Red Hat JBoss Fuse 6.3

Installation on Apache Karaf

Installing Red Hat JBoss Fuse on the Apache Karaf container

Last Updated: 2018-07-16

Red Hat JBoss Fuse 6.3 Installation on Apache Karaf

Installing Red Hat JBoss Fuse on the Apache Karaf container

JBoss A-MQ Docs Team
Content Services
fuse-docs-support@redhat.com

Legal Notice

Copyright © 2016 Red Hat.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

It is easy to install Red Hat JBoss Fuse and tailor the installation to a particular environment.

Table of Contents

CHAPTER 1. INSTALLATION PREREQUISITES

Abstract

Before attempting to install and use Red Hat JBoss Fuse, make sure your system meets the minimum requirements.

SUPPORTED PLATFORMS

Red Hat tests and supports Fuse products in the configurations listed at Red Hat JBoss Fuse Supported Configurations.

JAVA RUNTIME

For details of the Java runtimes supported by Red Hat JBoss Fuse, see Red Hat JBoss Fuse Supported Configurations.

WARNING

Do *not* install the Java runtime under a directory path that includes whitespace. For example, `C:\Program Files\Java\jde7` is not an acceptable install path and will lead to unpredictable errors in Red Hat JBoss Fuse at run time.

SUPPORTED STANDARDS

Red Hat JBoss Fuse supports the standards and protocols listed at Red Hat JBoss Fuse Supported Standards.

HARDWARE REQUIREMENTS

The minimum hardware requirements for installing Red Hat JBoss Fuse are:

- Full Installation
  - 700 MB of free disk space
  - 2 GB of RAM

In addition to the disk space required for the base installation, a running system will require space for caching, persistent message stores, and other functions.

CHAPTER 2. INSTALLATION TYPES

2.1. OVERVIEW OF INSTALLATION PACKAGES

Red Hat JBoss Fuse 6.3 is delivered in packages.

The standard install package is available for download from the Red Hat Customer Portal. It installs Apache Karaf, Apache Camel, Apache ActiveMQ, Apache CXF, SwitchYard, and the Fuse Management Console (Hawtio).

It is possible to create your own *custom assembly* of JBoss Fuse 6.3, containing a customized subset of the JBoss Fuse features and bundles. You can use this approach to replace the *Minimal* and the *Medium* install types, which were available for versions of JBoss Fuse prior to 6.2.

To create a custom assembly of JBoss Fuse, see Appendix C, *Generating a Custom Assembly or an Offline Repository*.

CHAPTER 3. INSTALLING

Abstract

Red Hat JBoss Fuse is installed by unpacking an archive system on a system. This provides an easy way for a developer to get up and running.

GETTING THE ARCHIVE

You can download the Red Hat JBoss Fuse archive from the Red Hat Customer Portal>Downloads>Red Hat JBoss Middleware>Downloads page, after you register and login to your customer account.

Once logged in:

1. Select **Fuse**, listed under **Integration Platforms**, in the sidebar menu.
2. Select **6.3** from the **Version** drop-down list on the **Software Downloads** page.
3. Click the **Download** button next to the Red Hat JBoss Fuse 6.3 on Karaf Installer file.

UNPACKING THE ARCHIVE

Red Hat JBoss Fuse is packaged as a **.zip** file. Using a suitable archive tool, such as **Zip**, unpack Red Hat JBoss Fuse into a directory to which you have full access.

WARNING

Do not unpack the archive file into a folder that has spaces in its path name. For example, do not unpack into **C:\Documents and Settings\Greco Roman\Desktop\fusesrc**.

WARNING

Do not unpack the archive file into a folder that has any of the following special characters in its path name: #, %, ^, ".

USING THE IBM JDK

If you are using the IBM JDK, remove the **saaj-api** jar from the ***installDir*/lib/endorsed** library using the following command:

1. ```
   rm lib/endorsed/org.apache.servicemix.specs.saaj-api-1.3-2.7.0.jar
   ```
   ```

2. Before invoking the **./bin/fuse** script:, set the **JAVA_OPTS** environment variable as follows:

```
JAVA_OPTS=-Xshareclasses:none
```

INSTALLING RED HAT JBOSS DEVELOPMENT TOOLS

The JBoss Fuse development environment is included in JBoss Developer Studio starting with Developer Studio 11.0. After you download Developer Studio, see the Red Hat JBoss Developer Studio 11.0 Installation Guide. During installation, in the **Select Additional Features to Install** page, be sure to select **JBoss Fuse Tooling**.

