Red Hat Fuse 7.6

スタートガイド

Red Hat Fuse を今すぐ使用する

Last Updated: 2023-08-11

Red Hat Fuse 7.6 スタートガイド

Red Hat Fuse を今すぐ使用する

法律上の通知

概要

Fuse on Spring Boot、Fuse on Apache Karaf、および Fuse on JBoss Enterprise Application Platform を今すぐ使用します。

目次

前書き

Fuse を使用するには、Spring Boot、JBoss EAP、または Apache Karaf に関わらず、該当のコンテナーのファイルをダウンロードおよびインストールする必要があります。本書では、これらの各コンテナーに初めて Fuse アプリケーションをインストール、開発、および構築するための情報および手順を提供します。

- 1章*Fuse on Spring Boot の使用*
- 2章*Fuse on Karaf の使用*
- 3章*Fuse on JBoss EAP の使用*
- 4章*Maven のローカルでの設定*

第1章 FUSE ON SPRING BOOT の使用

Spring Boot で Fuse アプリケーションを開発するには、最初に Spring Boot で実行する Fuse のサンプルブースタープロジェクトを生成およびビルドします。詳細は以下のセクションを参照してください。

- 「Fuse on Spring Boot」
- 「ブースタープロジェクトの生成」
- 「ブースタープロジェクトのビルド」

1.1. FUSE ON SPRING BOOT

Spring Boot は、よく知られている Spring コンテナーがさらに進化したものです。Spring Boot コンテナーの特徴は、コンテナー機能が独立してデプロイできる小さなチャンクに分割されていることです。これにより、特定のサービスに特化し、小さなフットプリントでコンテナーをデプロイできます。これは、**マイクロサービスアーキテクチャー**のパラダイムに対応するために必要になります。

このコンテナー技術の特徴は次のとおりです。

- 特に、スケーラブルなクラウドプラットフォーム (Kubernetes および OpenShift) での実行に適しています。
- マイクロサービスアーキテクチャーに適した小さなフットプリント。
- **設定より規約** (Convention over Configuration) 向けに最適化されています。
- アプリケーションサーバーは必要ありません。Spring Boot アプリケーション Jar を直接 JVM で実行できます。

1.2. ブースタープロジェクトの生成

Fuse ブースタープロジェクトは、スタンドアロンアプリケーションの実行を手助けする開発者向けのプロジェクトです。ここでは、ブースタープロジェクトの 1 つである Circuit Breaker ブースターの生成手順を説明します。この演習では、Fuse on Spring Boot の便利なコンポーネントを使用します。

Netflix/Hystrix サーキットブレーカーを使用すると、ネットワーク接続の中断やバックエンドサービスの一時的な利用停止に分散アプリケーションが対処できるようになります。サーキットブレーカーパターンの基本概念は、バックエンドサービスが一時的に利用できなくなった場合に備えて、依存するサービスの損失を自動的に検出し、代替動作をプログラムで作成できるようにすることです。

Fuse サーキットブレーカーブースターは、次の 2 つの関連サービスで構成されます。

- 呼び名を返すバックエンドサービスである **name** サービス。
- 名前を取得する **name** サービスを呼び出し、文字列 **Hello, NAME** を返すフロントエンドサービスである **greetings** サービス。

このブースターデモンストレーションでは、Hystrix サーキットブレーカーは **greetings** サービスと **name** サービスとの間に挿入されます。バックエンドの **name** サービスが利用できなくなると、**name** サービスが再起動するまでの間に **greetings** サービスはブロックされず、代替動作にフォールバックして即座にクライアントに応答します。

前提条件

- Red Hat Developer Platform にアクセスできる。
- サポートされるバージョンの Java Developer Kit (JDK) を持っている。詳細は Red Hat Fuse でサポートされる設定 を参照してください。
- Apache Maven 3.3.x 以上が必要です。

手順

1. https://developers.redhat.com/launch に移動します。
2. **START** をクリックします。
   ランチャーウィザードによって、Red Hat アカウントにログインするよう要求されます。
3. **Log in or register**ボタンをクリックし、ログインします。
4. **Launcher** ページで **Deploy an Example Application**ボタンをクリックします。
5. **Create Example Application** ページで **Create Example Application as**フィールドに名前 **fuse-circuit-breaker** を入力します。
6. **Select an Example** をクリックします。
7. **Example** ダイアログで、**Circuit Breaker** オプションを選択します。**Select a Runtime** ドロップダウンメニューが表示されます。
   a. **Select a Runtime** ドロップダウンメニューで **Fuse** を選択します。
   b. バージョンのドロップダウンメニューで **7.6 (Red Hat Fuse)** を選択します。**2.21.2 (Community)** バージョンは選択しないでください。
   c. **Save** をクリックします。
8. **Create Example Application** ページで **Download** をクリックします。
9. **Your Application is Ready** ダイアログが表示されたら、**Download.zip** をクリックします。ブラウザーが生成されたブースタープロジェクト (ZIP ファイルとしてパッケージ) をダウンロードします。
10. アーカイブユーティリティーを使用して、生成されたプロジェクトをローカルファイルシステムの任意の場所に展開します。

