Red Hat Fuse 7.3

スタートガイド

Red Hat Fuse を今すぐ使用する

Last Updated: 2021-02-10

Red Hat Fuse 7.3 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

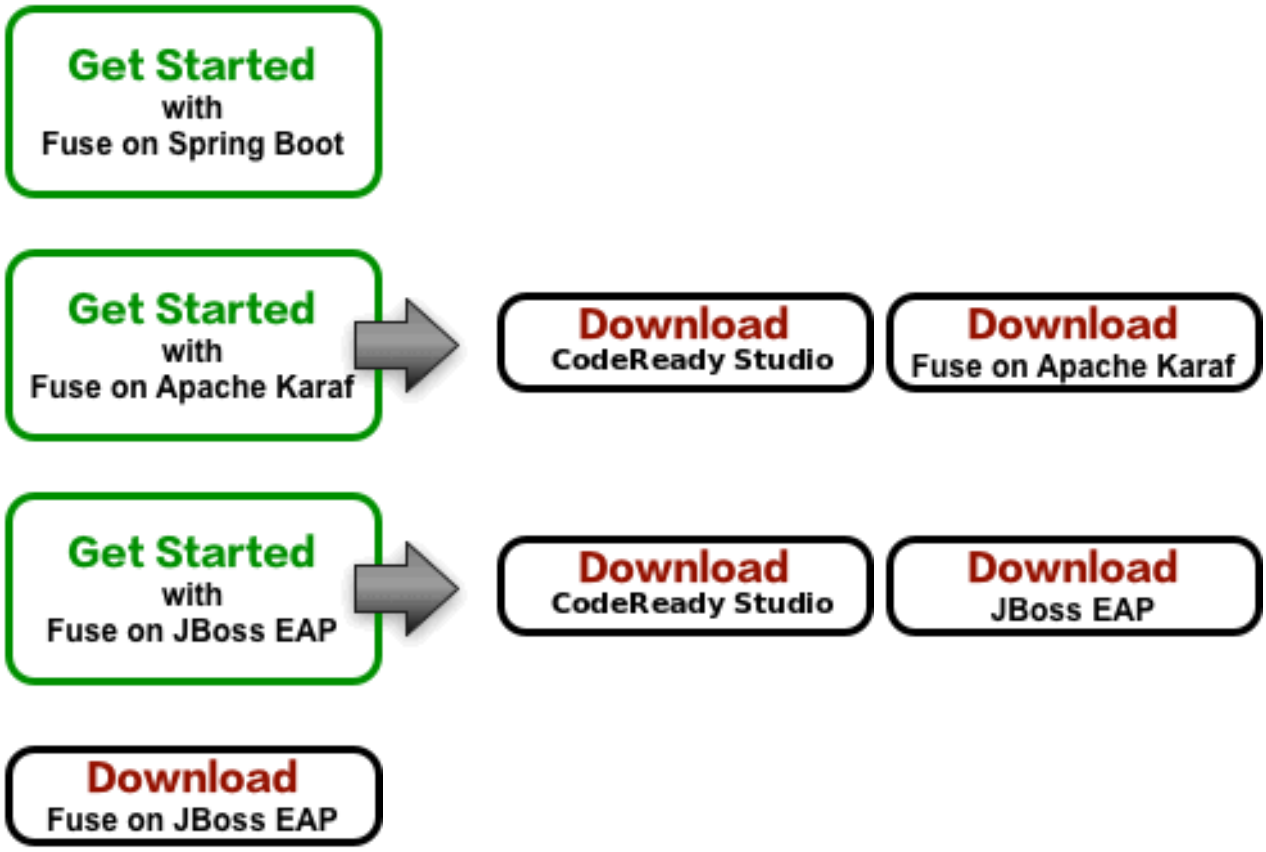
Fuse on Spring Boot、Fuse on Apache Karaf、Fuse on JBoss Enterprise Application Platform、および Fuse on JBoss Web Server を今すぐ使用します。

目次

第1章 スタートダッシュボード

1.1. FUSE スタンドアロンの代替

以下のダッシュボードは、Fuse スタンドアロンの代替の手順をコンテナータイプ別に表しています。



1.2. FUSE コンテナーの概要

プロジェクトに適したコンテナーの選択を容易にするため、以下のセクションでは各コンテナータイプの概要を説明します。

1.2.1. JBoss EAP

Eclipse Foundation の Jakarta EE (旧名称 Java EE) 技術をベースとした 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。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のサービスにアクセスするための) を実装する明確に定義されたパターンを特徴とし、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 (多くの依存関係は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。

