container (see "Installing the SSH public key in the container" on page 54).

Default key location

The `ssh` command automatically looks for the private key in the default key location. It is recommended that you install your key in the default location, because it saves you the trouble of specifying the location explicitly.

On a *NIX operating system, the default locations for an RSA key pair are:

```
~/.ssh/id_rsa
~/.ssh/id_rsa.pub
```

On a Windows operating system, the default locations for an RSA key pair are:

```
C:\Documents and Settings\Username\.ssh\id_rsa
C:\Documents and Settings\Username\.ssh\id_rsa.pub
```

Note

Fuse ESB Enterprise supports only RSA keys. DSA keys do *not* work.

Creating a new SSH key pair

Generate an RSA key pair using the `ssh-keygen` utility. Open a new command prompt and enter the following command:

```
ssh-keygen -t rsa -b 2048
```

The preceding command generates an RSA key with a key length of 2048 bits. You will then be prompted to specify the file name for the key pair:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/User
name/.ssh/id_rsa):
```

Type return to save the key pair in the default location. You will then be prompted for a pass phrase:

```
Enter passphrase (empty for no passphrase):
```

You can optionally enter a pass phrase here or type return twice to select no pass phrase.

Note

If you want to use the same key pair for running Fabric console commands, it is recommended that you select *no pass phrase*, because Fabric does not support using encrypted private keys.

Installing the SSH public key in the container

To use the SSH key pair for logging into the Fuse ESB Enterprise container, you must install the SSH public key in the container by creating a new user entry in the *InstallDir*`/etc/keys.properties` file. Each user entry in this file appears on a single line, in the following format:

```
Username=PublicKey,Role1,Role2,...
```

For example, given that your public key file, `~/.ssh/id_rsa.pub`, has the following contents:

```
ssh-rsa AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAM
MeP4C2USZpRV1AIlH7WT2NWPq/xfW6MPbLm1Vs14E7
gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnfqim
```

```
FQ8E+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAAAAFQCX
YFCPFSMLzLKSuYKi64QL8Fgc9QAAAnEA9+Gghd
abPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBx
CBgLRJFnEj6Ewo
FhO3zwkyjMim4TwWeotifI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zK
TxvqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoAAACB
AKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXf
skl8wO63OzP/qLmcJM0+JbcRU/53Jj7uyk31drV2qxhIOsLDC9dGCWj4
7Y7TyhPdXh/0dthTRBy6bqGtRPxGa7gJov1xm/UuYYXPI
UR/3x9MAZvZ5xvE0kYXO+rx jdoe@doemachine.local
```

You can create the `jdoe` user with the `admin` role by adding the following entry to the *`InstallDir`*`/etc/keys.properties` file (on a single line):

```
jdoe=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAM
MeP4C2USZpRV1AIlH7WT2NWPq/xfW6MPbLm1Vs14E7
gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnfqim
FQ8E+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAAAAFQCX
YFCPFSMLzLKSuYKi64QL8Fgc9QAAAnEA9+Gghd
abPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBx
CBgLRJFnEj6Ewo
FhO3zwkyjMim4TwWeotifI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zK
TxvqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoAAACB
AKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXf
skl8wO63OzP/qLmcJM0+JbcRU/53Jj7uyk31drV2qxhIOsLDC9dGCWj4
7Y7TyhPdXh/0dthTRBy6bqGtRPxGa7gJov1xm/UuYYXPI
UR/3x9MAZvZ5xvE0kYXO+rx,admin
```

Important

Do not insert the entire contents of the `id_rsa.pub` file here. Insert just the block of symbols which represents the public key itself.

Checking that public key authentication is supported

After starting the container, you can check whether public key authentication is supported by running the `jaas:realms` console command, as follows:

```
Index Realm                 Module Class

    1 karaf                org.apache.karaf.jaas.modules.prop
erties.PropertiesLoginModule
    2 karaf                 org.apache.karaf.jaas.modules.pub
lickey.PublickeyLoginModule
```

You should see that the `PublickeyLoginModule` is installed. With this configuration you can log in to the container using either username/password credentials or public key credentials.

Logging in using key-based SSH

You are now ready to login to the container using the key-based SSH utility. For example:

```
$ ssh -p 8101 jdoe@localhost
_______                       _____  _____ _______
|  ___|                     |  ___|/  ___|| ___ \
| |_  _   _  ___   ___  | |__  \ `--. | |_/ /
|  _|| | | |/ __| / _ \ |  __|  `--. \|  ___ \
| |  | |_| |\__ \|  __/ | |___ /\__/ /| |_/ /
\_|   \__,_||___/ \___| \____/ \____/ \____/

  Fuse ESB (7.1.0.fuse-036)
  http://fusesource.com/products/fuse-esb-enterprise/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown Fuse ESB.

FuseESB:fbolton@root>
```

Note

If you are using an encrypted private key, the `ssh` utility will prompt you to enter the pass phrase.

Stopping a Remote Container

Using the stop script

You can stop a remote container without starting up Fuse ESB Enterprise on your local host by running the **stop(.bat)** script from the local *InstallDir*/bin directory.

Example 7.9. stop Script Syntax

stop [-a *port*] {-h *hostname*} {-u *username*} {-p *password*}

-a *port*

The SSH port of the remote instance. Defaults to 8101.

-h *hostname*

The hostname of the machine that the remote instance is running on.

-u *username*

The username used to connect to the remote instance. Use valid JAAS login credentials that have admin privileges

-p *password*

The password used to connect to the remote instance.

Using the fabric:container-stop console command

If your containers are deployed in a fabric, you can stop any container in the fabric using the **fabric:container-stop** command. For example, to shut down the container called child1, you would enter the following console command:

```
FuseESB@root> fabric:container-stop child1
```

Using the remote console

If you have connected to a remote console using the **ssh:ssh** command, the **fabric:container-connect** command, or the remote client, you can stop the remote instance using the **osgi:shutdown** command.

Note

Pressing **Ctrl**+**D** in a remote console simply closes the remote connection and returns you to the local shell.

Chapter 8. Managing Child Containers

A child container is a container that shares a common Fuse ESB Enterprise runtime with a parent container, but has its own configuration files, runtime information, logs and temporary files. The child container functions as an independent container into which you can deploy bundles.

Standalone Child Containers

Using the admin console commands

The **admin** console commands allow you to create and manage instances of the Fuse ESB Enterprise runtime on the same machine. Each new runtime is a child instance of the runtime that created it. You can easily manage the children using names instead of network addresses. For details on the **admin** commands, see Admin Console Commands in *Console Reference*.

Creating child containers

You create a new runtime container by typing **`admin:create`** in the Fuse ESB Enterprise console.

As shown in Example 8.1 on page 60, **admin:create** causes the container to create a new child container in the active container's `instances/`*`containerName`* directory. The new container is a direct copy of its parent. The only difference between parent and child is the port number they listen on. The child container is assigned an SSH port number based on an incremental count starting at 8101.

Example 8.1. Creating a Runtime Instance

```
FuseESB@root>admin:create finn
Creating new instance on port 8106 at: /home/fuse/esb4/instances/finn
Creating dir:  /home/fuse/esb4/instances/finn/bin
Creating dir:  /home/fuse/esb4/instances/finn/etc
Creating dir:  /home/fuse/esb4/instances/finn/system
Creating dir:  /home/fuse/esb4/instances/finn/deploy
Creating dir:  /home/fuse/esb4/instances/finn/data
Creating file: /home/fuse/esb4/instances/finn/etc/config.properties
Creating file: /home/fuse/esb4/instances/finn/etc/java.util.log
ging.properties
Creating file: /home/fuse/esb4/instances/finn/etc/org.apache.fe
lix.fileinstall-deploy.cfg
Creating file: /home/fuse/esb4/in
stances/finn/etc/org.apache.karaf.log.cfg
Creating file: /home/fuse/esb4/instances/finn/etc/org.apache.karaf.fea
tures.cfg
Creating file: /home/fuse/esb4/instances/finn/etc/org.apache.karaf.man
agement.cfg
Creating file: /home/fuse/esb4/instances/finn/etc/org.ops4j.pax.log
ging.cfg
Creating file: /home/fuse/esb4/in
stances/finn/etc/org.ops4j.pax.url.mvn.cfg
Creating file: /home/fuse/esb4/instances/finn/etc/startup.properties
Creating file: /home/fuse/esb4/instances/finn/etc/system.properties
Creating file: /home/fuse/esb4/in
```