1.3. ブースタープロジェクトのビルド

以下の手順では、Fuse on Spring Boot で Circuit Breaker ブースターをビルドする方法を説明します。

前提条件

- Red Hat Developer Portal からブースタープロジェクトの生成およびダウンロードが完了している。
- サポートされるバージョンの Java Developer Kit (JDK) を持っている。詳細は Red Hat Fuse でサポートされる設定 を参照してください。
- Apache Maven 3.3.x 以上が必要です。

手順

手順

1. シェルプロンプトを開き、Maven を使用してコマンドラインからプロジェクトをビルドします。

```
cd fuse-circuit-breaker
```

```
mvn clean package
```

Maven によってプロジェクトがビルドされると、**Build Success** というメッセージが表示されます。

2. 新しいシェルプロンプトを開き、以下のように name サービスを起動します。

```
cd name-service
```

```
mvn spring-boot:run -DskipTests -Dserver.port=8081
```

Spring Boot が起動すると、以下のような出力が表示されます。

```
...
2019-05-06 20:19:59.401  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Route: route1 started and consuming from: servlet:/name?httpMethodRestrict=GET
2019-05-06 20:19:59.402  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Total 1 routes, of which 1 are started
2019-05-06 20:19:59.403  INFO 9553 --- [           main] o.a.camel.spring.SpringCamelContext
: Apache Camel 2.21.0.fuse-730078-redhat-00001 (CamelContext: camel-1) started in 0.287
seconds
2019-05-06 20:19:59.406  INFO 9553 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-05-06 20:19:59.473  INFO 9553 --- [           main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8081 (http)
2019-05-06 20:19:59.479  INFO 9553 --- [           main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 5.485 seconds (JVM
running for 9.841)
```

3. 新しいシェルプロンプトを開き、以下のように greetings サービスを起動します。

```
cd greetings-service
```

```
mvn spring-boot:run -DskipTests
```

Spring Boot が起動すると、以下のような出力が表示されます。

```
...
2019-05-06 20:22:19.051  INFO 9729 --- [           main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-05-06 20:22:19.115  INFO 9729 --- [           main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8080 (http)
2019-05-06 20:22:19.123  INFO 9729 --- [           main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 7.68 seconds (JVM running
for 12.66)
```

greetings サービスは **http://localhost:8080/camel/greetings** URL で REST エンドポイントを公開します。

4. web ブラウザーで URL を開くか、別のシェルプロンプトで以下の **curl** コマンドを入力して、REST エンドポイントを呼び出します。

```
curl http://localhost:8080/camel/greetings
```

応答は次のとおりです。

```
{"greetings":"Hello, Jacopo"}
```

5. Camel Hystrix によって提供されるサーキットブレーカー機能を実証するために、name サービスが実行されているシェルプロンプトウインドウで **Ctrl-C** を入力し、バックエンド name サービスを強制終了します。
これで name サービスが利用できなくなるため、呼び出されたときに greetings サービスがハングしないよう、サーキットブレーカーが作動します。

6. web ブラウザーで **http://localhost:8080/camel/greetings** を開くか、別のシェルプロンプトウインドウに以下の **curl** コマンドを入力して、greetings REST エンドポイントを呼び出します。

```
curl http://localhost:8080/camel/greetings
```

応答は次のとおりです。

```
{"greetings":"Hello, default fallback"}
```

greetings サービスが実行されているウインドウで、ログに以下のメッセージシーケンスが表示されます。

```
2019-05-06 20:24:16.952  INFO 9729 --- [-CamelHystrix-2] route2                   : Try to call name Service
2019-05-06 20:24:16.956  INFO 9729 --- [-CamelHystrix-2] o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.956  INFO 9729 --- [-CamelHystrix-2] o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2] o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2] o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2] o.a.c.httpclient.HttpMethodDirector      : I/O exception (java.net.ConnectException) caught when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.957  INFO 9729 --- [-CamelHystrix-2] o.a.c.httpclient.HttpMethodDirector      : Retrying request
2019-05-06 20:24:16.964  INFO 9729 --- [-CamelHystrix-2] route2                   : We are falling back!!!!
```

