



# Red Hat Process Automation Manager 7.3

Red Hat Process Automation Manager プロジェ  
クトのパッケージ化およびデプロイ



# Red Hat Process Automation Manager 7.3 Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ

---

Red Hat Customer Content Services  
brms-docs@redhat.com

## 法律上の通知

Copyright © 2019 Red Hat, Inc.

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

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

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

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

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

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

The OpenStack<sup>®</sup> 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.

## 概要

本書は、Red Hat Process Automation Manager 7.3 でプロジェクトをパッケージ化し、デプロイする方法を説明します。

## 目次

前書き .....	3
第1章 RED HAT PROCESS AUTOMATION MANAGER プロジェクトパッケージ .....	4
第2章 BUSINESS CENTRAL でのプロジェクトデプロイメント .....	5
2.1. BUSINESS CENTRAL に接続するように PROCESS SERVER の設定	5
2.2. PROCESS SERVER および BUSINESS CENTRAL での環境モードの設定	6
2.3. BUSINESS CENTRAL および PROCESS SERVER への外部 MAVEN リポジトリの設定	7
2.4. BUSINESS CENTRAL へのプロジェクトのビルドおよびデプロイメント	7
2.5. BUSINESS CENTRAL のデプロイメントユニット	9
2.5.1. Business Central でのデプロイメントユニットの作成	9
2.5.2. Business Central のデプロイメントユニットの起動、停止、および削除	9
2.6. BUSINESS CENTRAL プロジェクトの GAV 値の編集	10
2.7. BUSINESS CENTRAL における重複した GAV の検出	10
2.7.1. Business Central における重複した GAV 検出設定の管理	11
第3章 BUSINESS CENTRAL を使用しないプロジェクトデプロイメント .....	13
3.1. KIE モジュール記述子ファイルの設定	13
3.1.1. KIE モジュール設定のプロパティ	15
3.1.2. KIE モジュールでサポートされる KIE ベース属性	17
3.1.3. KIE モジュールでサポートされる KIE セッション属性	18
3.2. RED HAT PROCESS AUTOMATION MANAGER プロジェクトの MAVEN でのパッケージ化およびデプロイ	20
3.3. RED HAT PROCESS AUTOMATION MANAGER プロジェクトの JAVA アプリケーションでのパッケージ化およびデプロイ	24
3.4. PROCESS SERVER のサービスの起動	27
3.5. PROCESS SERVER のサービスの停止および削除	28
第4章 関連資料 .....	29
付録A バージョン情報 .....	30



## 前書き

ビジネスルール開発者は、Red Hat Process Automation Manager に作成したサービスの使用を開始するために、開発した Red Hat Process Automation Manager プロジェクトを Process Server にビルドしてデプロイする必要があります。プロジェクトの開発およびデプロイメントには、Business Central、独立した Maven プロジェクト、Java アプリケーション、もしくは複数のプラットフォームを組み合わせで使用できます。たとえば、Business Central のプロジェクトを開発して、Process Server REST API を使用してデプロイしたり、Business Central で設定した Maven にプロジェクトを開発して、Business Central を使用してデプロイしたりできます。

### 前提条件/事前作業

デプロイするプロジェクトを開発してテストしている。Business Central プロジェクトの場合は、テストシナリオを使用してプロジェクトのアセットをテストすることを検討してください。『[テストシナリオを使用したデシジョンサービスのテスト](#)』などを参照してください。

## 第1章 RED HAT PROCESS AUTOMATION MANAGER プロジェクトパッケージ

Red Hat Process Automation Manager プロジェクトには、Red Hat Process Automation Manager で開発するビジネスアセットが含まれます。Red Hat Process Automation Manager の各プロジェクトは、ナレッジ JAR (KJAR) ファイルとしてパッケージングされますが、その際にはプロジェクトのビルド、環境などの情報が含まれる Maven プロジェクトオブジェクトモデルファイル (**pom.xml**)、プロジェクトのアセットに対する KIE ベースおよび KIE セッションの設定が含まれる KIE モジュール記述子ファイル (**kmodule.xml**) などの設定ファイルが使用されます。KJAR ファイルから、パッケージ化した KJAR ファイルを、デシジョンサービス、プロセスアプリケーション、その他のデプロイ可能なアセット (総称して **サービス**) を実行する Process Server にデプロイします。このようなサービスは、ランタイム時に、インスタンス化した KIE コンテナ、または **デプロイメントユニット** を介して使用されます。プロジェクトの KJAR ファイルは Maven リポジトリに保存され、**GroupId**、**ArtifactId**、および **Version** (GAV) の 3 つの値で識別されます。この **Version** 値は、デプロイする可能性がある新しい全バージョンに対して一意である必要があります。(KJAR ファイルを含む) アーティファクトを識別するには、3 つの GAV 値がすべて必要になります。