```
stances/finn/etc/org.apache.karaf.shell.cfg
Creating file: /home/fuse/esb4/instances/finn/bin/karaf
Creating file: /home/fuse/esb4/instances/finn/bin/start
Creating file: /home/fuse/esb4/instances/finn/bin/stop
```

Changing a child's SSH port

You can change the SSH port number assigned to a child container using the **admin:change-port** command. The syntax for the command is:

`admin:change-port` {*containerName*} {*portNumber*}

Important

You can only use the **admin:change-port** command on stopped containers.

Starting child containers

New containers are created in the stopped state. To start a child container and make it ready to host applications, use the **admin:start** command. This command takes a single argument, *containerName*, that identifies the child you want started.

Listing all child containers

To see a list of all Fuse ESB Enterprise containers running under a particular installation, use the **admin:list** command:

Example 8.2. Listing Instances

```
FuseESB@root>admin:list
  Port   State       Pid  Name
[ 8107] [Started ] [10628] harry
[ 8101] [Started ] [20076] root
[ 8106] [Started ] [15924] dick
[ 8105] [Started ] [18224] tom
```

Connecting to a child container

You can connect to a started child container's remote console using the **admin:connect** command.As shown in Example 8.3 on page 61, this command takes three arguments:

Example 8.3. Admin connect Command

`admin:connect` { *containerName* } {-u *username*} {-p *password*}

containerName

The name of the child to which you want to connect.

-u *username*

The username used to connect to the child's remote console. Use valid JAAS user credentials that have admin privileges (see "Configuring JAAS Security" on page 85).

-p *password*

This argument specifies the password used to connect to the child's remote console.

Once you are connected to the child container., the prompt changes to display the name of the current instance, as shown:

```
FuseESB@harry>
```

Stopping a child container

To stop a child container, from within the container itself, type **osgi:shutdown** or simply **shutdown**.

To stop a child container remotely—in other words, from a parent or sibling instance—type **admin:stop *containerName***.

Destroying a child container

You can permanently delete a stopped child container using the **admin:destroy** *containerName* command.

Important

You can only remove stopped children.

Using the admin script

You can also use manage an Fuse ESB Enterprise container running in server mode without starting a new instance of the runtime. The **admin** script in the *InstallDir*/bin directory provides the all of the **admin** console commands except for **admin:connect**.

Example 8.4. The admin Script

```
admin.bat: Ignoring predefined value for KARAF_HOME
Available commands:
  change-port - Changes the port of an existing container instance.
```

```
  create - Creates a new container instance.
  destroy - Destroys an existing container instance.
  list - List all existing container instances.
  start - Starts an existing container instance.
  stop - Stops an existing container instance.
Type 'command --help' for more help on the specified command.
```

For example, to list all of the Fuse ESB Enterprise containers on your host machine, type:

Windows

```
admin.bat list
```

*NIX

```
./admin list
```

Fabric Child Containers

Creating child containers

You create a new child container using the `fabric:container-create-child` console command, which has the following syntax:

```
karaf@root> fabric:container-create-child parent child [number]
```

Where *`parent`* is the name of an existing container in the fabric and *`child`* is the name of the new child container. If you create multiple child containers (by specifying the optional *`number`* argument), the new child instances are named *`child`*1, *`child`*2, and so on.

For example, assuming the container, root, already belongs to your fabric, you can create two new child containers as follows:

```
karaf@root> fabric:container-create-child root child 2
The following containers have been created successfully:
 child1
 child2
```

Listing all container instances

To list all of the containers in the current fabric (including child instances), use the `fabric:container-list` console command. For example:

```
FuseESB@root> fabric:container-list
[id]                           [version] [alive] [profiles]                    [provision
 status]
root                           1.0       true    fabric, fabric-ensemble-0000-1
  child1                       1.0       true    default                       success
  child2                       1.0       true    default                       success
```

Assigning a profile to a child container

By default, a child is assigned the `default` profile when it is created. To assign a new profile (or profiles) to a child container after it has been created, use the `fabric:container-change-profile` console command.

Tip

You can assign a profile other than `default` to a newly created container by using the **fabric:container-create-child** command's `--profile` argument.

For example, to assign the `example-camel` profile to the `child1` container, enter the following console command:

```
FuseESBf@root> fabric:container-change-profile child1 example-
camel
```

The command removes the profiles currently assigned to `child1` and replaces them with the specified list of profiles (where in this case, there is just one profile in the list, `example-camel`).

Connecting to a child container

To connect to a child container, use the `fabric:container-connect` console command. For example, to connect to `child1`, enter the following console command:

```
FuseESB@root>fabric:container-connect -u admin -p admin child1
```

You should see output like the following in your console window:

```
Connecting to host YourHost on port 8102
Connected
_______                      ______  ______ _______
|  ___|                    |  ___|/  ___|| ___ \
| |_  _   _  ___   ___    | |__   \ `--. | |_/ /
|  _|| | | |/ __| / _ \   |  __|   `--. \|  __ \
| |  | |_| |\__ \|  __/ | |___ /\__/ /| |_/ /
\_|   \__,_||___/ \___| \____/ \____/ \____/

  Fuse ESB (7.0.0.fuse-beta-039)
  http://fusesource.org/esb/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown Fuse ESB.

FuseESB:admin@child1>
```

To terminate the session, enter **`Ctrl-D`**.

Starting a child container

To start a child container that was previously stopped, use the **fabric:container-start** command, providing the child container's name as the command argument—for example, to restart `child1`:

```
FuseESB@root>fabric:container-start child1
```

This command starts up the child in a separate JVM.

Stopping a child container

To stop a child instance, use the **fabric:container-stop** command, providing the child container's name as the command argument—for example, to stop `child1`:

```
FuseESB@root>fabric:container-stop child1
```

This command kills the JVM process that hosts the `child1` container.

Destroying a child container

To completely destroy a child container use the **fabric:container-delete** command. For example, to destroy the `child1` container instance, enter the following console command:

```
FuseESB@root> fabric:container-delete child1
```

Destroying a child container does the following:

- stops the child's JVM process
- physically removes all files related to the child container

Chapter 9. Configuring Fuse ESB Enterprise

Fuse ESB Enterprise uses the OSGi Configuration Admin service to manage the configuration of OSGi services. How you feed information to the configuration service depends on how the container is deployed.

Introducing Fuse ESB Enterprise Configuration

OSGi configuration

The OSGi Configuration Admin service specifies the configuration information for deployed services and ensures that the services receive that data when they are active.

A configuration is a list of name-value pairs read from a `.cfg` file in the *`InstallDir`*`/etc` directory. The file is interpreted using the Java properties file format. The filename is mapped to the persistent identifier (PID) of the service that is to be configured. In OSGi, a PID is used to identify a service across restarts of the container.

Configuration files

You can configure the Fuse ESB Enterprise runtime using the following files:

Table 9.1. Fuse ESB Enterprise Configuration Files

Filename	Description
`activemq.xml`	Configures the defaultApache ActiveMQ broker in a Fabric (used in combination with the `org.fusesource.mq.fabric.server-default.cfg` file).
`config.properties`	The main configuration file for the container See "Setting OSGi Framework and Initial Container Properties" on page 72 for details.
`keys.properties`	Lists the users who can access the Fuse ESB Enterprise runtime using the SSH key-based protocol. The file's contents take the format *`username=publicKey,role`*
`org.apache.aries.transaction.cfg`	Configures the transaction feature
`org.apache.felix.fileinstall-deploy.cfg`	Configures a watched directory and polling interval for hot deployment.
`org.apache.karaf.features.cfg`	Configures a list of feature repositories to be registered and a list of features to be installed when Fuse ESB Enterprise starts up for the first time.
`org.apache.karaf.features.obr.cfg`	Configures the default values for the features OSGi Bundle Resolver (OBR).
`org.apache.karaf.jaas.cfg`	Configures options for the Karaf JAAS login module. Mainly used for configuring encrypted passwords (disabled by default).

Filename	Description
`org.apache.karaf.log.cfg`	Configures the output of the **log** console commands. See "Logging Configuration" on page 94.
`org.apache.karaf.management.cfg`	Configures the JMX system. See "Configuring JMX" on page 83 for details.
`org.apache.karaf.shell.cfg`	Configures the properties of remote consoles. For more information see "Configuring a Container for Remote Access" on page 44.
`org.apache.servicemix.jbi.cfg`	Configures the shutdown timeout for the JBI container.
`org.apache.servicemix.nmr.cfg`	Configures the default thread pool settings for JBI. See Configuring JBI Component Thread Pools on page 111.
`org.apache.servicemix.components.`*`Name`*`.cfg`	Configures the thread pool settings specifically for the *`Name`* JBI component. See Configuring JBI Component Thread Pools on page 111.
`org.fusesource.bai.agent.cfg`	Configures the Fuse BAI (Business Activity Insight) feature, if it is installed.
`org.fusesource.fabric.fab.osgi.url.cfg`	Configures the Maven repositories used by the Fuse Application Bundle (FAB) runtime when downloading artifacts. If the properties in this file are not set, FAB defaults to the values in `org.ops4j.pax.url.mvn.cfg`.
`org.fusesource.fabric.maven.cfg`	Configures the Maven repositories used by the Fabric Maven Proxy when downloading artifacts, (The Fabric Maven Proxy is used for provisioning new containers on a remote host.)
`org.fusesource.mq.fabric.server-default.cfg`	Configures the defaultApache ActiveMQ broker in a Fabric (used in combination with the `activemq.xml` file).
`org.jclouds.shell.cfg`	Configures options for formatting the output of `jclouds:*` console commands.
`org.ops4j.pax.logging.cfg`	Configures the logging system. For more, see "Logging Configuration" on page 94.
`org.ops4j.pax.url.mvn.cfg`	Configures additional URL resolvers.
`org.ops4j.pax.web.cfg`	Configures the default Jetty container (Web server). See Securing the Web Console in *Security Guide*.

Filename	Description
`startup.properties`	Specifies which bundles are started in the container and their start-levels. Entries take the format *`bundle=start-level`*.
`system.properties`	Specifies Java system properties. Any properties set in this file are available at runtime using `System.getProperties()`. See Setting System and Config Properties on page 72 for more.
`users.properties`	Lists the users who can access the Fuse ESB Enterprise runtime either remotely or via the web console. The file's contents take the format *`username=password,role`*

Configuration file naming convention

The file naming convention for configuration files depends on whether the configuration is intended for an OSGi Managed Service or for an OSGi Managed Service factory.

The configuration file for an OSGi Managed Service obeys the following naming convention:

```
<PID>.cfg
```

Where `<PID>` is the *persistent ID* of the OSGi Managed Service (as defined in the OSGi Configuration Admin specification). A persistent ID is normally dot-delimited—for example, `org.ops4j.pax.web`.

The configuration file for an OSGi Managed Service Factory obeys the following naming convention:

```
<PID>-<InstanceID>.cfg
```

Where `<PID>` is the *persistent ID* of the OSGi Managed Service Factory. In the case of a managed service factory's `<PID>`, you can append a hyphen followed by an arbitrary instance ID, `<InstanceID>`. The managed service factory then creates a unique service instance for each `<InstanceID>` that it finds.

JBI component configuration

In addition to the container's configuration files, the *`InstallDir`*`/etc` folder may contain a number of configuration files for the JBI components that ship with Fuse ESB Enterprise.

The component configuration files are named using the scheme `org.apache.servicemix.components.ComponentName.cfg`. For example, you would configure the JMS component using a file called `org.apache.servicemix.components.jms.cfg`.

The contents of a component's configuration file is largely component specific. However, each component configuration file contains properties for configuring the thread pool used by the component to process message exchanges. See "Configuring JBI Component Thread Pools" on page 111 for details.

Setting OSGi Framework and Initial Container Properties

Overview

There are a number of configuration properties that are set when a container is bootstrapped. These properties include the container's name, the default features repository used by the container, the OSGi framework provider, and other settings. These properties are specified in two property files in the `etc` folder:

- `config.properties`—specifies the bootstrap properties for the OSGi framework
- `system.properties`—specifies properties to configure container functions

OSGi framework properties

The `etc/config.properties` file contains the properties used to specify which OSGi framework implementation to load and properties for configuring the framework's behaviors. Table 9.2 on page 72 describes the key properties to set.

Table 9.2. Properties for the OSGi Framework

Property	Description
`karaf.framework`	Specifies the OSGi framework that Fuse ESB Enterprise uses. The default framework is Apache Felix which is specified using the value `felix`.
`karaf.framework.felix`	Specifies the path to the Apache Felix JAR on the file system.

Important

Fuse ESB Enterprise only supports the Apache Felix OSGi implementation.

Initial container properties

The `etc/system.properties` file contains properties that configure how various aspects of the container behave including:

- the container's name
- the default feature repository used by the container
- the default port used by the OSGi HTTP service
- the initial message broker configuration

Table 9.3 on page 73 describes some of the common properties.

Table 9.3. Container Properties

Property	Description
`karaf.name`	Specifies the name of this container. The default is `root`.
`karaf.default.repository`	Specifies the location of the feature repository the container will use by default. The default setting is the local feature repository installed with Fuse ESB Enterprise.
`org.osgi.service.http.port`	Specifies the default port for the OSGi HTTP Service.

Configuring Standalone Containers Using the Command Console

Overview

The command console's **config** shell provides commands for editing the configuration of a standalone container. The commands allow you to inspect the container's configuration, add new PIDs, and edit the properties of any PID used by the container. These configuration changes are applied directly to the container and will persist across container restarts.

For more details on the **config** commands see "Config Console Commands" in *Console Reference*.

Listing the current configuration

The **config:list** command will show all of the PIDs currently in use by the container. As shown in Example 9.1 on page 74, the output from **config:list** contains all of the PIDs and all of the properties for each of the PIDs.

Example 9.1. Output of the config:list Command

```
...
----------------------------------------------------------------
Pid:            org.ops4j.pax.logging
BundleLocation: mvn:org.ops4j.pax.logging/pax-logging-service/1.4
Properties:
   log4j.appender.out.layout.ConversionPattern = %d{ABSOLUTE} | %-
5.5p | %-16.16
t | %-32.32c{1} | %-32.32C %4L | %m%n
   felix.fileinstall.filename = org.ops4j.pax.logging.cfg
   service.pid = org.ops4j.pax.logging
   log4j.appender.stdout.layout.ConversionPattern = %d{ABSOLUTE} |
%-5.5p | %-16
.16t | %-32.32c{1} | %-32.32C %4L | %m%n
   log4j.appender.out.layout = org.apache.log4j.PatternLayout
   log4j.rootLogger = INFO, out, osgi:VmLogAppender
   log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
   log4j.appender.out.file = C:\apache\apache-servicemix-7.1.0.fuse-
047/data/log/karaf.log
   log4j.appender.stdout = org.apache.log4j.ConsoleAppender
   log4j.appender.out.append = true
   log4j.appender.out = org.apache.log4j.FileAppender
----------------------------------------------------------------
Pid:            org.ops4j.pax.web
BundleLocation: mvn:org.ops4j.pax.web/pax-web-runtime/0.7.1
Properties:
   org.apache.karaf.features.configKey = org.ops4j.pax.web
```

```
   service.pid = org.ops4j.pax.web
   org.osgi.service.http.port = 8181
-------------------------------------------------------------------
...
```

Listing the container's configuration is a good idea before editing a container's configuration. You can use the output to ensure that you know the exact PID to change.

Editing the configuration

Editing a container's configuration involves a number of commands and must be done in the proper sequence. Not following the proper sequence can lead to corrupt configurations or the loss of changes.

To edit a container's configuration:

1. Start an editing session by typing **`config:edit`** ***`PID`***.

   `PID` is the PID for the configuration you are editing. It **must** be entered exactly. If it does not match the desired PID, the container will create a new PID with the specified name.

2. Remind yourself of the available properties in a particular configuration by typing **`config:proplist`**.

3. Use one of the editing commands to change the properties in the configuration.

   The editing commands include:

   - **config:propappend**—appends a new property to the configuration
   - **config:propset**—set the value for a configuration property
   - **config:propdel**—delete a property from the configuration

4. Update the configuration in memory and save it to disk by typing **`config:update`**.

Note

To exit the configuration, without saving your changes, type **`config:cancel`**.

Example 9.2 on page 76 shows a configuration editing session that changes a container's logging behavior.

Example 9.2. Editing a Configuration