7. この例に関する詳細は、**greetings-service** の実行中に http://localhost:8080/ で **Circuit Breaker - Red Hat Fuse** ページを開いてください。このページには、サーキットブレーカーの状態を監視する Hystrix ダッシュボードへのリンクが含まれます。

第2章 FUSE ON KARAF の使用

ここで説明する情報や手順は、Fuse On Karaf について学び、Karaf コンテナーで初めて Fuse アプリケーションをインストール、開発、および構築するのに役立ちます。詳細は以下のトピックを参照してください。

- 「Fuse on Karaf」
- 「Fuse on Karaf のインストール」
- 「Karaf で初めて Fuse アプリケーションを構築する」

2.1. FUSE ON KARAF

Apache Karaf は、OSGi Alliance の OSGi 標準 をベースとしています。OSGi は電気通信業界で開発され、サーバーをシャットダウンする必要なく、オンザフライでアップグレードできる (**ホットコードスワッピング** として知られる機能) ゲートウェイサーバーを開発するために使用されました。その後、OSGi コンテナー技術はその他多くの用途に使用されるようになり、一般的に Eclipse IDE などのモジュール化されたアプリケーションで使用されています。

このコンテナー技術の特徴は次のとおりです。

- 特にスタンドアロンモードでの実行に適しています。
- モジュール化 (OSGi バンドル) の強力なサポート、および高度なクラスローディングのサポート。
- 1つのコンテナーに複数のバージョンの依存関係を一緒にデプロイできます (ただし、これは注意して行う必要があります)。
- ホットコードスワッピングにより、コンテナーをシャットダウンせずにモジュールをアップグレードまたは置き換えできます。これは独自の機能ですが、適切に行うには多大な労力が必要になります。

注記: Spring Dynamic Modules (Spring-DM) (Spring XML と Apache Karaf の OSGi サービス層を統合) はサポート対象ではありません。代わりに Blueprint フレームワークを使用する必要があります。Blueprint XML を使用しても、Spring フレームワークから Java ライブラリーを使用することはできます。最新バージョンの Spring は Blueprint と互換性があります。

2.2. FUSE ON KARAF のインストール

Red Hat カスタマーポータルから Fuse 7.6 on Karaf の標準インストールパッケージをダウンロードできます。このパッケージは Karaf コンテナーの標準アセンブリーをインストールし、完全な Fuse テクノロジースタックを提供します。

前提条件

- Red Hat カスタマーポータル のフルサブスクリプションアカウントを持っている。
- カスタマーポータルにログインしている。
- CodeReady Studio インストーラー がダウンロードされている必要があります。
- Fuse on Karaf インストーラー がダウンロードされている必要があります。

手順

手順

1. ダウンロードした Fuse on Apache Karaf の **.zip** アーカイブファイルを、ファイルシステム **FUSE_INSTALL** の任意の場所に展開します。
2. Fuse ランタイムに管理ユーザーを追加します。
   a. テキストエディターで **FUSE_INSTALL/etc/users.properties** ファイルを開きます。
   b. **#admin = admin** で始まる行の最初の **#** 文字を削除します。
   c. **#_g_\:admingroup** で始まる行の最初の **#** 文字を削除します。
   d. ユーザーエントリーのユーザー名 **USERNAME** とパスワード **PASSWORD** をカスタマイズして、次のようなユーザーエントリーと管理グループエントリーを (連続した行に) 作成します。

```
USERNAME = PASSWORD,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
```

   e. **etc/users.properties** ファイルを保存します。
3. 以下のように、CodeReady Studio インストーラー を実行します。

```
java -jar DOWNLOAD_LOCATION/codereadystudio-12.14.0.GA-installer-standalone.jar
```

4. インストール中、以下を行います。
   a. 契約条件に同意します。
   b. インストールパスを選択します。
   c. Java 8 JVM を選択します。
   d. **Select Platforms and Servers** で、**Add** をクリックして **FUSE_INSTALL** ディレクトリーの場所を確認し、Fuse on Karaf ランタイムを設定します。
   e. **Select Additional Features to Install** で **Red Hat Fuse Tooling** を選択します。
5. CodeReady Studio が起動します。**Searching for runtimes** ダイアログが表示されたら **OK** をクリックして Fuse on Karaf ランタイムを作成します。
6. **(任意手順)**: コマンドラインから Apache Maven を使用するには、Maven をインストールおよび設定する必要があります。

注記

CodeReady Studio のみを使用する場合、CodeReady Studio には Maven が事前インストールおよび設定されているため、Maven をインストールする必要は厳密にはありません。しかし、コマンドラインから Maven を呼び出す場合は、インストールを行う必要があります。