Business Central のプロジェクトは、プロジェクトをビルドしてデプロイする際に自動的にパッケージ化されます。Business Central 以外のプロジェクト (独立した Maven プロジェクト、Java アプリケーションのプロジェクトなど) をビルドしてデプロイする場合は、追加した **kmodule.xml** ファイルに KIE モジュール記述子設定を追加するか、Java アプリケーションに直接指定する必要があります。

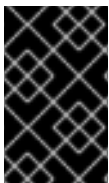


## 第2章 BUSINESS CENTRAL でのプロジェクトデプロイメント

Business Central を使用してビジネスアセットおよびサービスを開発し、プロジェクトデプロイメントに設定した Process Server を管理できます。プロジェクトを開発する際に、Business Central にプロジェクトをビルドして、Process Server に自動的にデプロイできます。自動デプロイメントを有効にするために、Business Central には Maven リポジトリが組み込まれています。Business Central から、ビルドしてデプロイしておいたサービスおよびプロジェクトバージョンを含むデプロイメントユニット (KIE コンテナ) を起動、停止、または削除できます。

(Menu → Deploy → Execution Servers で) 複数の Process Server を同じ Business Central インスタンスに接続して、複数のサーバー設定にグループ分けすることもできます。同じサーバー設定に属するサーバーは同じサービスを実行しますが、別のサーバー設定の別のプロジェクト、または別のバージョンのプロジェクトをデプロイすることもできます。

たとえば、**Test** 設定のテストサーバーと、**Production** 設定の実稼働サーバーを使用できます。ビジネスアセットとサービスをプロジェクトに開発し、**Test** サーバー設定にプロジェクトをデプロイしてから、十分にテストしたプロジェクトのバージョンを **Production** サーバー設定にデプロイできます。このとき、プロジェクトの開発を継続するには、プロジェクト設定でバージョンを変更します。これにより、組み込み Maven リポジトリで、新しいバージョンと古いバージョンが別のアーティファクトと見なされるため、**Test** サーバー設定に新しいバージョンをデプロイし、**Production** サーバー設定で古いバージョンを実行し続けることができます。このデプロイメントプロセスは単純ですが、重要な制約があります。とりわけ、アクセス制御は十分ではなく、プロジェクトを実稼働環境に直接デプロイできてしまいます。



### 重要

Business Central を使用して、Process Server を別のサーバー設定に移動することはできません。サーバーの設定名を変更するには、サーバーの設定ファイルを変更する必要があります。

## 2.1. BUSINESS CENTRAL に接続するように PROCESS SERVER の設定

Process Server を Red Hat Process Automation Manager 環境に設定していない場合、または Red Hat Process Automation Manager 環境に Process Server を追加する必要がある場合は、Process Server を設定して Business Central に接続する必要があります。



### 注記

Red Hat OpenShift Container Platform に Process Server をデプロイする場合は、[『Red Hat OpenShift Container Platform への Red Hat Process Automation Manager 管理サーバー環境のデプロイメント』](#)で、Business Central に接続する設定手順を参照してください。

### 前提条件/事前作業

Process Server がインストールされている。インストールオプションは [『Red Hat Process Automation Manager インストールの計画』](#) を参照してください。

### 手順

1. Red Hat Process Automation Manager インストールディレクトリーで、**standalone-full.xml** ファイルに移動します。たとえば、Red Hat Process Automation Manager に Red Hat JBoss EAP インストールを使用する場合は **\$EAP\_HOME/standalone/configuration/standalone-full.xml** に移動します。

2. **standalone-full.xml** を開き、**<system-properties>** タグの下に、以下のプロパティを設定します。
  - **org.kie.server.controller.user:** Business Central にログインするユーザーのユーザー名。
  - **org.kie.server.controller.pwd:** Business Central にログインするユーザーのパスワード。
  - **org.kie.server.controller:** Business Central の API に接続する URL。通常、URL は **http://<centralhost>:<centralport>/business-central/rest/controller** です。**<centralhost>** と **<centralport>** はそれぞれ Business Central のホスト名とポートになります。Business Central を OpenShift にデプロイしている場合は、URL から **business-central/** を削除します。
  - **org.kie.server.location:** Process Server の API に接続する URL。通常、URL は **http://<serverhost>:<serverport>/kie-server/services/rest/server** (**<serverhost>** および **<serverport>** はそれぞれ Process Server のホスト名およびポート) になります。
  - **org.kie.server.id:** サーバー設定の名前。このサーバー設定が Business Central に存在しない場合は、Process Server が Business Central に接続する時に自動的に作成されます。

例:

```
<property name="org.kie.server.controller.user" value="central_user"/>
<property name="org.kie.server.controller.password" value="central_password"/>
<property name="org.kie.server.controller" value="http://central.example.com:8080/business-central/rest/controller"/>
<property name="org.kie.server.location" value="http://kieserver.example.com:8080/kie-server/services/rest/server"/>
<property name="org.kie.server.id" value="production-servers"/>
```