Switchyard is included in JBoss Developer Studio Integration Stack. To install Switchyard tooling, see the Red Hat JBoss Developer Studio Integration Stack Installation Guide.

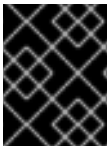
CHAPTER 4. ADDING A REMOTE CONSOLE USER

Abstract

The server's remote command console is not configured with a default user. Before you can connect to the server's console remotely, you must add a user to the configuration.

4.1. ADD A USER

Red Hat JBoss Fuse is not installed with a default user for the remote console. Before you can remotely manage a server, you must add a user by editing ***InstallDir*/etc/users.properties**.



IMPORTANT

The information in this file is unencrypted so it is not suitable for environments that require strict security.

To add a user:

1. Open ***InstallDir*/etc/users.properties** in your favorite text editor.
2. Locate the line **#admin=admin,admin,manager,viewer,Operator, Maintainer, Deployer, Auditor, Administrator, SuperUser**.

   This line specifies a user **admin** with the password **admin** and comma-separated list of roles.
3. Uncomment the line by removing the leading **#**.
4. Change the first **admin** to the desired user name.
5. Change the second **admin** to the desired password.
6. Leave the role list, **admin,manager,viewer,Operator, Maintainer, Deployer, Auditor, Administrator, SuperUser**, as is.
7. Save the changes.

CHAPTER 5. OFFLINE MODE

Abstract

You can run the JBoss Fuse container in offline mode (that is, without an Internet connection). But if you are planning to deploy custom applications to the container, it might be necessary to download additional dependencies (to a local Maven repository) before you can run the container in offline mode with these applications.

5.1. SUPPORTED FUNCTIONALITY

If you want to run the JBoss Fuse container in offline mode, it is necessary to distinguish between the following kinds of dependency:

- *Runtime dependencies*—the dependencies required to run the JBoss Fuse container, in its default configuration.
- *Build-time dependencies*—the dependencies required to build a custom application (which might include third-party libraries).

Here is a summary of what can be done in offline mode and what needs to be done in online mode (with an Internet connection):

- *Running the JBoss Fuse container in its default configuration*—is supported in offline mode. The default configuration of the JBoss Fuse container is specified by the **`featuresBoot`** property in the **`etc/org.apache.karaf.features.cfg`** file. The requisite dependencies are all provided in the **`system/`** sub-directory of the installation.
- *Installing additional features*—is, in general, *not* supported in offline mode. In principle, you can use the **`features:install`** command to install any of the features from the standard feature repositories (as specified by the **`featuresRepositories`** property in the **`etc/org.apache.karaf.features.cfg`** file), but the majority of these features must be downloaded from the Internet and are thus not supported in offline mode.
- *Deploying custom applications*—is, in general, *not* supported in offline mode. There may be some cases where an application with a minimal set of build-time dependencies is deployable offline, but in general, custom applications would have third-party dependencies that require an Internet connection (so that JAR files can be downloaded by Apache Maven).

If you do need to deploy an application with dependencies that are not available offline, you can use the Maven dependency plug-in to download the application's dependencies into a Maven offline repository. This customized Maven offline repository can then be distributed internally to any machines that do not have an Internet connection. For more details of this approach, see section "Offline Repository for a Maven Project" in "Fabric Guide".