2.3. KARAF で初めて FUSE アプリケーションを構築する

次の手順は、Karaf で初めて Fuse アプリケーションを構築する場合に便利です。

前提条件

- Red Hat カスタマーポータル のフルサブスクリプションアカウントを持っている。
- カスタマーポータルにログインしている。
- CodeReady Studio インストーラー がダウンロードされている必要があります。
- ダウンロードした Fuse on Karaf インストーラー が正常にインストールされている必要があります。

手順

1. CodeReady Studio で以下のように新しいプロジェクトを作成します。
   a. **File→New→Fuse Integration Project** と選択します。
   b. **Project Name** フィールドに **fuse-camel-cbr** を入力します。
   c. **Next** をクリックします。
   d. **Select a Target Environment** ペインで以下の設定を選択します。
      - **Standalone** をデプロイメントプラットフォームとして選択します。
      - **Karaf/Fuse on Karaf** をランタイム環境として選択し、**Runtime (optional)** ドロップダウンメニューを使用して **fuse-karaf-7.9.0.fuse-790071-redhat-00001 Runtime** サーバーをターゲットランタイムとして選択します。
   e. ターゲットランタイムの選択後、**Camel Version** が自動的に選択され、フィールドがグレーアウトされます。
   f. **Next** をクリックします。
   g. **Advanced Project Setup** ペインで **Beginner→Content Based Router - Blueprint DSL** テンプレートを選択します。
   h. **Finish** をクリックします。
   i. 関連する Fuse Integration パースペクティブを開くように要求された場合は、**Yes** をクリックします。
   j. CodeReady Studio が必要なアーティファクトをダウンロードし、バックグラウンドでプロジェクトをビルドする間待機します。

重要

CodeReady Studio で初めて Fuse プロジェクトをビルドする場合は、リモート Maven リポジトリーから依存関係をダウンロードするため、ウィザードがプロジェクトの生成を完了するまで **数分**かかることがあります。プロジェクトがバックグラウンドでビルドされている間は、ウィザードを中断したり、CodeReady Studio を閉じたりしないようにしてください。

2. 以下のように、プロジェクトをサーバーにデプロイします。
   a. サーバーが起動していない場合は、**Servers** ビュー (Fuse Integration パースペクティブの左下隅) で **fuse-karaf-7.6.0.fuse-760025-redhat-00001 Runtime Server** サーバーを選択

し、緑色の矢印をクリックして起動します。

注記

Warning: The authenticity of host 'localhost' can't be established.というダイアログが表示されたら、**Yes** をクリックしてサーバーに接続し、Karaf コンソールにアクセスします。

b. **Console** ビューに以下のようなメッセージが表示されるまで待機します。

```
Karaf started in 1s. Bundle stats: 12 active, 12 total
```

c. サーバーが起動した後、**Servers** ビューに切り替え、サーバーを右クリックしてコンテキストメニューで **Add and Remove** を選択します。

d. **Add and Remove** ダイアログで **fuse-camel-cbr** プロジェクトを選択し、**Add >** ボタンをクリックします。

e. **Finish** をクリックします。

f. **Terminal** ビューに移動し、**bundle:list | tail** を入力して、プロジェクトの OSGi バンドルが起動したかどうかをチェックします。以下のような出力が表示されるはずです。

```
...
228 │ Active │  80 │ 1.0.0.201505202023          │ org.osgi:org.osgi.service.j
232 │ Active │  80 │ 1.0.0.SNAPSHOT              │ Fuse CBR Quickstart
```

注記

Camel ルートが起動すると、即座に **work/cbr/input** ディレクトリーが Fuse インストールに作成されます (**fuse-camel-cbr** プロジェクトには作成**されません**)。

3. プロジェクトの **src/main/data** ディレクトリーにあるファイルを **FUSE_INSTALL/work/cbr/input** ディレクトリーにコピーします。これは、システムファイルブラウザー (Eclipse の外部) で実行できます。

4. しばらく待ってから、**FUSE_INSTALL/work/cbr/output** ディレクトリーをチェックし、同じファイルが国ごとに整理されていることを確認します。

   a. **work/cbr/output/others** の **order1.xml**

   b. **work/cbr/output/uk** の **order2.xml** および **order4.xml**

   c. **work/cbr/output/us** の **order3.xml** および **order5.xml**

5. 以下のようにプロジェクトをアンデプロイします。

   a. **Servers** ビューで **Red Hat Fuse 7+ Runtime Server** サーバーを選択します。

   b. サーバーを右クリックし、コンテキストメニューで **Add and Remove** を選択します。

   c. **Add and Remove** ダイアログで **fuse-camel-cbr** プロジェクトを選択し、**< Remove** ボタンをクリックします。

d. **Finish** をクリックします。

第3章 FUSE ON JBOSS EAP の使用

この章では、Fuse on JBoss EAP を紹介し、JBoss EAP コンテナーで初めて Fuse アプリケーションをインストール、開発、および構築する方法を説明します。