3. Process Server を起動または再起動します。

## 2.2. PROCESS SERVER および BUSINESS CENTRAL での環境モードの設定

Process Server は、**production** (実稼働) モードと **development** (開発) モードでの実行が設定可能です。開発モードでは、柔軟な開発ポリシーが提供され、小規模な変更の場合はアクティブなプロセスインスタンスを維持しながら、既存のデプロイメントユニット (KIE コンテナ) を更新できます。また、大規模な変更の場合には、アクティブなプロセスインスタンスを更新する前に、デプロイメントユニットの状態をリセットすることも可能です。実稼働モードは、各デプロイメントで新規デプロイメントユニットが作成される実稼働環境に最適です。

開発環境では、**Deploy** をクリックすると、(該当する場合) 実行中のインスタンスを中止することなくビルドした KJAR ファイルを Business Server にデプロイすることができます。または **Redeploy** をクリックして、ビルドした KJAR ファイルをデプロイして実行中のインスタンスを中止することもできます。ビルドした KJAR ファイルを次回にデプロイまたは再デプロイすると、以前のデプロイメントユニット (KIE コンテナ) が同じターゲット Process Server で自動的に更新されます。

実稼働環境では、Business Central の **Redeploy** オプションは無効になり、**Deploy** をクリックして、ビルドした KJAR ファイルを Process Server 上の新規デプロイメントユニット (KIE コンテナ) にデプロイすることのみが可能です。

### 手順

1. Process Server 環境モードを設定するには、**org.kie.server.mode** システムプロパティを **org.kie.server.mode=development** または **org.kie.server.mode=production** に設定します。
2. Business Central のプロジェクトにデプロイメントの動作を設定するには、プロジェクトの **Settings** → **General Settings** → **Version** に移動して、**Development Mode** オプションを切り替えます。



#### 注記

デフォルトでは、Process Server と Business Central の新規プロジェクトはすべて、開発モードになっています。

**Development Mode** がオンのプロジェクトや、実稼働モードの Process Server に手動で **SNAPSHOT** バージョンの接尾辞を追加したプロジェクトをデプロイすることはできません。

## 2.3. BUSINESS CENTRAL および PROCESS SERVER への外部 MAVEN リポジトリの設定

組み込みリポジトリの代わりに、Nexus または Artifactory などの外部 Maven リポジトリを使用するように Business Central および Process Server を設定できます。この場合、Business Central でプロジェクトをビルドする時に、ビルドしたプロジェクトの全 KJAR ファイルがこの外部リポジトリにプッシュされます。統合プロセスを実装する場合に、必要に応じてリポジトリからこのファイルを処理し、Business Central または Process Server REST API を使用して KJAR ファイルをデプロイできます。

### 前提条件/事前作業

Business Central および Process Server がインストールされている。インストールオプションは『[Red Hat Process Automation Manager インストールの計画](#)』を参照してください。

### 手順

1. 外部リポジトリの接続およびアクセスの詳細が含まれる Maven **settings.xml** ファイルを作成します。**settings.xml** ファイルの詳細は Maven の『[Settings Reference](#)』を参照してください。
2. 既知の場所 (例: **/opt/custom-config/settings.xml**) にファイルを保存します。
3. Red Hat Process Automation Manager インストールディレクトリーで、**standalone-full.xml** ファイルに移動します。たとえば、Red Hat Process Automation Manager に Red Hat JBoss EAP インストールを使用する場合は **\$EAP\_HOME/standalone/configuration/standalone-full.xml** に移動します。
4. **standalone-full.xml** の **<system-properties>** タグで、**kie.maven.settings.custom** プロパティに **settings.xml** ファイルのフルパス名を設定します。  
以下に例を示します。

```
<property name="kie.maven.settings.custom" value="/opt/custom-config/settings.xml"/>
```

5. Business Central および Process Server を起動または再起動します。

## 2.4. BUSINESS CENTRAL へのプロジェクトのビルドおよびデプロイメント

プロジェクトを作成したら、Business Central でプロジェクトをビルドして、設定した Process Server にデプロイできます。プロジェクトをビルドしてデプロイする際に、Business Central のプロジェクトは、必要なすべてのコンポーネントとともに KJAR として自動的にパッケージ化されます。

## 手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. 右上隅で **Deploy** をクリックしてプロジェクトをビルドし、Process Server にデプロイします。Process Server にデプロイせずにプロジェクトをコンパイルするには、**Build** をクリックします。

## 注記

**Build & Install** オプションを選択してプロジェクトをビルドし、KJAR ファイルを Process Server にデプロイせずに設定済みの Maven リポジトリに公開することもできます。開発環境では、**Deploy** をクリックすると、(該当する場合) 実行中のインスタンスを中止することなくビルドされた KJAR ファイルを Process Server にデプロイすることができます。または **Redeploy** をクリックして、ビルドされた KJAR ファイルをデプロイして実行中のインスタンスを中止することもできます。ビルドされた KJAR ファイルを次回にデプロイまたは再デプロイすると、以前のデプロイメントユニット (KIE コンテナ) が同じターゲット Process Server で自動的に更新されます。実稼働環境では **Redeploy** オプションは無効になっており、**Deploy** をクリックして KJAR ファイルを Process Server 上の新規デプロイメントユニット (KIE コンテナ) にデプロイすることのみが可能です。