```
FuseESB@root> config:edit org.apache.karaf.log
FuseESB@root> config:proplist
   service.pid = org.apache.karaf.log
   size = 500
   felix.fileinstall.filename = org.apache.karaf.log.cfg
   pattern = %d{ABSOLUTE} | %-5.5p | %-16.16t | %-32.32c{1} | %-
32.32C %4L | %m%n
FuseESB@root> config:propset size 300
FuseESB@root> config:update
```

Configuring Fabric Containers

Overview

When a container is part of a fabric, it does not manage its configuration. The container's configuration is managed by the Fabric Agent. The agent runs along with the container and updates the container's configuration based on information from the fabric's registry.

Because the configuration is managed by the Fabric Agent, any changes to the container's configuration needs to be done by updating the fabric's registry. In a fabric, container configuration is determined by one or more profiles that are deployed into the container. To change a container's configuration, you must update the profile(s) deployed into the container using either the console's **fabric:** shell or Fuse Management Console.

Profiles

All configuration in a fabric is stored as profiles in the Fabric Registry. One or more profiles are assigned to containers that are part of the fabric. A profile is a collection of configuration that specifies:

- the Apache Karaf features to be deployed
- OSGi bundles to be deployed
- the feature repositories to be scanned for features
- properties that configure the container's runtime behavior

The configuration profiles are collected into versions. Versions are typically used to make updates to an existing profile without effecting deployed containers. When a container is configured it is assigned a profile version from which it draws the profiles. Therefore, when you create a new version and edit the profiles in the new version, the profiles that are in use are not changed. When you are ready to test the changes, you can roll them out incrementally by moving containers to a new version one at a time.

When a container joins a fabric, a Fabric Agent is deployed with the container and takes control of the container's configuration. The agent will ask the Fabric Registry what version and profile(s) are assigned to the container and configure the container based on the profiles. The agent will download and install of

the specified bundles and features. It will also set all of the specified configuration properties.

Best practices

Editing a profile makes changes to the copy in the Fabric Registry and all of the Fabric Agents are alerted when changes are made. If a running container is using a profile that is changed, its agent will automatically apply the new settings. If the update is benign having the change rolled out to the entire fabric is not an issue. If, on the other hand, the change causes issues, the entire fabric could become unstable.

To avoid having untested changes infecting an entire fabric, you should always make a new version before editing a profile. This isolates the changes in a version that is not running on any containers and provides a quick backup in case the changes are bad.

Once the profile changes have been made, you should test them out by upgrading only a few containers to the new version to see how they behave. As you become confident that the changes are good, you can then upgrade more containers.

Making changes using the command console

The command console's **fabric** shell has commands for managing profiles and versions in a fabric. These commands include:

- **fabric:version-create**—create a new version
- **fabric:profile-create**—create a new profile
- **fabric:profile-edit**—edit the properties in a profile
- **fabric:container-change-profile**—change the profiles assigned to a container

Example 9.3 on page 78 shows a session for updating a profile using the command console.

Example 9.3. Editing Fabric Profile

```
FuseESB:karaf@root> fabric:version-create
Created version: 1.1 as copy of: 1.0
FuseESB:karaf@root> fabric:profile-edit -p org.apache.karaf.log/size=300 NEBroker
```

The change made in Example 9.3 on page 78 is not applied to any running containers because it is made in a new version. In order to apply the change you need to update one or more containers using the **fabric:container-upgrade** command.

See "Fabric Console Commands" in *Console Reference* for more information.

Using Fuse Management Console

Fuse Management Console simplifies the process of configuring containers in a fabric by providing an easy to use Web-based interface and reducing the number of steps required to make the changes. For more information on using Fuse Management Console see the Fuse Management Console Documentation[1].

[1] http://fusesource.com/documentation/fuse-management-console-documentation

Chapter 10. Configuring the Hot Deployment System

Standalone containers scan a directory for OSGi bundles and JBI artifacts to automatically load. You can change the location of this folder and the interval at which the folder is scanned.

Overview

Standalone containers will automatically load and deploy OSGi bundles and JBI artifacts from a pre-configured folder. It scans the folder once a second for new bundles or JBI artifacts. You can change the folder a container scans and the scan interval by editing properties in the `org.apache.felix.fileinstall-deploy` PID.

Important

The hot deployment system is *not* not enabled for fabric containers.

Specifying the hot deployment folder

By default, a container scans the `deploy` folder that is relative to the folder from which you launched the container. You change the folder the container monitors by setting the `felix.fileinstall.dir` property in the `rg.apache.felix.fileinstall-deploy` PID. The value is the absolute path of the folder to monitor. If you set the value to **`/home/joe/deploy`**, the container will monitor a folder in Joe's home directory.

Specifying the scan interval

By default containers scan the hot deployment folder every 1000 milliseconds. To change the interval between scans of the hot deployment folders, you change the `felix.fileinstall.poll` property in the `org.apache.felix.fileinstall-deploy` PID. The value is specified in milliseconds.

Example

Example 10.1 on page 81 shows a configuration editing session that sets `/home/smx/jbideploy` as the hot deployment folder and sets the scan interval to half a second.

Example 10.1. Configuring the Hot Deployment Folders

```
FuseESB@root> config:edit org.apache.felix.fileinstall-deploy
FuseESB@root> config:propset felix.fileinstall.dir /home/smx/jbideploy
```

```
FuseESB@root> config:propset felix.fileinstall.poll 500
FuseESB@root> config:update
```

Chapter 11. Configuring JMX

Fuse ESB Enterprise uses JMX for its underlying management features. You can configure the JMX RMI port, the JMX URL, and the credentials used to access the JMX features.

Overview

Fuse ESB Enterprise uses JMX for reporting runtime metrics and providing some limited management capabilities. You can configure how the JMX management features are accessed by changing the properties in the `org.apache.karaf.management` PID.

Changing the RMI port and JMX URL

Two of the most commonly changed parts of a container's JMX configuration are the RMI port and the JMX URL. You can set these using the properties described in Table 11.1 on page 83.

Table 11.1. JMX Access Properties

Property	Description
`rmiRegistryPort`	Specifies the RMI registry port. The default value is 1099.
`serviceUrl`	Specifies the the URL used to connect to the JMX server. The default URL is `service:jmx:rmi:///jndi/rmi://localhost:1099/karaf-`*`KarafName`*, where *`KarafName`* is the container's name (by default, `root`).

Setting the JMX username and password

In a standalone container, use any valid JAAS user credentials (see "Create a secure JAAS user" on page 24).

In a fabric, the default username is `admin` and the default password is `admin`.

You can change the username and password used to connect to the JMX server by configuring the JAAS security system as described in "Configuring JAAS Security" on page 85.

Troubleshooting on Linux platforms

On Linux platforms, if you have trouble getting a remote JConsole instance to connect to the JMX server, check the following points:

- Check that the hostname resolves to the correct IP address. For example, if the `hostname -i` command returns 127.0.0.1, JConsole will not be able to connect to the JMX server. To fix this, edit the `/etc/hosts` file so that the hostname resolves to the correct IP address.
- Check whether the Linux machine is configured to accept packets from the host where JConsole is running (packet filtering is built in the Linux kernel). You can enter the command, `/sbin/iptables --list`, to determine whether an external client is allowed to connect to the JMX server.

  Use the following command to add a rule to allow an external client such as JConsole to connect:

  ```
  /usr/sbin/iptables -I INPUT -s JconsoleHost -p tcp --destina
  tion-port JMXRemotePort -j ACCEPT
  ```

  Where *`JconsoleHost`* is either the hostname or the IP address of the host on which JConsole is running and *`JMXRemotePort`* is the IP port exposed by the JMX server.

Chapter 12. Configuring JAAS Security

Alternative JAAS Realms

Overview

The Java Authentication and Authorization Service (JAAS) is a pluggable authentication service, which is implemented by a *login module*. A particular instance of a JAAS service is known as a JAAS realm and is identified by a *realm name*.

Applications integrated with JAAS must be configured to use a specific realm, by specifying the realm name.

Default realm

The default realm in Fuse ESB Enterprise is identified by the `karaf` realm name. The standard administration services in Fuse ESB Enterprise (SSH remote console, JMX port, and so on) are all configured to use the `karaf` realm by default.

Available realm implementations

Fuse ESB Enterprise provides the following alternative JAAS realm implementations:

- "Standalone JAAS realm" on page 86.
- "Fabric JAAS realm" on page 87.
- "LDAP JAAS realm" on page 87.

Standalone JAAS realm

In a standalone container, the `karaf` realm installs two JAAS login modules, which are used in parallel:

`PropertiesLoginModule`

Authenticates username/password credentials and stores the secure user data in the *`InstallDir`*`/etc/users.properties` file.

`PublickeyLoginModule`

Authenticates SSH key-based credentials (consisting of a username and a public/private key pair). Secure user data is stored in the *`InstallDir`*`/etc/keys.properties` file.

Fabric JAAS realm

In a fabric, a `karaf` realm based on the `ZookeeperLoginModule` login module is automatically installed in every container (the `fabric-jaas` feature is included in the default profile) and is responsible for securing the SSH remote console and other administrative services. The Zookeeper login module stores the secure user data in the Fabric Registry.

Note

In containers where the standalone JAAS realm and the Fabric JAAS realm are both installed, the Fabric JAAS realm takes precedence, because it defines a `karaf` realm with a *higher rank*.

LDAP JAAS realm

It is also possible to configure a container to use an LDAP login module with JAAS. For details of how to set this up, see LDAP Authentication Tutorial in *Security Guide*.

JAAS Console Commands

Editing user data from the console

Fuse ESB Enterprise provides a set of `jaas:*` console commands, which you can use to edit JAAS user data from the console. This works both for standalone JAAS realms and for Fabric JAAS realms.

Note

The `jaas:*` console commands are not compatible with the LDAP JAAS module.

Standalone realm configuration

A standalone container (which uses the JAAS `PropertiesLoginModule` and the `PublickeyLoginModule`) maintains its own database of secure user data, independently of any other containers. To configure the user data for a standalone container, you must log into the specific container (see Connecting and Disconnecting Remotely on page 45) whose data you want to modify. Each standalone container must be configured separately.

To start editing the standalone JAAS user data, you must first specify the JAAS realm that you want to modify. To see the available realms, enter the `jaas:realms` command, as follows:

```
karaf@root> jaas:realms
Index Realm                Module Class

    1 karaf               org.apache.karaf.jaas.modules.prop
erties.PropertiesLoginModule
    2 karaf                org.apache.karaf.jaas.modules.pub
lickey.PublickeyLoginModule
```

Both of these login modules are active in the default `karaf` JAAS realm. Enter the following console command to start editing the properties login module in the `karaf` realm:

```
karaf@root> jaas:manage --index 1
```

Fabric realm configuration

A container in a fabric (which uses the JAAS `ZookeeperLoginModule` by default) shares its secure user data with all of the other containers in the fabric and the user data is stored in the Fabric Registry. To configure the user data for a fabric, you can log into any of the containers. Because the user

data is shared in the registry, any modifications you make are instantly propagated to all of the containers in the fabric.

To start editing the fabric JAAS user data, you must first specify the JAAS login module you want to modify. In the context of fabric, you must modify the Zookeeper login module. For example, if you enter the `jaas:realms` console command, you might see a listing similar to this:

```
Index Realm                Module Class
    1 karaf                org.fusesource.fabric.jaas.ZookeepErLoginModule
    2 karaf                org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
    3 karaf                org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
```

The `ZookeeperLoginModule` login module has the highest priority and is used by the fabric (you cannot see this from the listing, but its realm is defined to have a higher rank than the other modules). In this example, the `ZookeeperLoginModule` has the index `1`, but it might have a different index number in your container.

Enter the following console command to start editing the fabric's JAAS realm (specifying the index of the `ZookeeperLoginModule`):

```
karaf@root> jaas:manage --index 1
```

Adding a new user to the JAAS realm

For example, consider how to add a new user, `jdoe`, to the JAAS realm.

First of all, start to manage the relevant JAAS realm as follows:

1. List the available realms and login modules by entering the following command:

   ```
   karaf@root> jaas:realms
   ```

2. Choose the login module to edit by specifying its index, *`Index`*, using a command of the following form:

   ```
   karaf@root> jaas:manage --index Index
   ```

Add the user, `jdoe`, with password, `secret`, by entering the following console command:

```
karaf@root> jaas:useradd jdoe secret
```

Add the `admin` role to `jdoe`, by entering the following console command:

```
karaf@root> jaas:roleadd jdoe admin
```

As a matter of fact, these changes are *not* applied right away. Initially, the changes are queued in a list of pending operations. To see this list, enter the `jaas:pending` console command, as follows:

```
karaf@root> jaas:pending
Jaas Realm:karaf Jaas Module:org.apache.karaf.jaas.modules.prop
erties.PropertiesLoginModule
UserAddCommand{username='jdoe', password='secret'}
RoleAddCommand{username='jdoe', role='admin'}
```

Now you can apply the changes by invoking `jaas:update`, as follows:

```
karaf@root> jaas:update
```

The new user entry is then persisted (either by writing to the remote container's `etc/users.properties` file, in the case of a standalone container, or by storing the user data in the Fabric Registry, in the case of a fabric).

Canceling pending changes

If you decide that you do *not* want to make the changes permanent after all, instead of invoking the `jaas:update` command, you could abort the pending changes using the `jaas:cancel` command, as follows:

```
karaf@root> jaas:cancel
```

Standalone Realm Properties File

Overview

The default JAAS realm used by a standalone container is implemented by the `PropertiesLoginModule` JAAS module. This login module stores its user data in a Java properties file in the following location:

```
InstallDir/etc/users.properties
```

Format of users.properties entries

Each entry in the `etc/users.properties` file has the following format (on its own line):

```
Username=Password,Role1,Role2,...
```

Changing the default username and password

The `etc/users.properties` file initially contains a commented out entry for a single user, `smx`, with password `smx` and role `admin`. It is strongly recommended that you create a new user entry that is *different* from the `smx` user example.

For example, you could create a new user in the following format:

```
Username=Password,admin
```

Where the `admin` role grants full administration privileges to this user.

Chapter 13. Logging

The Fuse ESB Enterprise runtime uses OPS4j Pax Logging as its logging mechanism. It is easily configured using the standard OSGi Admin mechanism and can be easily integrated with applications deployed in a container. The command console provides commands to manage the logs.

Fuse ESB Enterprise uses the *OPS4j Pax Logging* system. Pax Logging is an open source OSGi logging service that extends the standard OSGi logging service to make it more appropriate for use in enterprise applications. It uses Apache Log4j as the back-end logging service. Pax Logging has its own API, but it also supports the following APIs:

- Apache Log4j
- Apache Commons Logging
- SLF4J
- Java Util Logging

For more information on OPS4j Pax Logging see http://www.ops4j.org/projects/pax/logging/.

Logging Configuration

Overview

The logging system is configured by a combination of two OSGi Admin PIDs and one configuration file:

- `etc/system.properties`—the configuration file that sets the logging level during the container's boot process. The file contains a single property, `org.ops4j.pax.logging.DefaultServiceLog.level`, that is set to `ERROR` by default.

- `org.ops4j.pax.logging`—the PID used to configure the logging back end service. It sets the logging levels for all of the defined loggers and defines the appenders used to generate log output. It uses standard Log4j configuration. By default, it sets the root logger's level to `INFO` and defines two appenders: one for the console and one for the log file.

Tip

The console's appender is disabled by default. To enable it, add `log4j.appender.stdout.append=true` to the configuration For example, to enable the console appender in a standalone container, you would use the following commands:

```
FuseESB:karaf@root> config:edit org.ops4j.pax.logging
FuseESB:karaf@root> config:propappend log4j.append
er.stdout.append true
FuseESB:karaf@root> config:update
```

- `org.apache.karaf.log.cfg`—configures the output of the **log** console commands.

The most common configuration changes you will make are changing the logging levels, changing the threshold for which an appender writes out log messages, and activating per bundle logging.

Changing the log levels

The default logging configuration sets the logging levels so that the log file will provide enough information to monitor the behavior of the runtime and provide clues about what caused a problem. However, the default configuration will not provide enough information to debug most problems.

The most useful logger to change when trying to debug an issue with Fuse ESB Enterprise is the root logger. You will want to set its logging level to generate more fine grained messages. To do so you change the value of the `org.ops4j.pax.logging` PID's `log4j.rootLogger` property so that the logging level is one of the following:

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`
- `FATAL`
- `NONE`

Example 13.1 on page 95 shows the commands for setting the root loggers log level in a standalone container.

Example 13.1. Changing Logging Levels