詳細は以下のトピックを参照してください。

- 「Fuse on JBoss EAP」
- 「Fuse on JBoss EAP のインストール」
- 「JBoss EAP で最初の Fuse アプリケーションをビルド」

3.1. FUSE ON JBOSS EAP

Eclipse Foundation の Jakarta EE の技術 (旧名称 Java EE) をベースとした JBoss Enterprise Application Platform (EAP) は、当初はエンタープライズアプリケーション開発のユースケースに対応するために作成されました。JBoss EAP は、サービスおよび標準化された Java API (永続性、メッセージング、セキュリティーなど) を実装する明確に定義されたパターンを特徴としています。近年ではディペンデンシーインジェクション (依存性の注入) の CDI の導入や、エンタープライズ Java Bean の簡素化されたアノテーションの導入により、この技術はより軽量化されました。

このコンテナー技術の特徴は次のとおりです。

- 特にスタンドアロンモードでの実行に適しています。
- 事前設定された多くの標準サービス (永続性、メッセージング、セキュリティーなど) をそのまま使用できます。
- 通常、アプリケーション WAR は小型および軽量です (多くの依存関係がコンテナーに事前インストールされているため)。
- 標準化された後方互換性のある Java API。

3.2. FUSE ON JBOSS EAP のインストール

Red Hat カスタマーポータルから Fuse 7.6 on JBoss EAP の標準インストールパッケージをダウンロードできます。このパッケージは JBoss EAP コンテナーの標準アセンブリーをインストールし、完全な Fuse テクノロジースタックを提供します。

前提条件

- Red Hat カスタマーポータル のフルサブスクリプションアカウントを持っている。
- カスタマーポータルにログインする必要があります。
- JBoss EAP および JBoss EAP 7.2 Update 05 がダウンロードされている必要があります。
- Fuse on JBoss EAP がダウンロードされている必要があります。
- CodeReady Studio インストーラー がダウンロードされている必要があります。

手順

1. 次のように、シェルプロンプトから JBoss EAP インストーラーを実行します。

```
java -jar DOWNLOAD_LOCATION/jboss-eap-7.2.0-installer.jar
```

2. インストール中、以下を行います。
   a. 契約条件に同意します。
   b. JBoss EAP ランタイムのインストールパス **EAP_INSTALL** を選択します。
   c. 管理ユーザーを作成し、後で必要になる管理ユーザーの認証情報を注意して書き留めておきます。
   d. その他の画面では、デフォルト設定をそのまま使用できます。
3. シェルプロンプトを開き、**EAP_INSTALL** ディレクトリーに移動します。
4. **EAP_INSTALL** ディレクトリーから JBoss EAP 7.2 Update 05 を適用します。以下に例を示します。

```
bin/jboss-cli.sh "patch apply jboss-eap-7.2.x-patch.zip"
```

5. **EAP_INSTALL** ディレクトリーから、以下のように Fuse on EAP のインストーラーを実行します。

```
java -jar DOWNLOAD_LOCATION/fuse-eap-installer-7.6.0.jar
```

6. 以下のように、CodeReady Studio インストーラーを実行します。

```
java -jar DOWNLOAD_LOCATION/codereadystudio-12.14.0.GA-installer-standalone.jar
```

7. インストール中、以下を行います。
   a. 契約条件に同意します。
   b. インストールパスを選択します。
   c. Java 8 JVM を選択します。
   d. **Select Platforms and Servers** ステップで、**Add** をクリックして **EAP_INSTALL** ディレクトリーの場所を確認し、JBoss EAP ランタイムを設定します。
   e. **Select Additional Features to Install** で **Red Hat Fuse Tooling** を選択します。
8. CodeReady Studio が起動します。**Searching for runtimes** ダイアログが表示されたら **OK** をクリックして JBoss EAP ランタイムを作成します。
9. **(任意手順)**: コマンドラインから Apache Maven を使用するには、Maven をインストールおよび設定する必要があります。

注記

CodeReady Studio のみを使用する場合、CodeReady Studio には Maven が事前インストールおよび設定されているため、厳密には Maven をインストールする必要はありません。しかし、コマンドラインから Maven を呼び出す場合は、インストールを行う必要があります。