Process Server の環境モードを設定するには、**org.kie.server.mode** システムプロパティを **org.kie.server.mode=development** または **org.kie.server.mode=production** に設定します。Business Central の対応するプロジェクトでのデプロイメント動作を設定するには、プロジェクトの **Settings → General Settings → Version** に移動し、**Development Mode** オプションを選択します。デフォルトでは、Process Server および Business Central のすべての新規プロジェクトは開発モードになっています。**Development Mode** をオンにしたプロジェクトをデプロイしたり、実稼働モードになっている Process Server に手動で **SNAPSHOT** バージョンの接尾辞を追加したプロジェクトをデプロイすることはできません。

Business Central に Process Server を 1 つだけ接続する場合、または接続したすべての Process Server が同じサーバー設定にある場合は、デプロイメントユニット (KIE コンテナ) にあるプロジェクトのサービスが自動的に起動します。

複数のサーバー設定が利用できる場合は、サーバーとデプロイメントの詳細の入力を求めるデプロイメントダイアログが Business Central に表示されます。

3. デプロイメントダイアログが表示されたら、以下の値を確認または設定します。
  - **Deployment Unit Id / Deployment Unit Alias:** Process Server でサービスを実行しているデプロイメントユニット (KIE コンテナ) の名前およびエイリアスを確認します。通常は、この設定を変更する必要はありません。
  - **Server Configuration:** このプロジェクトをデプロイするサーバー設定を選択します。後で、プロジェクトを再ビルドしなくても、設定したその他のサーバーにデプロイできます。

- **Start Deployment Unit?**: このボックスを選択してデプロイメントユニット (KIE コンテナ) を起動するか、選択を解除して、サービスがサーバーにデプロイしても起動しないようにします。

## 2.5. BUSINESS CENTRAL のデプロイメントユニット

プロジェクトのサービスが、設定した Process Server のインスタンス化された KIE コンテナ、またはデプロイメントユニット を介してランタイム時に使用されます。Business Central にプロジェクトをビルドおよびデプロイすると、設定されたサーバーにデプロイメントユニットが自動的に作成されます。必要に応じて、Business Central にデプロイメントユニットを起動、停止、または削除できます。ビルドされているプロジェクトから追加デプロイメントユニットを作成したり、Business Central に設定した既存または新しい Process Server のデプロイメントユニットを起動したりすることもできます。

### 2.5.1. Business Central でのデプロイメントユニットの作成

お使いの Red Hat Process Automation Manager にはすでにデプロイメントユニットが1つ以上あるはずですが、ない場合は、Business Central にビルドされているプロジェクトからデプロイメントユニットを作成できます。

#### 前提条件/事前作業

新しいデプロイメントユニットを作成するプロジェクトが Business Central にビルドされている。

#### 手順

1. Business Central で **Menu → Deploy → Execution servers** に移動します。
2. **Server Configurations** の下で既存の設定を選択するか、**New Server Configuration** をクリックして設定を作成します。
3. **Deployment Units** の下で **Add Deployment Unit** をクリックします。
4. ウィンドウのテーブルで GAV を選択し、GAV の横にある **Select** を選択して、デプロイメントユニットのデータフィールドを追加します。
5. **Start Deployment Unit?** ボックスを選択してサービスを直ちに起動するか、選択を解除して後で起動します。
6. **Finish** をクリックします。  
サービスに新しいデプロイメントユニットが作成され、このサーバー設定で指定した Process Server に置かれました。**Start Deployment Unit?** を選択した場合は、サービスが起動します。

### 2.5.2. Business Central のデプロイメントユニットの起動、停止、および削除

デプロイメントユニットを起動したら、デプロイメントユニットのサービスが利用できるようになります。Business Central に Process Server を1つだけ接続する場合、または接続したすべての Process Server が同じサーバー設定にある場合は、デプロイメントユニットでサービスが自動的に起動します。複数のサーバー設定が利用可能な場合は、デプロイメント時に、サーバーとデプロイメントの詳細を指定して、デプロイメントユニットを起動するように求められます。ただし、必要に応じていつでも手動で Business Central でデプロイメントユニットを起動、停止、または削除して、デプロイしたサービスを管理できます。

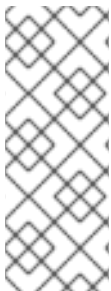
#### 手順

1. Business Central で **Menu → Deploy → Execution servers** に移動します。

2. **Server Configurations** の下で、設定を選択します。
3. **Deployment Units** の下で、デプロイメントユニットを選択します。
4. 右上の **Start**、**Stop**、または **Remove** をクリックします。実行中のデプロイメントユニットを削除する場合は、停止してから削除します。