CHAPTER 6. GETTING STARTED WITH SWITCHYARD ON APACHE KARAF

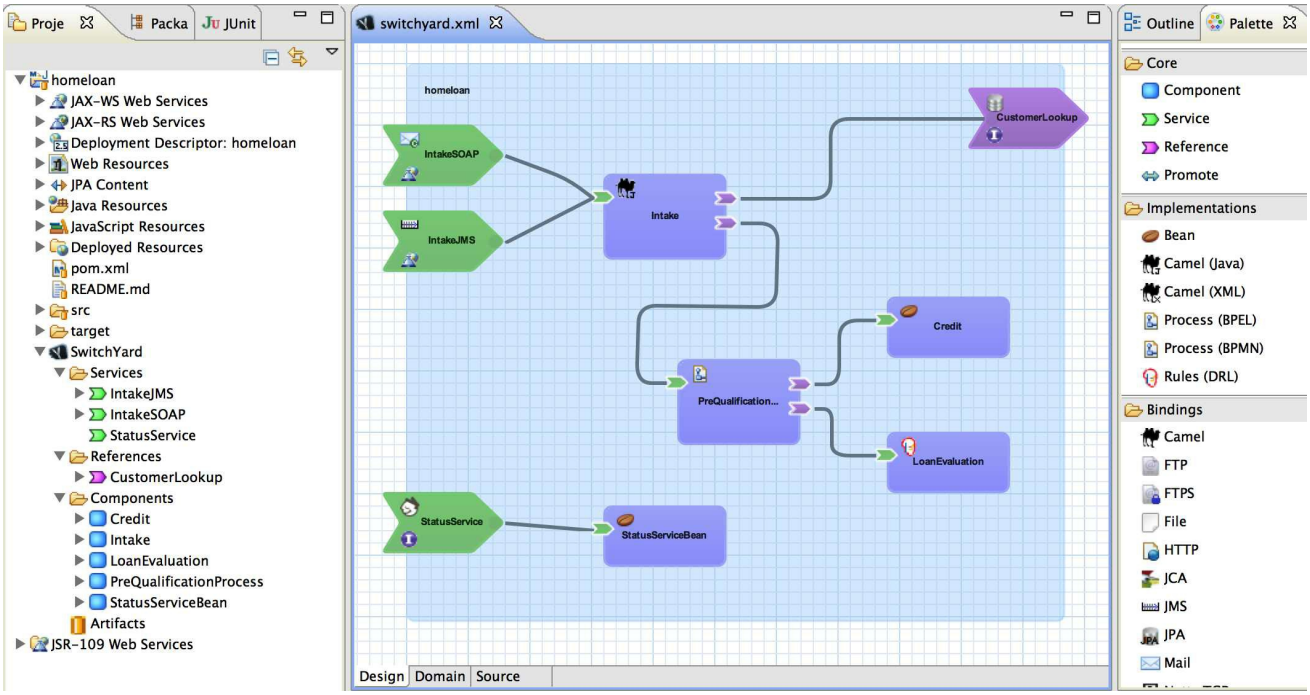
6.1. SWITCHYARD OVERVIEW

SwitchYard is a lightweight service delivery framework providing full lifecycle support for developing, deploying, and managing service-oriented applications.

It works with Apache Camel to provide a flexible integration runtime with comprehensive connectivity and transports.

SwitchYard tooling is provided with Red Hat JBoss Developer Studio Integration Stack. To install Switchyard tooling, see the Red Hat JBoss Developer Studio Integration Stack installation guide available from the customer portal.

Figure 6.1. SwitchYard Visual Editor

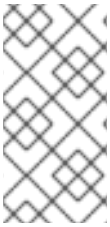


6.2. INSTALLING SWITCHYARD ON APACHE KARAF

When a SwitchYard application is installed, its dependencies will be installed automatically as long as its `features` file is defined correctly.

You can test this by installing a quickstart application on a fresh installation of the JBoss Fuse Apache Karaf container.

You can also install SwitchYard and its related components manually.



NOTE

Each SwitchYard component and application is provided as a feature. Packaged features (including quickstart applications) are defined in the SwitchYard features file located at `installDir/system/org/switchyard/karaf/switchyard/2.1.0.redhat-630187/switchyard-2.1.0.redhat-630187-core-features.xml`.

Procedure 6.1. Install SwitchYard Manually

1. Start the Apache Karaf container:

```
./bin/fuse
```

2. List the available SwitchYard features by entering the following console command. If this is a fresh instance, they will all be uninstalled at this point.

```
JBossFuse:karaf@root> features:list | grep switchyard
```

3. Install the core SwitchYard components by entering the following console command:

```
JBossFuse:karaf@root> features:install switchyard
```

4. List the SwitchYard features again. You will now see that SwitchYard (the switchyard feature) has been installed, along with its dependencies as defined by the SwitchYard features file.

6.3. SWITCHYARD QUICKSTART APPLICATIONS

You can find a variety of SwitchYard applications in ***installDir*/quickstarts/switchyard**.

Quickstart applications are typically installed from the Karaf command line console as follows:

1. Start the Apache Karaf container:

```
./installDir/bin/fuse
```

2. Install the application by entering the following console command:

```
JBossFuse:karaf@root> features:install switchyard-
quickstart-APP_NAME
```

You can find out more about running each quickstart application by reading the accompanying **Readme.md** file for each.

NOTE

There is no need to add the features URL as described by the **Readme.md** file. SwitchYard features are already defined in ***installDir*/system/org/switchyard/karaf/switchyard/2.1.0.redhat-630187/switchyard-2.1.0.redhat-630187-core-features.xml**.