```
FuseESB:karaf@root> config:edit org.ops4j.pax.logging
FuseESB:karaf@root> config:propset log4j.rootLogger "DEBUG, out,
osgi:VmLogAppender"
FuseESB:karaf@root> config:update
```

Changing the appenders' thresholds

When debugging a problem in Fuse ESB Enterprise you may want to limit the amount of logging information that is displayed on the console, but not the amount written to the log file. This is controlled by setting the thresholds for each of the appenders to a different level. Each appender can have a `log4j.appender.`*`appenderName`*`.threshold` property that controls what level of messages are written to the appender. The appender threshold values are the same as the log level values.

Example 13.2 on page 96 shows an example of setting the root logger to `DEBUG` but limiting the information displayed on the console to `WARN`.

Example 13.2. Changing the Log Information Displayed on the Console

```
FuseESB:karaf@root> config:edit org.ops4j.pax.logging
FuseESB:karaf@root> config:propset log4j.rootLogger "DEBUG, out,
osgi:VmLogAppender"
FuseESB:karaf@root> config:propappend log4j.appender.stdout.threshold
 WARN
FuseESB:karaf@root> config:update
```

Logging per bundle

It is possible to reconfigure Fuse ESB Enterprise logging so that it writes one log file for each bundle, instead of writing all of the log messages into a single log file. This feature is enabled by adding the Log4j `sift` appender to the Log4j root logger as shown in Example 13.3 on page 96.

Example 13.3. Enabling Per Bundle Logging

```
FuseESB:karaf@root> config:edit org.ops4j.pax.logging
FuseESB:karaf@root> config:propset log4j.rootLogger "INFO, out, sift,
 osgi:VmLogAppender"
FuseESB:karaf@root> config:update
```

After restarting the container, you can see that each *`BundleName`* bundle now has its own log file, located at `data/log/`*`BundleName`*`.log`.

This is the behavior you will see with the default sift appender settings. You can edit this behavior using the sift appender configuration settings in `org.ops4j.pax.logging.cfg`.

Logging per Application

Overview

Using Mapped Diagnostic Context (MDC) logging, you create a separate log file for each of your applications. The basic idea of MDC logging is that you associate each logging message with a particular context (for example, by associating it with a set of key-value pairs). Later on, when it comes to writing the log stream, you can use the context data to sort or filter the logging messages in various ways.

Note

MDC logging is supported only by log4j and slf4j.

Application key

To use MDC logging, you must define a unique MDC key for each of your applications. The MDC key is a string that is associated with one application or logging context. At runtime, you can then use the application key to sort logging messages and write them into separate files for each application key.

Enabling per application logging

To enable per application logging:

1. In each of your applications, edit the Java source code to define a unique application key.

   If you are using slf4j, add the following static method call to your application:

   ```
   org.slf4j.MDC.put("app.name","MyFooApp");
   ```

   If you are using log4j, add the following static method call to your application:

   ```
   org.apache.log4j.MDC.put("app.name","MyFooApp");
   ```

2. Edit the `etc/org.ops4j.pax.logging` PID to customize the sift appender.

   a. Set `log4j.appender.sift.key` to `app.name`.

b. Set `log4j.appender.sift.appender.file` to `=${karaf.data}/log/$\\{app.name\\}.log`.

3. Edit the `etc/org.ops4j.pax.logging` PID to add the sift appender to the root logger.

```
FuseESB:karaf@root> config:edit org.ops4j.pax.logging
FuseESB:karaf@root> config:propset log4j.rootLogger "INFO, out,
 sift, osgi:VmLogAppender"
FuseESB:karaf@root> config:update
```

Log Commands

The Fuse ESB Enterprise console provides the following commands for managing logging output:

log:display
Displays the most recent log entries. By default, the number of entries returned and the pattern of the output depends on the `size` and `pattern` properties in the `org.apache.karaf.log.cfg` file. You can override these using the `-p` and `-d` arguments.

log:display-exception
Displays the most recently logged exception.

log:get
Displays the current log level.

log:set
Sets the log level.

log:tail
Continuously display log entries .

log:clear
Clear log entries.

Chapter 14. Persistence

The Fuse ESB Enterprise container caches information about its state and the artifacts deployed to it. It uses this data to make startup faster. You can configure how this information is stored on your file system.

Overview

Fuse ESB Enterprise containers store all of their persistent caches relative to its start location. It will create a `data` folder in the directory from which you launch the container. This folder is populated by folders storing information about the message broker used by the container, the OSGi framework, and the JBI container.

The data folder

The `data` folder is used by the Fuse ESB Enterprise runtime to store persistent state information. It contains the following folders:

`activemq`
: Contains persistent data needed by any Apache ActiveMQ brokers that are started by the container.

`cache`
: The OSGi bundle cache. The directory structure differs according to whether you are running Equinox (the default) or Felix as your OSGi container. In both cases, the cache contains a directory for each bundle, where the directory name corresponds to the bundle identifier number.

`generated-bundles`
: Contains bundles that are generated by the container. Typically these are to support deployed JBI artifacts.

`jbi`
: Contains a subdirectory for each JBI artifact deployed to the Fuse ESB Enterprise runtime. For JBI components the folder's name is generated by the component's name. For JBI service assemblies, the folder's name is the identifier of the bundle generated to support the service assembly.

`log`
: Contains the log files.

`maven`
: A temporary directory used by the Fabric Maven Proxy when uploading files.

`txlog`

Contains the log files used by the transaction management system. You can set the location of this directory in the `org.apache.aries.transaction.cfg` file

Changing the bundle cache location

By default, the bundle cache is stored in *`InstallDir`*`/data/cache`.

To specify an alternative location, modify the `org.osgi.framework.storage` property in `config.properties`.

If you use a relative path, the cache location is added to the root of the Fuse ESB Enterprise installation directory.

Flushing the bundle cache

You can configure Fuse ESB Enterprise to flush the bundle cache every time the runtime starts by setting the `org.osgi.framework.storage.clean` property to `onFirstInit` in `config.properties`. This property is set to `none` by default.

Changing the generated-bundle cache location

The generated-bundle cache is where the container caches bundles it creates to support JARs that are not supplied as OSGi bundles.

You can configure the location of this cache by changing the `felix.fileinstall.tmpdir` property in the `org.apache.felix.fileinstall-deploy.cfg` file.

Adjusting the bundle cache buffer

The `felix.cache.bufsize` property controls the size of the buffer used to copy bundles into the bundle cache. Its default value is 4096 bytes.

You can adjust this property by editing its value in the `config.properties` configuration file. The value is specified in bytes.

Chapter 15. Failover Deployments

Fuse ESB Enterprise provides failover capability using either a simple lock file system or a JDBC locking mechanism. In both cases, a container-level lock system allows bundles to be preloaded into the slave kernel instance in order to provide faster failover performance.

Using a Simple Lock File System

Overview

When you first start Fuse ESB Enterprise a lock file is created at the root of the installation directory. You can set up a master/slave system whereby if the master instance fails, the lock is passed to a slave instance that resides on the same host machine.

Configuring a lock file system

To configure a lock file failover deployment, edit the `etc/system.properties` file on both the master and the slave installation to include the properties in Example 15.1 on page 104.

Example 15.1. Lock File Failover Configuration

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.SimpleFileLock
karaf.lock.dir=PathToLockFileDirectory
karaf.lock.delay=10000
```

- `karaf.lock`—specifies whether the lock file is written.
- `karaf.lock.class`—specifies the Java class implementing the lock. For a simple file lock it should always be `org.apache.karaf.main.SimpleFileLock`.
- `karaf.lock.dir`—specifies the directory into which the lock file is written. This **must** be the same for both the master and the slave installation.
- `karaf.lock.delay`—specifies, in milliseconds, the delay between attempts to reaquire the lock.

Using a JDBC Lock System

Overview

The JDBC locking mechanism is intended for failover deployments where Fuse ESB Enterprise instances exist on separate machines.

In this scenario, the master instance holds a lock on a locking table hosted on a database. If the master loses the lock, a waiting slave process gains access to the locking table and fully starts its container.

Adding the JDBC driver to the classpath

In a JDBC locking system, the JDBC driver needs to be on the classpath for each instance in the master/slave setup. Add the JDBC driver to the classpath as follows:

1. Copy the JDBC driver JAR file to the *ESBInstallDir*/lib directory for each Fuse ESB Enterprise instance.
2. Modify the bin/karaf start script so that it includes the JDBC driver JAR in its CLASSPATH variable.