## 2.6. BUSINESS CENTRAL プロジェクトの GAV 値の編集

**GroupId**、**ArtifactId**、および **Version** (GAV) 値は、Maven リポジトリのプロジェクトを識別します。Business Central と Process Server が同じファイルシステムにあり、同じ Maven リポジトリを使用する場合は、新しいバージョンのプロジェクトをビルドするたびに、リポジトリでプロジェクトが自動的に更新されます。ただし、Business Central と Process Server が別のファイルシステムにあり、ローカルの Maven リポジトリをそれぞれ使用している場合は、新しいバージョンのプロジェクトでプロジェクトの GAV 値 (通常はバージョン) を更新し、古いバージョンと新しいバージョンのプロジェクトが別のアーティファクトとして表示されるようにします。



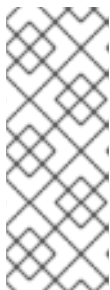
### 注記

開発目的でのみ、プロジェクトの **Settings** → **General Settings** → **Version** で **Development Mode** オプションを切り替えて、プロジェクトバージョンに **SNAPSHOT** のサフィックスを追加できます。このサフィックスで、Maven に対して、Maven ポリシーに従い、新しいスナップショットの更新を取得するように指示します。**開発モード** を使用したり、手動で実稼働環境に **SNAPSHOT** バージョンのサフィックスを追加しないでください。

プロジェクトの **Settings** 画面に GAV 値を設定できます。

### 手順

1. Business Central で、**Menu** → **Design** → **Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Settings** タブをクリックします。
3. 必要に応じて、**General Settings** の **Group ID** フィールド、**Artifact ID** フィールド、**Version** フィールドを修正します。プロジェクトをデプロイし、新しいバージョンを開発中の場合は、通常はバージョン番号を変更する必要があります。



### 注記

開発目的でのみ、プロジェクトの **Settings** → **General Settings** → **Version** で **Development Mode** オプションを切り替えて、プロジェクトバージョンに **SNAPSHOT** のサフィックスを追加できます。このサフィックスで、Maven に対して、Maven ポリシーに従い、新しいスナップショットの更新を取得するように指示します。**開発モード** を使用したり、手動で実稼働環境に **SNAPSHOT** バージョンのサフィックスを追加しないでください。

4. **Save** をクリックして終了します。

## 2.7. BUSINESS CENTRAL における重複した GAV の検出

Business Central のすべての Maven リポジトリで、**GroupId**、**ArtifactId**、**Version** (GAV) の各値が重複しているかどうかを確認されます。GAV が重複していると、実行された操作が取り消されます。



### 注記

重複する GAV の検出は、**Development Mode** のプロジェクトでは無効になっていません。Business Central で重複する GAV 検出を有効にするには、プロジェクトの **Settings → General Settings → Version** に移動して、**Development Mode** オプションを **OFF** (該当する場合) に切り替えます。

重複した GAV の検出は、以下の操作を実行するたびに実行されます。

- プロジェクトのプロジェクト定義の保存。
- **pom.xml** ファイルの保存。
- プロジェクトのインストール、ビルド、またはデプロイメント。

以下の Maven リポジトリで重複の GAV が確認されます。

- **pom.xml** ファイルの **<repositories>** 要素および **<distributionManagement>** 要素で指定されたりポジトリ。
- Maven の **settings.xml** 設定ファイルに指定されたりポジトリ。

## 2.7.1. Business Central における重複した GAV 検出設定の管理

**admin** ロールを持つ Business Central ユーザーは、プロジェクトで **GroupId** 値、**ArtifactId** 値、**Version** 値 (GAV) が重複しているかどうかを確認するリポジトリの一覧を修正できます。



### 注記

重複する GAV の検出は、**Development Mode** のプロジェクトでは無効になっていません。Business Central で重複する GAV 検出を有効にするには、プロジェクトの **Settings → General Settings → Version** に移動して、**Development Mode** オプションを **OFF** (該当する場合) に切り替えます。

### 手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Settings** タブをクリックし、**Validation** をクリックしてリポジトリの一覧を開きます。
3. 一覧表示したリポジトリオプションの中から選択するか選択を解除して、重複した GAV の検出を有効または無効にします。  
今後、重複した GAV の報告は、検証を有効にしたリポジトリに対してのみ行われます。



## 注記

この機能を無効にするには、システムの起動時に Business Central の **org.guvnor.project.gav.check.disabled** システムプロパティを **true** に設定します。

```
$ ~/EAP_HOME/bin/standalone.sh -c standalone-full.xml  
-Dorg.guvnor.project.gav.check.disabled=true
```