6.4. INSTALLING YOUR FIRST QUICKSTART APPLICATION

The following procedure demonstrates how to install and run the **camel-quartz-binding** quickstart application. This application makes use of the **camel-quartz** component which schedules a logging service to run every second.

1. Start the Apache Karaf container:

```
./bin/fuse
```

2. List the available SwitchYard features by entering the following console command. If this is a fresh instance, they will all be uninstalled at this point.

```
JBossFuse:karaf@root> features:list | grep switchyard
```

   Install the **camel-quartz-binding** application by entering the following console command:

```
JBossFuse:karaf@root> features:install switchyard-quickstart-camel-
quartz-binding
```

3. List the SwitchYard features again. You will now see that the camel-quartz-binding application (the switchyard-quickstart-camel-quartz-binding feature) has been installed, along with its dependencies as defined by the SwitchYard features file.

4. View the output from the installed (and running) application by entering the following console command:

```
JBossFuse:karaf@root> log:tail
```

NOTE

Find out more about this quickstart application in ***installDir*/quickstarts/switchyard/camel-quartz-binding/Readme.md**.

6.5. WHAT'S NEXT?

Find out more about SwitchYard.

APPENDIX A. INSTALLING THE APACHE COMPONENTS

Abstract

Red Hat JBoss Fuse includes standard distributions of Apache Camel and Apache CXF in the installation's **`extras`** directory.

A.1. ZIP ARCHIVES FOR THE APACHE COMPONENTS

If you want to use a standard distribution of Apache Camel or Apache CXF, without the OSGi container or Fuse Fabric, use the archived versions in the installation's **`extras`** directory.

The following archives are provided for all supported platforms:

- apache-camel-2.17.0.redhat-630187.zip
- apache-cxf-3.1.5.redhat-630187.zip

You can copy these files to the desired location and decompress them using the appropriate utility for your platform.

WARNING

Do not unpack an archive file into a folder that has spaces in its path name. For example, do not unpack into **`C:\Documents and Settings\Greco Roman\Desktop\fusesrc`**.

APPENDIX B. RED HAT JBOSS FUSE MAVEN REPOSITORIES

Abstract

Red Hat JBoss Fuse strongly supports Maven, an open source build system available from Apache Maven. To use Maven to build your projects, you need to specify, in a Maven `settings.xml` file, where required artifacts are located.

For details on setting up Maven to work with Red Hat JBoss Fuse, see chapter "Building with Maven" in "Deploying into Apache Karaf".

B.1. MAVEN REPOSITORIES

The following repositories contain artifacts your projects may need:

- JBoss Fuse repository

  Provides access to the artifacts in the Red Hat Maven repository. This repository is required:

  `https://maven.repository.redhat.com/ga`

- JBoss Fuse early access repository

  Provides access to the artifacts in the Red Hat Maven repository for early access releases. This repository is optional:

  `https://maven.repository.redhat.com/earlyaccess/all`

APPENDIX C. GENERATING A CUSTOM ASSEMBLY OR AN OFFLINE REPOSITORY

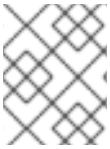
OVERVIEW

The Karaf container in JBoss Fuse comes with a large collection of features installed by default. While this offers the advantage of a highly functional container, it requires a large amount of memory and resources. For some deployment scenarios, it can be an advantage to strip down the Karaf container to use the minimal set of features required for your application. A custom assembly is the most effective way to create a stripped down Karaf container that deploys only the features needed for your application.

The same procedure that is used to generate a custom assembly also generates an *offline repository*.

CUSTOM ASSEMBLY

A *custom assembly* is a custom distribution of the Karaf container based on JBoss Fuse, which is generated using a specific toolchain of Maven plug-ins. The quickstart examples include a template, **`quickstart/custom`**, which demonstrates how this toolchain works and provides you with the starting point for creating your own custom assemblies.



NOTE

A custom assembly is generated using essentially the same toolchain as the official JBoss Fuse Karaf container distribution.

PATCHED CUSTOM ASSEMBLY

If your starting point is a JBoss Fuse rollup patch (which is, in fact, equivalent to a complete distribution of the JBoss Fuse product with patches already applied), and you then follow the steps for generating a custom assembly, what you end up with is a *patched custom assembly*. No actual patching step needs to be performed in this case, however, because the patch is already built-in to the custom assembly.