3.3. JBOSS EAP で最初の FUSE アプリケーションをビルド

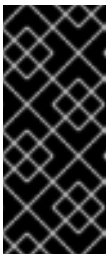
次の手順は、JBoss EAP で初めて Fuse アプリケーションを構築する場合に便利です。

前提条件

- Red Hat カスタマーポータル のフルサブスクリプションアカウントを持っている。
- カスタマーポータルにログインしている。
- ダウンロードした Fuse on JBoss EAP が正常にインストールされている必要があります。
- CodeReady Studio インストーラー がダウンロードされ、正しくインストールされている必要があります。

手順

1. CodeReady Studio で以下のように新しいプロジェクトを作成します。
   a. **File→New→Fuse Integration Project** と選択します。
   b. **Project Name** フィールドに **eap-camel** を入力します。
   c. **Next** をクリックします。
   d. **Select a Target Environment** ペインで以下の設定を選択します。
      - **Standalone** をデプロイメントプラットフォームとして選択します。
      - **Wildfly/Fuse on EAP** をランタイム環境として選択し、**Runtime (optional)** ドロップダウンメニューを使用して **JBoss EAP 7.x Runtime** サーバーをターゲットランタイムとして選択します。
   e. ターゲットランタイムの選択後、**Camel Version** が自動的に選択され、フィールドがグレーアウトされます。
   f. **Next** をクリックします。
   g. **Advanced Project Setup** ペインで **Spring Bean - Spring DSL** テンプレートを選択します。
   h. **Finish** をクリックします。



重要

CodeReady Studio で初めて Fuse プロジェクトをビルドする場合、ウィザードがプロジェクトの生成を完了するまで **数分** かかることがあります。これは、リモート Maven リポジトリーから依存関係をダウンロードするためです。プロジェクトがバックグラウンドでビルドされている間は、ウィザードを中断したり、CodeReady Studio を閉じたりしないでください。

   i. 関連する Fuse Integration パースペクティブを開くように要求された場合は、**Yes** をクリックします。
   j. CodeReady Studio が必要なアーティファクトをダウンロードし、バックグラウンドでプロジェクトをビルドする間待機します。

2. 以下のように、プロジェクトをサーバーにデプロイします。

   a. サーバーが起動していない場合は、**Servers** ビュー (Fuse Integration パースペクティブの右下隅) で **Red Hat JBoss EAP 7.2 Runtime** サーバーを選択し、緑色の矢印をクリックして起動します。

   b. **Console** ビューに以下のようなメッセージが表示されるまで待機します。

   > 14:47:07,283 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP 7.2.0.GA (WildFly Core 6.0.11.Final-redhat-00001) started in 13948ms - Started 495 of 680 services (326 services are lazy, passive or on-demand)

   c. サーバーが起動した後、**Servers** ビューに切り替え、サーバーを右クリックしてコンテキストメニューで **Add and Remove** を選択します。

   d. **Add and Remove** ダイアログで **eap-camel** プロジェクトを選択し、**Add >** をクリックします。

   e. **Finish** をクリックします。

3. 以下のように、プロジェクトが動作していることを確認します。

   a. URL http://localhost:8080/camel-test-spring?name=Kermit に移動し、**eap-camel** プロジェクトで実行されているサービスにアクセスします。

   b. ブラウザーウィンドウに **Hello Kermit** という応答が表示されるはずです。

4. 以下のようにプロジェクトをアンデプロイします。

   a. **Servers** ビューで **Red Hat JBoss EAP 7.2 Runtime** サーバーを選択します。

   b. サーバーを右クリックし、コンテキストメニューで **Add and Remove** を選択します。

   c. **Add and Remove** ダイアログで **eap-camel** プロジェクトを選択し、**< Remove** をクリックします。

   d. **Finish** をクリックします。

第4章 MAVEN のローカルでの設定

一般的な Fuse アプリケーションの開発では、Maven を使用してプロジェクトをビルドおよび管理します。

以下のトピックでは、Maven をローカルで設定する方法を説明します。

- 「Maven 設定の準備」
- 「Maven への Red Hat リポジトリーの追加」
- 「ローカル Maven リポジトリーの使用」
- 「環境変数またはシステムプロパティーを使用した Maven ミラーの設定」
- 「Maven アーティファクトおよびコーディネート」

4.1. MAVEN 設定の準備

Maven は、Apache の無料のオープンソースビルドツールです。通常は、Maven を使用して Fuse アプリケーションを構築します。

手順

1. Maven ダウンロードページ から最新バージョンの Maven をダウンロードします。
2. システムがインターネットに接続していることを確認します。
   デフォルトの動作では、プロジェクトのビルド中、Maven は外部リポジトリーを検索し、必要なアーティファクトをダウンロードします。Maven はインターネット上でアクセス可能なリポジトリーを探します。