   For example, given the JDBC JAR file, *JDBCJarFile*.jar, you could modify the start script as follows (on a *NIX operating system):

```
...
# Add the jars in the lib dir
for file in "$KARAF_HOME"/lib/karaf*.jar
do
    if [ -z "$CLASSPATH" ]; then
        CLASSPATH="$file"
    else
        CLASSPATH="$CLASSPATH:$file"
    fi
done
CLASSPATH="$CLASSPATH:$KARAF_HOME/lib/JDBCJarFile.jar"
```

Note

If you are adding a MySQL driver JAR or a PostgreSQL driver JAR, you must rename the driver JAR by prefixing it with the karaf-

prefix. Otherwise, Apache Karaf will hang and the log will tell you that Apache Karaf was unable to find the driver.

Configuring a JDBC lock system

To configure a JDBC lock system, update the `etc/system.properties` file for each instance in the master/slave deployment as shown

Example 15.2. JDBC Lock File Configuration

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.DefaultJDBCLock
karaf.lock.level=50
karaf.lock.delay=10
karaf.lock.jdbc.url=jdbc:derby://dbserver:1527/sample
karaf.lock.jdbc.driver=org.apache.derby.jdbc.ClientDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

In the example, a database named sample will be created if it does not already exist. The first Fuse ESB Enterprise instance to acquire the locking table is the master instance. If the connection to the database is lost, the master instance tries to gracefully shutdown, allowing a slave instance to become master when the database service is restored. The former master will require manual restart.

Configuring JDBC locking on Oracle

If you are using Oracle as your database in a JDBC locking scenario, the `karaf.lock.class` property in the `etc/system.properties` file must point to `org.apache.karaf.main.OracleJDBCLock`.

Otherwise, configure the `system.properties` file as normal for your setup, as shown:

Example 15.3. JDBC Lock File Configuration for Oracle

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.OracleJDBCLock
karaf.lock.jdbc.url=jdbc:oracle:thin:@hostname:1521:XE
karaf.lock.jdbc.driver=oracle.jdbc.OracleDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
```

```
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

Note

The `karaf.lock.jdbc.url` requires an active Oracle system ID (SID). This means you must manually create a database instance before using this particular lock.

Configuring JDBC locking on Derby

If you are using Derby as your database in a JDBC locking scenario, the `karaf.lock.class` property in the `etc/system.properties` file should point to `org.apache.karaf.main.DerbyJDBCLock`. For example, you could configure the `system.properties` file as shown:

Example 15.4. JDBC Lock File Configuration for Derby

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.DerbyJDBCLock
karaf.lock.jdbc.url=jdbc:derby://127.0.0.1:1527/dbname
karaf.lock.jdbc.driver=org.apache.derby.jdbc.ClientDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

Configuring JDBC locking on MySQL

If you are using MySQL as your database in a JDBC locking scenario, the `karaf.lock.class` property in the `etc/system.properties` file must point to `org.apache.karaf.main.MySQLJDBCLock`. For example, you could configure the `system.properties` file as shown:

Example 15.5. JDBC Lock File Configuration for MySQL

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.MySQLJDBCLock
karaf.lock.jdbc.url=jdbc:mysql://127.0.0.1:3306/dbname
karaf.lock.jdbc.driver=com.mysql.jdbc.Driver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
```

```
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

Configuring JDBC locking on PostgreSQL

If you are using PostgreSQL as your database in a JDBC locking scenario, the `karaf.lock.class` property in the `etc/system.properties` file must point to `org.apache.karaf.main.PostgreSQLJDBCLock`. For example, you could configure the `system.properties` file as shown:

Example 15.6. JDBC Lock File Configuration for PostgreSQL

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.PostgreSQLJDBCLock
karaf.lock.jdbc.url=jdbc:postgresql://127.0.0.1:5432/dbname
karaf.lock.jdbc.driver=org.postgresql.Driver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=0
```

JDBC lock classes

The following JDBC lock classes are currently provided by Apache Karaf:

```
org.apache.karaf.main.DefaultJDBCLock
org.apache.karaf.main.DerbyJDBCLock
org.apache.karaf.main.MySQLJDBCLock
org.apache.karaf.main.OracleJDBCLock
org.apache.karaf.main.PostgreSQLJDBCLock
```

Container-level Locking

Overview

Container-level locking allows bundles to be preloaded into the slave kernel instance in order to provide faster failover performance. Container-level locking is supported in both the simple file and JDBC locking mechanisms.

Configuring container-level locking

To implement container-level locking, add the following to the `etc/system.properties` file on each system in the master/slave setup:

Example 15.7. Container-level Locking Configuration

```
karaf.lock=true
karaf.lock.level=50
karaf.lock.delay=10
```

The `karaf.lock.level` property tells the Fuse ESB Enterprise instance how far up the boot process to bring the OSGi container. Bundles assigned the same start level or lower will then also be started in that Fuse ESB Enterprise instance.

Bundle start levels are specified in `etc/startup.properties`, in the format *`BundleName`*`.jar=level`. The core system bundles have levels below 50, where as user bundles have levels greater than 50.

Table 15.1. Bundle Start Levels

Start Level	Behavior
1	A 'cold' standby instance. Core bundles are not loaded into container. Slaves will wait until lock acquired to start server.
<50	A 'hot' standby instance. Core bundles are loaded into the container. Slaves will wait until lock acquired to start user level bundles. The console will be accessible for each slave instance at this level.

Start Level	Behavior
>50	This setting is not recommended as user bundles will be started.

Avoiding port conflicts

When using a 'hot' spare on the same host you need to set the JMX remote port to a unique value to avoid bind conflicts. You can edit the `servicemix` start script (or the `karaf` script on a child instance) to include the following:

```
DEFAULT_JAVA_OPTS="-server $DEFAULT_JAVA_OPTS -Dcom.sun.man
agement.jmxremote.port=1100 -Dcom.sun.management.jmxremote.au
thenticate=false"
```

Chapter 16. Configuring JBI Component Thread Pools

The JBI components included in Fuse ESB Enterprise use a thread pool to process message exchanges. You can configure each component's thread pool independently.

Overview

The JBI components are multi-threaded. Each one maintains a thread pool that it uses to process message exchanges. These thread pools are configured using three properties that control the minimum number of threads in the pool, the maximum number of threads in the pool, and the depth of the component's job queue.

Component configuration PIDs

The thread pool properties can be customised for a particular JBI component, `ComponentName`, by adding the relevant PID to the OSGi Admin service. Each JBI component has a corresponding PID that matches the pattern `org.apache.servicemix.components.ComponentName`.

The thread pool properties can also be configured using a JMX console.

Important

The component needs to be restarted for changes to take effect.

Thread pool properties

Table 16.1 on page 111 lists the properties used to configure component thread properties.

Table 16.1. Component Thread Pool Properties

Property	Default	Description
`corePoolSize`	8	Specifies the minimum number of threads in a thread pool. If the number of available threads drops below this limit, the runtime will always create a new thread to handle the job.
`maximumPoolSize`	32	Specifies the maximum number of threads in a thread pool. Setting this

Property	Default	Description
		property to -1 specifies that it is unbounded.
queueSize	256	Specifies the number of jobs allowed in a component's job queue.

Thread selection

When a component receives a new message exchange it choose the thread to process the exchange as follows:

1. If the component's thread pool is smaller than the `corePoolSize`, a new thread is created to process the task.
2. If less than `queueSize` jobs are in the component's job queue, the task is placed on the queue to wait for a free thread.
3. If the component's job queue is full and the thread pool has less than `maximumPoolSize` threads instantiated, a new thread is created to process the task.
4. The job is processed by the current thread.

Example

Example 16.1 on page 112 shows the configuration for a component whose thread pool can have between 10 and 200 threads.

Example 16.1. Component Thread Pool Configuration