1.2.2. Apache 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、**ホットコードスワッピング** と呼ばれる機能により、サーバーをシャットダウンする必要がなくすぐにアップグレードできる、ゲートウェイサーバーの開発に使用されました。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、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、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。

1.2.3. Spring Boot

Spring Boot はよく知られる Spring コンテナーが改良されたものです。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。

第2章 SPRING BOOT の使用

2.1. サーキットブレーカーブースターの概要

Netflix/Hystrix サーキットブレーカーは、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応できるようにします。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、依存するサービスの損失が自動的に検出され、代替動作をプログラムで作成できることです。

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

- 対応する名前を返すバックエンドサービスである **name サービス**。
- 名前を取得するために 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待機する間ブロックする代わりに、greetings サービスは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。

2.2. 前提条件

ブースターデモンストレーションをビルドおよび実行するには、以下をインストールします。

- サポートされるバージョンの Java Developer Kit (JDK)。詳細は「Red Hat Fuse でサポートされる構成」を参照してください。
- Apache Maven 3.3.x 以上。Maven の Download ページを参照してください。Maven の詳細は「付録A *Maven を使用する準備*」を参照してください。

2.3. ブースタープロジェクトの生成

サーキットブレーカーのブースタープロジェクトを生成するには、以下の手順を行います。

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 **7.3.0 (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。

2.4. ブースターのビルドおよび実行

ブースタープロジェクトをビルドおよび実行するには、以下の手順を実行します。

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た後、**Build Success** 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には、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 サービスを中止します。
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3章 APACHE KARAF の使用

3.1. カスタマーポータルへのログイン

必要なパッケージをダウンロードする前に、Red Hat Fuse サブスクリプションを持つ Red Hat カスタマーポータのアカウントが必要になります。このアカウントを使用して、https://access.redhat.com/login のポータルにログインします。

3.2. 必要なパッケージのダウンロード

各 **Download** ボタンをクリックして、カスタマーポータルから必要なパッケージを取得します。

3.3. FUSE ON APACHE KARAF のインストールおよび設定

Fuse on Apache Karaf をインストールおよび設定するには、以下のステップを実行します。

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、以下のようなユーザーエントリーおよび管理グループエントリーにします (連続した行)。

```
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。

3.4. 開発環境の設定

開発環境を設定するには、以下の手順を実行します。

1. 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/devstudio-12.11.0.GA-installer-standalone.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. 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 (詳細は 「Fuse on Apache Karaf のインストールおよび設定」 を参照してください)。

e. **Select Additional Features to Install** で **Red Hat Fuse Tooling** を選択します。

3. 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。

4. **(任意手順)**: コマンドラインから Apache Maven を使用するには、 付録A *Maven を使用する準備* の説明どおりに 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。

3.5. 最初のアプリケーションのビルド

以下の手順を実行して、Fuse on Karaf で最初のアプリケーションをビルドします。

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. **fuse-camel-cbr** 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。
- **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 **Red Had JBoss Middleware> Red Hat Fuse 7+ 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。

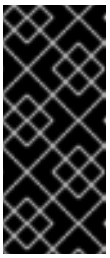
f. **Next** をクリックします。

g. **Advanced Project Setup** ペーンで **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、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-7.3.0.fuse-730079-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
```

3.5.1. プロジェクトの検証

Camel ルートが起動すると、すぐに **fuse-camel-cbr** プロジェクトにディレクトリー **work/cbr/input** が作成されます。

これで、Camel ルートをテストしてその動作を確認できるようになります。

1. **Project Explorer** ビューで **Refresh** をクリックし、新たに作成された **work/cbr/input** ディレクトリーを表示します。

2. 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 **work/cbr/input** ディレクトリーにコピーします。

3. しばらく待ってから再度 **Project Explorer** ビューを更新し、国別に分類された同じファイルが **work/cbr/output** ディレクトリーにあることを確認します。

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

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

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

3.5.2. プロジェクトのアンデプロイ

以下のようにプロジェクトをアンデプロイします。

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

2. 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。

3. **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。

4. **Finish** をクリックします。

第4章 FUSE ON JBOSS ENTERPRISE APPLICATION PLATFORM の使用

4.1. カスタマーポータルへのログイン

必要なパッケージをダウンロードする前に、Red Hat Fuse サブスクリプションを持つ Red Hat カスタマーポータのアカウントが必要になります。このアカウントを使用して、https://access.redhat.com/login のポータルにログインします。