OFFLINE REPOSITORY

An **`offline repository`** is a Maven repository that contains a custom collection of Maven artifacts that would otherwise need to be downloaded from the Internet. For JBoss Fuse deployments that lack Internet access, the offline repository makes it possible to run the container in an isolated network by providing all of the Maven artifacts that your application might need. For details about how to install an offline repository in a standalone Karaf container, see chapter "Locating Artifacts with Maven and HTTP" in "Deploying into Apache Karaf".

QUICKSTART/CUSTOM EXAMPLE

The **`quickstart/custom`** example provides a template for generating your own custom assembly. The default configuration of the custom example includes a very minimal collection of Karaf features. You can easily customize this configuration to specify example which Karaf features you would like to include in the custom assembly.

PROCEDURE TO GENERATE A CUSTOM ASSEMBLY OR AN OFFLINE REPOSITORY

To generate a custom assembly, or a patched custom assembly, or an offline Maven repository, perform the following steps:

1. Choose the appropriate starting point for this procedure:

   - *To generate a custom assembly*—download and install the standard install from the Red Hat Customer Portal, as described in Chapter 3, *Installing*; or

   - *To generate a patched custom assembly*—download the latest rollup patch from the Red Hat Customer Portal.

2. Go to the ***InstallDir*/quickstarts/custom** project.

3. Customize the features that will be included in the custom assembly's **system/** directory (or in the generated offline repository) by editing the project's **pom.xml** file. The **patch** feature and the **fabric** feature are both *required*, and must be added to the configuration.

   Open the **quickstarts/custom/pom.xml** file in a text editor, search for the **features-maven-plugin** plug-in configuration, and add the **patch** feature and the **fabric** feature as children of the **configuration/features** XML element—for example:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    ...
    <build>
        <plugins>
            ...
            <!--
                Step 3: gather all the bundles for the features your
want to install
            -->
            <plugin>
                <groupId>org.apache.karaf.tooling</groupId>
                <artifactId>features-maven-plugin</artifactId>
                <version>2.4.1</version>
                <executions>
                    <execution>
                        <id>add-features</id>
                        <goals>
                            <goal>add-features-to-repo</goal>
                        </goals>
                        <phase>generate-resources</phase>
                        <configuration>
                            ...
                            <features>
                                <!-- TODO: add the feature you
want to use in your custom assembly to the list below -->
                                <feature>shell</feature>
                                <feature>admin</feature>
                                <feature>management</feature>
                                <feature>camel-blueprint</feature>
                                <feature>camel-jaxb</feature>
                                <feature>patch</feature>
                                <feature>fabric</feature>
```

```
                            </features>

<repository>target/repository</repository>
                        </configuration>
                    </execution>
                </executions>
                ...
            </plugin>
            ...
        </plugins>
    </build>
    ...
</project>
```

4. *(Not required for an offline repository)* Customize the list of features that will be installed automatically when the custom assembly's container first starts. The **patch** feature and the **fabric** feature are both *required*, and must be added to this list.

   Open the **src/main/filtered-resources/etc/org.apache.karaf.features.cfg** file and modify the **featuresBoot** property setting as required, adding at least the **fabric** feature and the **patch** feature to the list:

   ```
   ...
   # TODO: list the boot features for your own assembly below
   featuresBoot=shell,admin,management,fabric,patch,camel-
   blueprint,camel-jaxb
   ```

5. Install the JBoss Fuse distribution into your local Maven repository. In the following example, you would replace xxx with the version number in your artifact:

   ```
   mvn install:install-file \
        -Dfile=/path/to/fuse/jboss-fuse-karaf-6.3.0.redhat-xxx.zip \
        -DgroupId=org.jboss.fuse \
        -DartifactId=jboss-fuse-karaf \
        -Dversion=6.3.0.redhat-xxx \
        -Dpackaging=zip \
        -DgeneratePom=false
   ```

6. After customizing the configuration of the **quickstarts/custom** project, build the project as follows:

   ```
   mvn clean package
   ```

7. The offline repository is now available in the following directory:

   ```
   InstallDir/quickstarts/custom/target/repository
   ```

   If the purpose of this build is to create an *offline repository*, the procedure is now finished and you can skip the remaining steps.

8. The result of building this project is a custom assembly file, in the following location:

```
InstallDir/quickstarts/custom/target/custom-distro-ProductVersion-
bin.zip
```

9. The custom assembly requires some additional modifications. In preparation for this, unpack the custom assembly file, **custom-distro-*ProductVersion*-bin.zip**, into a convenient working location on the file system:

```
Working/custom-distro-ProductVersion-bin
```

10. To reduce the size of the custom assembly, delete the **extras/** subdirectory from **custom-distro-*ProductVersion*-bin** (saving about 67 MB).

11. Create a new baseline archive for the custom assembly, as follows:

    1. Make a copy of the entire ***Working*/custom-distro-*ProductVersion*-bin** directory, to give a copy like the following:

```
Working/custom-distro-ProductVersion-bin(copy)
```

    2. Rename the copied directory, **custom-distro-*ProductVersion*-bin(copy)**, to the following:

```
jboss-fuse-karaf-ProductVersion-baseline
```

       For example, if the ***ProductVersion*** is 6.3.0.redhat-187, the new directory name must be **jboss-fuse-karaf-6.3.0.redhat-187-baseline**.

    3. Delete the **system/** subdirectory from the **jboss-fuse-karaf-*ProductVersion*-baseline** directory. In addition, if any of the following subdirectories are present, they can also be deleted (as they are not needed in the baseline):

```
data/
deploy/
extras/
quickstarts/
```

    4. Using an archive tool, compress the **jboss-fuse-karaf-*ProductVersion*-baseline** directory in Zip format to give the following archive file:

```
jboss-fuse-karaf-ProductVersion-baseline.zip
```

       The preceding file is the baseline archive for the custom assembly.

12. Copy the baseline archive to the following subdirectory (must be created) of the original custom assembly, ***Working*/custom-distro-*ProductVersion*-bin**:

```
system/org/jboss/fuse/jboss-fuse-karaf/ProductVersion
```

13. The custom assembly, ***Working*/custom-distro-*ProductVersion*-bin**, is now complete. If you like, you can compress this directory into a Zip archive and distribute it as is.

14. Test the new custom assembly to make sure it is working properly. Launch the Karaf container, as follows:

```
cd Working/custom-distro-ProductVersion-bin/bin
./fuse
```

The container should start up without error. To make sure that no errors have occurred, check the log:

```
JBossFuse:karaf@root> log:display
```

You can also check to see whether all of the expected Karaf features are installed:

```
JBossFuse:karaf@root> features:list -i
```

When you are finished testing the custom assembly container, shut it down by entering the **shutdown** console command, and tidy up the custom assembly installation by removing the **data/** subdirectory (which holds the data from the current container instance).

TROUBLESHOOTING

There are a few common errors you might encounter when attempting to start the custom assembly container. Here is a brief explanation of what these errors mean:

- The following error indicates that the baseline archive was not correctly installed in the **system** directory:

```
ERROR: Bundle org.apache.felix.framework [0] EventDispatcher: Error
during dispatch. (io.fabric8.patch.management.PatchException: Can't
find baseline distribution in patches dir or inside system
repository.)
io.fabric8.patch.management.PatchException: Can't find baseline
distribution in patches dir or inside system repository.
        at
io.fabric8.patch.management.impl.GitPatchManagementServiceImpl.track
BaselineRepository(GitPatchManagementServiceImpl.java:1820)
        at
...
```

- The following error indicates that the **patch** feature is not deployed in the container:

```
Bundle listed in startup.properties configuration not found:
io/fabric8/patch/patch-management/1.2.0.redhat-621092/patch-
management-1.2.0.redhat-621092.jar
Could not create framework: java.lang.Exception: Aborting due to
missing startup bundles
java.lang.Exception: Aborting due to missing startup bundles
 at
org.apache.karaf.main.Main.processConfigurationProperties(Main.java:
1281)
 at org.apache.karaf.main.Main.loadStartupProperties(Main.java:1082)
 at org.apache.karaf.main.Main.launch(Main.java:346)
 at org.apache.karaf.main.Main.main(Main.java:561)
```

- The following error indicates that the **patch** feature was successfully deployed, but the required **fabric** feature is missing:

```
ERROR: Bundle io.fabric8.patch.patch-core [49] Error starting
mvn:io.fabric8.patch/patch-core/1.2.0.redhat-621092
(org.osgi.framework.BundleException: Unresolved constraint in bundle
io.fabric8.patch.patch-core [49]:
Unable to resolve 49.0: missing requirement [49.0]
osgi.wiring.package; (&(osgi.wiring.package=io.fabric8.api)
(version>=1.2.0)(!(version>=2.0.0))))
```