   このデフォルト動作を変更し、Maven によってローカルネットワーク上のリポジトリーのみが検索されるようにすることができます。これは Maven をオフラインモードで実行できることを意味します。オフラインモードでは、Maven によってローカルリポジトリーのアーティファクトが検索されます。「ローカル Maven リポジトリーの使用」 を参照してください。

4.2. MAVEN への RED HAT リポジトリーの追加

Red Hat Maven リポジトリーにあるアーティファクトにアクセスするには、Red Hat Maven リポジトリーを Maven の **settings.xml** ファイルに追加する必要があります。Maven は、ユーザーのホームディレクトリーの **.m2** ディレクトリーで **settings.xml** ファイルを探します。ユーザー指定の **settings.xml** ファイルがない場合、Maven は **M2_HOME/conf/settings.xml** にあるシステムレベルの **settings.xml** ファイルを使用します。

前提条件

Red Hat リポジトリーを追加する **settings.xml** ファイルがある場所を把握している。

手順

以下の例のように、**settings.xml** ファイルに Red Hat リポジトリーの **repository** 要素を追加します。

```
<?xml version="1.0"?>
<settings>
```

```
<profiles>
  <profile>
    <id>extra-repos</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
     <repository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
      <repository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
      <repository>
        <id>jboss-public</id>
        <name>JBoss Public Repository Group</name>
        <url>https://repository.jboss.org/nexus/content/groups/public/</url>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
      <pluginRepository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
      <pluginRepository>
        <id>jboss-public</id>
```

```
        <name>JBoss Public Repository Group</name>
        <url>https://repository.jboss.org/nexus/content/groups/public</url>
      </pluginRepository>
    </pluginRepositories>
  </profile>
 </profiles>

 <activeProfiles>
  <activeProfile>extra-repos</activeProfile>
 </activeProfiles>

</settings>
```

4.3. ローカル MAVEN リポジトリーの使用

インターネットへ接続せずに Apache Karaf コンテナーを実行し、オフライン状態では使用できない依存関係を持つアプリケーションをデプロイする場合、Maven 依存関係プラグインを使用してアプリケーションの依存関係を Maven オフラインリポジトリーにダウンロードすることができます。ダウンロード後、このカスタマイズされた Maven オフラインリポジトリーをインターネットに接続していないマシンに提供することができます。

手順

1. **pom.xml** ファイルが含まれるプロジェクトディレクトリーで、以下のようなコマンドを実行し、Maven プロジェクトのリポジトリーをダウンロードします。

   ```
   mvn org.apache.maven.plugins:maven-dependency-plugin:3.1.0:go-offline -
   Dmaven.repo.local=/tmp/my-project
   ```

   この例では、プロジェクトのビルドに必要な Maven 依存関係とプラグインは **/tmp/my-project** ディレクトリーにダウンロードされます。

2. **etc/org.ops4j.pax.url.mvn.cfg** ファイルを編集し、**org.ops4j.pax.url.mvn.offline** を true に設定します。これによりオフラインモードが有効になります。

   ```
   ##
   # If set to true, no remote repository will be accessed when resolving artifacts
   #
   org.ops4j.pax.url.mvn.offline = true
   ```

3. このカスタマイズされた Maven オフラインリポジトリーを、インターネットに接続していない内部のマシンに提供します。

4.4. 環境変数またはシステムプロパティーを使用した MAVEN ミラーの設定

アプリケーションの実行時に、Red Hat Maven リポジトリーにあるアーティファクトにアクセスする必要があります。このリポジトリーは、Maven の **settings.xml** ファイルに追加されます。Maven は以下の場所で **settings.xml** を探します。

- 指定の URL を検索します。
- 見つからない場合は **${user.home}/.m2/settings.xml** を検索します。

- 見つからない場合は **${maven.home}/conf/settings.xml** を検索します。
- 見つからない場合は **${M2_HOME}/conf/settings.xml** を検索します。
- どの場所にも見つからない場合は、空の **org.apache.maven.settings.Settings** インスタンスが作成されます。

4.4.1. Maven ミラー

Maven では、一連のリモートリポジトリーを使用して、ローカルリポジトリーで現在利用できないアーティファクトにアクセスします。ほとんどの場合、リポジトリーのリストには Maven Central リポジトリーが含まれますが、Red Hat Fuse では Maven Red Hat リポジトリーも含まれます。リモートリポジトリーへのアクセスが不可能な場合や許可されない場合は、Maven ミラーのメカニズムを使用できます。ミラーは、特定のリポジトリー URL を異なるリポジトリー URL に置き換えるため、リモートアーティファクトの検索時にすべての HTTP トラフィックを単一の URL に転送できます。