4.2. 必要なパッケージのダウンロード

各 **Download** ボタンをクリックして、カスタマーポータルから必要なパッケージを取得します。

4.3. FUSE ON EAP のインストールおよび設定

Fuse on EAP をインストールおよび設定するには、以下のステップを実行します。

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クレデンシャル情報を注意して書き留めておきます。
   d. 残りの画面では、デフォルト設定を使用できます。
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、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-7.3.0.jar
```

4.4. 開発環境の設定

開発環境を設定するには、以下の手順を実行します。

1. 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/devstudio-12.11.0.GA-installer-standalone.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. 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 (「Fuse on EAP のインストールおよび設定」 を参照してください)。
   e. **Select Additional Features to Install** で **Red Hat Fuse Tooling** を選択します。
3. 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。
4. **(任意手順)**: コマンドラインから Apache Maven を使用するには、付録A *Maven を使用する準備* の説明どおりに 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。

4.5. 最初のアプリケーションのビルド

以下の手順を実行して、Fuse on JBoss EAP で最初のアプリケーションをビルドします。

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。

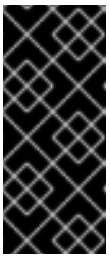
f. **Next** をクリックします。

g. **Advanced Project Setup** ペーンで **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。

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、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 **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 >** ボタンをクリックします。

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. ブラウザーウインドウには、**Hello Kermit** が応答として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** ボタンをクリックします。

d. **Finish** をクリックします。

付録A MAVEN を使用する準備

A.1. 概要

ここでは、Red Hat Fuse プロジェクトをビルドするために Maven を準備する方法の概要を説明し、Maven アーティファクトの検索に使用される Maven コーディネートの概念を紹介します。

A.2. 前提条件

Maven を使用してプロジェクトをビルドするには、以下が必要になります。

- **Maven インストール**: Maven は Apache の無料のオープンソースビルドツールです。最新バージョンは Maven のダウンロードページ からダウンロードできます。
- **ネットワーク接続**: ビルドの実行中、Maven は追加設定を必要とせずに動的に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。デフォルトでは、Maven はインターネット上でアクセスされたリポジトリーを探します。Maven がローカルネットワーク上のリポジトリーを優先して検索するように、この挙動を変更することができます。

注記

Maven はオフラインモードで実行できます。オフラインモードでは、Maven はローカルリポジトリーのアーティファクトのみを検索します。

A.3. RED HAT MAVEN リポジトリーの追加

Red Hat Maven リポジトリーから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** のシステムレベルの **settings.xml** ファイルを使用します。

Red Hat リポジトリーを Maven のリポジトリーリストに追加するには、新しい **.m2/settings.xml** ファイルを追加するか、システムレベルの設定を変更します。Red Hat Fuse リポジトリーの Maven への追加 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。

Red Hat Fuse リポジトリーの Maven への追加

```
<?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>
```

A.4. アーティファクト

アーティファクトは Maven ビルドシステムの基本的な要素です。Maven ビルドの実行後、アーティファクトの出力は通常、JAR や WAR などのアーカイブになります。

A.5. MAVEN コーディネート

Maven の主な機能には、アーティファクトを見つけ、アーティファクトとの間の依存関係を管理する機能があります。Maven は、特定のアーティファクトの場所を一意に定義する **Maven コーディネート** を使用して、アーティファクトの場所を定義します。基本的なコーディネートタプルの形式は **{groupId, artifactId, version}** です。追加のコーディネートである **packaging** および **classifier** をタプルに使用することもあります。タプルは、基本のコーディネート、追加の **packaging** コーディネート、または追加の **packaging** および **classifier** コーディネートの両方使用して、以下のように作成できます。

> **groupId**:**artifactId**:**version**
> **groupId**:**artifactId**:**packaging**:**version**
> **groupId**:**artifactId**:**packaging**:**classifier**:**version**

各コーディネートの説明は次のとおりです。

groupId

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。例: **org.fusesource.example**

artifactId

アーティファクト名 (グループ ID に関連する) 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 つのパートを含めることができ、最後のパートには数字以外の文字を含めることができます (たとえば **1.0-SNAPSHOT** の場合、最後のパートは英数字のサブ文字列 **0-SNAPSHOT** です)。

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。

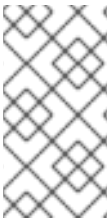
グループ ID、アーティファクト ID、パッケージング、およびバージョンは、アーティファクトの POM ファイルの対応する要素によって定義されます。以下に例を示します。

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

たとえば、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 に追加することができます。

```
<project ... >
```

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



注記

バンドルは特定タイプの JAR ファイルで、**jar** はデフォルトの Maven パッケージタイプであるため、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**。ただし、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。