## 第3章 BUSINESS CENTRAL を使用しないプロジェクトデプロイメント

Business Central インターフェースにプロジェクトを開発およびデプロイする代わりに、独立した Maven プロジェクトまたは独自の Java アプリケーションを使用して、Red Hat Process Automation Manager プロジェクトを開発し、KIE コンテナ (デプロイメントユニット) のプロジェクトを、設定した Process Server にデプロイします。Process Server REST API を使用して、ビルドおよびデプロイしたサービスおよびプロジェクトバージョンを含む KIE コンテナを起動、停止、または削除できます。このような柔軟性により、引き続き既存のアプリケーションのワークフローを使用して、Red Hat Process Automation Manager 機能を使用するビジネスアセットを開発できます。

Business Central のプロジェクトは、プロジェクトをビルドしてデプロイする際に自動的にパッケージ化されます。Business Central 以外のプロジェクト (独立した Maven プロジェクト、Java アプリケーションのプロジェクトなど) をビルドしてデプロイする場合は、追加した **kmodule.xml** ファイルに KIE モジュール記述子設定を追加するか、Java アプリケーションに直接指定する必要があります。

### 3.1. KIE モジュール記述子ファイルの設定

KIE モジュールは、追加メタデータファイル **META-INF/kmodule.xml** を持つ Maven プロジェクトまたはモジュールです。Red Hat Process Automation Manager プロジェクトを適切にパッケージングしてデプロイするには **kmodule.xml** ファイルが必要になります。この **kmodule.xml** ファイルは、プロジェクトのアセットの KIE ベースおよび KIE セッション設定を定義する KIE モジュール記述子ファイルです。KIE ベースには、Red Hat Process Automation Manager のルール、プロセス、その他のビジネスアセットのすべてが含まれるリポジトリですが、ランタイムデータは含まれません。KIE セッションは、ランタイムデータを保存および実行し、**kmodule.xml** ファイルに KIE セッションを定義した場合は KIE ベース、または KIE コンテナから直接作成されます。

Business Central 以外のプロジェクト (独立した Maven プロジェクト、Java アプリケーションのプロジェクトなど) を作成する場合は、追加した **kmodule.xml** ファイルに KIE モジュール記述子設定を指定するか、Java アプリケーションに直接指定することでプロジェクトをビルドしてデプロイします。

#### 手順

1. プロジェクトの **~/resources/META-INF** ディレクトリーに、最低でも以下の内容を含む **kmodule.xml** メタデータを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
</kmodule>
```

プロジェクトの **resources** パスで見つかったすべてのファイルを含むデフォルトの KIE ベースを1つ作成するには、この空の **kmodule.xml** ファイルで十分です。デフォルトの KIE ベースには、ビルド時にアプリケーションに KIE コンテナを作成する際に発生するデフォルト KIE セッションも1つ含まれます。

以下は、より高度な **kmodule.xml** ファイルの例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.drools.org/xsd/kmodule">
  <configuration>
    <property key="drools.evaluator.supersetOf"
value="org.mycompany.SupersetOfEvaluatorDefinition"/>
  </configuration>
```

```

<kbase name="KBase1" default="true" eventProcessingMode="cloud"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">
  <ksession name="KSession1_1" type="stateful" default="true" />
  <ksession name="KSession1_2" type="stateful" default="true" beliefSystem="jtsms" />
</kbase>
<kbase name="KBase2" default="false" eventProcessingMode="stream"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
  <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
    <fileLogger file="debugInfo" threaded="true" interval="10" />
    <workItemHandlers>
      <workItemHandler name="name" type="new org.domain.WorkItemHandler()" />
    </workItemHandlers>
    <listeners>
      <ruleRuntimeEventListener type="org.domain.RuleRuntimeListener" />
      <agendaEventListener type="org.domain.FirstAgendaListener" />
      <agendaEventListener type="org.domain.SecondAgendaListener" />
      <processEventListener type="org.domain.ProcessListener" />
    </listeners>
  </ksession>
</kbase>
</kmodule>

```

この例は、KIE ベースを 2 つ定義します。ルールアセットの特定の **パッケージ** は両方 KIE ベースに含まれます。このようにパッケージを指定した場合は、指定したパッケージを反映するディレクトリー構造にルールファイルを整理する必要があります。KIE ベース **KBase1** から 2 つの KIE セッションをインスタンス化し、**KBase2** から KIE セッションを 1 つインスタンス化します。**KBase2** の KIE セッションは **ステートレス** な KIE セッションですが、これは 1 つ前の KIE セッションで呼び出されたデータ (1 つ前のセッションの状態) が、セッションの呼び出しと呼び出しの間で破棄されることを示しています。また、その KIE セッションには、ファイル (またはコンソール) ログ、**WorkItemHandler**、サポートされる 3 種類のリスナー (**ruleRuntimeEventListener**、**agendaEventListener**、および **processEventListener**) も指定されます。**<configuration>** 要素は、**kmodule.xml** ファイルをさらにカスタマイズするのに使用できる任意のプロパティを定義します。

プロジェクトに **kmodule.xml** ファイルを手動で追加する代わりに、Java アプリケーションの **KieModuleModel** インスタンスを使用するか、プログラムで **kmodule.xml** ファイルを作成し、KIE ベースおよび KIE セッションを定義し、KIE 仮想ファイルシステム **KieFileSystem** に、プロジェクトのリソースをすべて追加します。