4.4.2. Maven ミラーの **settings.xml** への追加

Maven ミラーを設定するには、以下のセクションを Maven の **settings.xml** に追加します。

```
<mirror>
    <id>all</id>
    <mirrorOf>*</mirrorOf>
    <url>http://host:port/path</url>
</mirror>
```

settings.xml ファイルに上記のセクションがない場合は、ミラーが使用されません。XML 設定を提供せずにグローバルミラーを指定するには、システムプロパティーまたは環境変数を使用します。

4.4.3. 環境変数またはシステムプロパティーを使用した Maven ミラーの設定

環境変数またはシステムプロパティーのいずれかを使用して Maven ミラーを設定するには、以下を追加します。

- 環境変数 **MAVEN_MIRROR_URL** を **bin/setenv** ファイルに追加します。
- システムプロパティー **mavenMirrorUrl** を **etc/system.properties** ファイルに追加します。

4.4.4. Maven オプションを使用した Maven ミラー URL の指定

環境変数またはシステムプロパティーによって指定された Maven ミラー URL ではなく、別の Maven ミラー URL を使用するには、アプリケーションの実行時に以下の Maven オプションを使用します。

- **-DmavenMirrorUrl=mirrorId::mirrorUrl**
  たとえば、**-DmavenMirrorUrl=my-mirror::http://mirror.net/repository** となります。
- **-DmavenMirrorUrl=mirrorUrl**
  たとえば、**-DmavenMirrorUrl=http://mirror.net/repository** となります。この例では、<mirror> の <id> は mirror になります。

4.5. MAVEN アーティファクトおよびコーディネート

Maven ビルドシステムでは、**アーティファクト** が基本的なビルディングブロックです。ビルド後のアーティファクトの出力は、通常 JAR や WAR ファイルなどのアーカイブになります。

Maven の主な特徴として、アーティファクトを検索し、検索したアーティファクト間で依存関係を管理できる機能が挙げられます。**Maven コーディネート** は、特定のアーティファクトの場所を特定する値のセットです。基本的なコーディネートには、以下の形式の 3 つの値があります。

groupId:artifactId:version

Maven は、**packaging** の値、または **packaging** 値と **classifier** 値の両方を使用して基本的なコーディネートを拡張することがあります。Maven コーディネートには以下の形式のいずれかを使用できます。

```
groupId:artifactId:version
groupId:artifactId:packaging:version
groupId:artifactId:packaging:classifier:version
```

値の説明は次のとおりです。

groupdId
: アーティファクトの名前の範囲を定義します。通常、パッケージ名のすべてまたは一部をグループ ID として使用します。たとえば、**org.fusesource.example** です。

artifactId
: グループ名に関連するアーティファクト名を定義します。

version
: アーティファクトのバージョンを指定します。バージョン番号には **n.n.n.n** のように最大 4 つの部分を使用でき、最後の部分には数字以外の文字を使用できます。たとえば **1.0-SNAPSHOT** の場合は、最後の部分が英数字のサブ文字列である **0-SNAPSHOT** になります。

packaging
: プロジェクトのビルド時に生成されるパッケージ化されたエンティティーを定義します。OSGi プロジェクトでは、パッケージングは **bundle** になります。デフォルト値は **jar** です。

classifier
: 同じ POM からビルドされた内容が異なるアーティファクトを区別できるようにします。

次に示すように、アーティファクトの POM ファイル内の要素で、アーティファクトのグループ ID、アーティファクト ID、パッケージング、およびバージョンを定義します。

```
<project ... >
  ...
  <groupId>org.fusesource.example</groupId>
  <artifactId>bundle-demo</artifactId>
  <packaging>bundle</packaging>
  <version>1.0-SNAPSHOT</version>
  ...
</project>
```

前述のアーティファクトの依存関係を定義するには、以下の **dependency** 要素を POM ファイルに追加します。

```
<project ... >
  ...
  <dependencies>
    <dependency>
      <groupId>org.fusesource.example</groupId>
      <artifactId>bundle-demo</artifactId>
      <version>1.0-SNAPSHOT</version>
```

```
    </dependency>
  </dependencies>
  ...
</project>
```

注記

前述の依存関係に **bundle** パッケージを指定する必要はありません。バンドルは特定タイプの JAR ファイルであり、**jar** はデフォルトの Maven パッケージタイプであるためです。依存関係でパッケージタイプを明示的に指定する必要がある場合は、**type** 要素を使用できます。