```
corePoolSize = 10
maximumPoolSize = 200
...
```

Chapter 17. Applying Patches

Fuse ESB Enterprise supports incremental patching. FuseSource will supply you with easy to install patches that only make targeted changes to a deployed container.

Incremental patching allows you apply targets fixes to a container without needing to reinstall an updated version of Fuse ESB Enterprise. It also allows you to easily back the patch out if it causes problems with your deployed applications.

Patches are ZIP files that contain the artifacts needed to update a targeted set of bundles in a container. The patch file includes a `.patch` file that lists the contained artifacts. The artifacts are typically one or more bundles. They can, however, include configuration files and feature descriptors.

You get a patch file in one of the following ways:

- Customer Support sends you a patch.
- Customer Support sends you a link to download a patch.

The process of applying a patch to a container depends on how the container is deployed:

- standalone—the container's command console's **patch** shell has commands for managing the patching process
- fabric—patching a fabric requires applying the patch to a profile and then applying the profile to a container

  Fuse Management Console is the recommended way to patch containers in a fabric. See the Fuse Management Console patching documentation[1] for more information.

[1] http://http://fusesource.com/docs/fmc/1.0/user_guide/FMCUGPatchPart.html

Patching a Standalone Container

Overview

Patching a standalone container directs the container to load the patch versions of artifacts instead of the non-patch versions. The **patch** shell provides commands to patches to the container's environment, see which bundles are effected by applying the patch, apply the patch to the container, and back the patch out if needed.

To make sure that the a patch can be rolled back Fuse ESB Enterprise applies the patch in a non-destructive manner. The patching process does not overwrite the artifacts included in the original installation. The patched artifacts are placed in the container's `system` folder. When the patch is applied, the container's configuration is changed so that it points to the patched artifacts instead of the artifacts from the original installation. This makes it easy for the system to be restored to its original state or to selectively back out patches.

Important

Patches **do not** persist across installations. If you delete and reinstall a Fuse ESB Enterprise instance you will need to download the patches and reapply them.

Applying a patch

To apply a patch to a standalone container:

1. Add the patch to the container's environment using the **patch:add** command.

   Example 17.1 on page 114 shows the command for adding the patch contained in the patch file `patch.zip` from the local file system.

   Example 17.1. Adding a Patch to a Broker's Environment

   ```
   FuseMQ> patch:add file://patch.zip
   ```

   This command copies the specified patch file to the container's `system` folder and unpacks it.

2. Simulate installing the patch using the **patch:simulate** command.

   This will generate a log of the changes that will be made to the container when the patch is installed, but will not make any actual changes to the container.

Tip

The **patch:list** command will display a list of all patches added to the container's `system` folder.

3. Review the simulation log to understand the changes that will be made to the container.
4. Apply the patch to the container using the **patch:install** command.

Warning

Running **patch:install** before the container is fully started and all of the bundles are active will cause the container to hang.

Tip

The **patch:list** command will display a list of all patches added to the container's `system` folder.

The container will need to restart to apply the patch. If you are using a remote console, you will lose the connection to the container. If you are using the container's local console, it will automatically reconnect when the container restarts.

Rolling back a patch

Occasionally a patch will not work or introduce new issues to a container. In these cases you can easily back the patch out of the system and restore it pre-patch behavior using the **patch:rollback** command. As shown in Example 17.2 on page 115, the command takes the name of patch to be backed out.

Example 17.2. Rolling Back a Patch

```
FuseMQ> patch:rollback patch1
```

Tip

The **patch:list** command will display a list of all patches added to the container's `system` folder.

The container will need to restart to rollback the patch. If you are using a remote console, you will lose the connection to the container. If you are using the container's local console, it will automatically reconnect when the container restarts.

Patching a Container in a Fabric

Overview

The bundles loaded by a container in a fabric is controlled by the container's Fabric Agent. The agent inspects the profiles applied to the container to determine what bundles to load, and the version of each bundle, and then loads the specified version of each bundle for the container.

A patch typically includes a new version of one or more bundles, so to apply the patch to container in a fabric you need to update the profiles applied to it. This will cause the Fabric Agent to load the patched versions of the bundles.

Fuse Management Console is the recommended tool for patching containers in a fabric. However, the command console's **fabric** shell also provides the commands needed to patch containers running in a fabric.

Procedure

Patching a container in a fabric involves:

1. Getting a patch file.
   - Customer Support sends you a patch.
   - Customer Support sends you a link to download a patch.
   - You, or your organization, generate a patch file for an internally created application.
2. Uploading one or more patch files to the fabric's Maven repository.
3. Applying the patch(es) to a profile version.

   This creates a new profile version that points to the new versions of the patched bundles and repositories.
4. Migrate one or two containers to the patched profile version to ensure that the patch does not introduce any new issues.
5. After you are certain that the patch works, migrate the remaining containers in the fabric to the patched version.

Using Fuse Management Console

Fuse Management Console is the easiest and most verbose method of patching containers in a fabric. The Fuse Management Console **Patching** tab uploads patches to a fabric's Maven repository and applies the patch to a specified

profile version. You can then use Fuse Management Console to roll the patch out to all of the containers in the fabric.

See the Fuse Management Console patching documentation[2] for more information.

Using the command console

The Apache ActiveMQ command console can also be used to patch containers running in a fabric. To patch a fabric container:

1. Upload the patch file to the fabric's Maven repository.
2. Create a new profile version to which the patch will be applied.
3. Modify all of the profiles in the new version which require the patch.
4. Use the **fabric:container-upgrade** command to roll the patch out to the containers running in the fabric.

[2] http://http://fusesource.com/docs/fmc/1.0/user_guide/FMCUGPatchPart.html

Chapter 18. Configuring a Fabric's Maven Proxy

The Fabric Ensemble creates a Maven proxy to access the repositories from which artifacts are distributed to the fabric's containers. You can modify the default settings to use a different set of repositories or the make an internal repository accessible.

Overview

The Fabric Ensemble creates a Maven proxy to facilitate access to the artifacts required the containers in the fabric. Each Fabric Server deployed in the fabric runs an instance of a Maven proxy. The ensemble aggregates all of the proxies so that it appears to the Fabric Agents as a single Maven proxy.

Tip

Advanced users can configure each Fabric Server to act as a proxy for a different set of repositories. However, this is not a recommended set up.

The Fabric Agents use the fabric's Maven proxy to access the known repositories. This ensures that all of the containers use the same set of repositories and artifacts.

Tip

Fuse IDE provides tooling for uploading bundles using the Maven proxy. You can also add the fabric's Maven Proxy to a POM file so that bundles can be distributed to the ensemble as part of an automated build process.

Default repositories

By default a fabric's Maven proxy is configured to be a proxy for the following Maven repositories:

- Maven Central (`http://repo1.maven.org/maven2`)
- Fuse Releases (`http://repo.fusesource.com/nexus/content/repositories/releases`)

- Fuse Early Access (`http://repo.fusesource.com/nexus/content/groups/ea`)
- SpringSource (`http://repository.springsource.com/maven/bundles/release`, `http://repository.springsource.com/maven/bundles/external`)
- Scala Tools (http://scala-tools.org/repo-releases)
- User's Local (`~/.m2/repository`)

Changing the repositories

To change the repositories the ensemble proxies:

1. Create a new profile version.

   From the command console this is done using the **fabric:version-create** command. See fabric:version-create in *Console Reference* for more information.

2. Change the `org.ops4j.pax.url.mvn.repositories` property in the `org.fusesource.fabric.agent` PID of the `default` profile.

   Example 18.1 on page 120 shows the console command for editing this property.

Example 18.1. Configuring the Maven Proxy URL

```
FuseESB:karaf@root> fabric:profile-edit -p org.fusesource.fabric.agent/org.ops4j.pax.url.mvn.repositories= \
    http://repo1.maven.org/maven2, \
    http://repo.fusesource.com/nexus/content/repositories/releases, \
    http://repo.fusesource.com/nexus/content/groups/ea, \
    http://repository.springsource.com/maven/bundles/release, \
    http://repository.springsource.com/maven/bundles/external, \
    http://scala-tools.org/repo-releases default
```

Note

The `org.fusesource.fabric.agent` PID is refined in all of the fabric profiles. Setting the proxy URL, the `org.ops4j.pax.url.mvn.repositories` property, in the `default` profile ensures that all of the other fabric profiles share the same Maven proxy setting.

Important

The `fabric` profile's `org.fusesource.fabric.maven` PID, which ultimately controls the Maven proxy, imports its value from the `default` profile's `org.fusesource.fabric.agent` PID. You should **not** change the settings of the `org.fusesource.fabric.maven` PID.

3. Roll the changes out the fabric by upgrading the containers to the new profile version.

Important

You cannot test this configuration change out on a few containers to validate it. The change **must** be made to the entire fabric or it will result in conflicts.

Index

R

S