### プログラムを使用して **kmodule.xml** を作成し、**KieFileSystem** に追加

```

import org.kie.api.KieServices;
import org.kie.api.builder.model.KieModuleModel;
import org.kie.api.builder.model.KieBaseModel;
import org.kie.api.builder.model.KieSessionModel;
import org.kie.api.builder.KieFileSystem;

KieServices kieServices = KieServices.Factory.get();
KieModuleModel kieModuleModel = kieServices.newKieModuleModel();

KieBaseModel kieBaseModel1 = kieModuleModel.newKieBaseModel("KBase1")
    .setDefault(true)
    .setEqualsBehavior(EqualityBehaviorOption.EQUALITY)
    .setEventProcessingMode(EventProcessingOption.STREAM);

```

```
KieSessionModel ksessionModel1 = kieBaseModel1.newKieSessionModel("KSession1_1")
    .setDefault(true)
    .setType(KieSessionModel.KieSessionType.STATEFUL)
    .setClockType(ClockTypeOption.get("realtime"));

KieFileSystem kfs = kieServices.newKieFileSystem();
kfs.writeKModuleXML(kieModuleModel.toXML());
```

2. 手動またはプログラムで **kmodule.xml** ファイルをプロジェクトに設定したら、設定を検証する KIE コンテナから KIE ベースおよび KIE セッションを取得します。

```
KieServices kieServices = KieServices.Factory.get();
KieContainer kContainer = kieServices.getKieClasspathContainer();

KieBase kBase1 = kContainer.getKieBase("KBase1");
KieSession kieSession1 = kContainer.newKieSession("KSession1_1"),
    kieSession2 = kContainer.newKieSession("KSession1_2");

KieBase kBase2 = kContainer.getKieBase("KBase2");
StatelessKieSession kieSession3 = kContainer.newStatelessKieSession("KSession2_1");
```

**kmodule.xml** ファイルに、**KieBase** または **KieSession** を **default="true"** と設定している場合は、先ほどの **kmodule.xml** 例のように、名前を渡さずに KIE コンテナから取得できます。

```
KieContainer kContainer = ...

KieBase kBase1 = kContainer.getKieBase();
KieSession kieSession1 = kContainer.newKieSession(),
    kieSession2 = kContainer.newKieSession();

KieBase kBase2 = kContainer.getKieBase();
StatelessKieSession kieSession3 = kContainer.newStatelessKieSession();
```

**kmodule.xml** ファイルの詳細は、Red Hat Process Automation Manager [VERSION] Source Distribution ZIP ファイルを [Red Hat カスタマーポータル](#) から取得し、`$FILE_HOME/rhpam-$VERSION-sources/kie-api-parent-$VERSION/kie-api/src/main/resources/org/kie/api/` に保存してある XML スキーマ **kmodule.xsd** を参照してください。

### 3.1.1. KIE モジュール設定のプロパティ

プロジェクトにおいて、KIE モジュール記述子ファイル (**kmodule.xml**) の任意の **<configuration>** 要素は、プロパティの キー および 値 ペアを定義し、**kmodule.xml** ファイルをさらにカスタマイズするのに使用できます。

#### **kmodule.xml** ファイルの設定プロパティの例

```
<kmodule>
...
<configuration>
  <property key="drools.dialect.default" value="java"/>
...
</configuration>
...
</kmodule>
```

以下は、プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) でサポートされる **<configuration>** プロパティのキーおよび値です。

#### drools.dialect.default

デフォルトの Drools 方言を設定します。  
サポートされる値: **java**、**mvel**

```
<property key="drools.dialect.default"
  value="java"/>
```

#### drools.accumulate.function.\$FUNCTION

指定した関数名に累積関数を実装するクラスのリンク。デシジョンエンジンにカスタムの累積関数を追加できます。

```
<property key="drools.accumulate.function.hyperMax"
  value="org.drools.custom.HyperMaxAccumulate"/>
```

#### drools.evaluator.\$EVALUATION

デシジョンエンジンにカスタムのエバリュエーターを追加できるように、指定したエバリュエーター名にエバリュエーター定義を実装するクラスをリンクします。エバリュエーターは、カスタムオペレーターと類似しています。

```
<property key="drools.evaluator.soundslike"
  value="org.drools.core.base.evaluators.SoundslikeEvaluatorsDefinition"/>
```

#### drools.dump.dir

Red Hat Process Automation Manager の **dump/log** ディレクトリーにパスを設定します。

```
<property key="drools.dump.dir"
  value="$DIR_PATH/dump/log"/>
```

#### drools.defaultPackageName

プロジェクトのビジネスアセットにデフォルトパッケージを設定します。

```
<property key="drools.defaultPackageName"
  value="org.domain.pkg1"/>
```

#### drools.parser.processStringEscapes

文字列のエスケープ機能を設定します。このプロパティを **false** に設定すると、**\n** 文字が改行文字として解釈されません。

サポートされる値: **true** (デフォルト)、**false**

```
<property key="drools.parser.processStringEscapes"
  value="true"/>
```

#### drools.kbuilder.severity.\$DUPLICATE

KIE ベースがビルドされたときに報告される重複したルール、プロセス、または関数のインスタンスの重大度を設定します。たとえば、**duplicateRule** を **ERROR** に設定すると、KIE ベースのビルド時に検出された重複ルールに対してエラーが生成されます。

サポートされるキー接尾辞: **duplicateRule**、**duplicateProcess**、**duplicateFunction**

サポートされる値: **INFO**、**WARNING**、**ERROR**

```
<property key="drools.kbuilder.severity.duplicateRule"
value="ERROR"/>
```

### drools.propertySpecific

デシジョンエンジンのプロパティの反応を設定します。

サポートされる値: **DISABLED**、**ALLOWED**、**ALWAYS**

```
<property key="drools.propertySpecific"
value="ALLOWED"/>
```

### drools.lang.level

DRL 言語レベルを設定します。

サポートされる値: **DRL5**、**DRL6**、**DRL6\_STRICT** (デフォルト)

```
<property key="drools.lang.level"
value="DRL_STRICT"/>
```

## 3.1.2. KIE モジュールでサポートされる KIE ベース属性

KIE ベースは、プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) を定義するリポジトリで、Red Hat Process Automation Manager のルール、プロセス、その他のビジネスアセットが含まれます。**kmodule.xml** ファイルで KIE ベースを定義した場合は、特定の属性および値を指定して、KIE ベース設定をさらにカスタマイズできます。

### kmodule.xml ファイルの KIE ベース設定例

```
<kmodule>
...
<kbase name="KBase2" default="false" eventProcessingMode="stream" equalsBehavior="equality"
declarativeAgenda="enabled" packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">
...
</kbase>
...
</kmodule>
```

以下は、プロジェクトの KIE モジュール記述ファイル (**kmodule.xml**) でサポートされる **kbase** 属性および値です。

表3.1 KIE モジュールでサポートされる KIE ベース属性

属性	サポートされている値	説明
<b>name</b>	すべての名前	<b>KieContainer</b> から <b>KieBase</b> を取得する名前を定義します。この属性は必須です。

属性	サポートされている値	説明
<b>includes</b>	KIE モジュールのその他の KIE ベースオブジェクトのコンマ区切り一覧	この KIE ベースに追加するその他の KIE ベースオブジェクトとアーティファクトを定義します。モジュールの <b>pom.xml</b> ファイルに依存関係として宣言している場合は、KIE ベースを複数の KIE モジュールに追加できます。
<b>packages</b>	KIE ベースに追加するパッケージのコンマ区切りの一覧  デフォルト値: <b>all</b>	(ルールやプロセスなど) この KIE ベースに追加するアーティファクトのパッケージを定義します。デフォルトでは、 <b>~/resources</b> ディレクトリーのすべてのアーティファクトは KIE ベースに含まれます。この属性は、コンパイルしたアーティファクトの数を制限できます。この属性に指定した一覧に含まれるパッケージだけがコンパイルされます。
<b>default</b>	<b>true</b> 、 <b>false</b>  デフォルト値: <b>false</b>	KIE ベースは、モジュールのデフォルトの KIE ベースで、名前を渡さずに KIE コンテナから作成できます。各モジュールにはデフォルトの KIE ベースを1つだけ指定できます。
<b>equalsBehavior</b>	<b>identity</b> 、 <b>equality</b>  デフォルト値: <b>identity</b>	新しいファクトが作業メモリーに挿入された場合の Red Hat Process Automation Manager の動作を定義します。 <b>identity</b> に設定した場合は、同じオブジェクトが作業メモリーに存在する場合を除いて、常に新しい <b>FactHandle</b> が作成されます。 <b>equality</b> に設定した場合は、 <b>equals()</b> メソッドにより、新たに挿入したオブジェクトが既存のファクトと同一でないと判断された場合に限り、新しい <b>FactHandle</b> が作成されます。
<b>eventProcessingMode</b>	<b>cloud</b> 、 <b>stream</b>  デフォルト値: <b>cloud</b>	イベントが KIE ベースで処理される方法を指定します。このプロパティーを <b>cloud</b> に設定すると、KIE ベースはイベントを通常のファクトとして扱います。 <b>stream</b> に設定すると、イベントに対する一時的な推論が可能になります。
<b>declarativeAgenda</b>	<b>disabled</b> 、 <b>enabled</b>  デフォルト値: <b>disabled</b>	宣言型アジェンダが有効かどうかを指定します。

### 3.1.3. KIE モジュールでサポートされる KIE セッション属性

KIE セッションがランタイムデータを保存および実行し、プロジェクトの KIE モジュール記述子ファイ

ル (**kmodule.xml**) で KIE セッションを定義した場合は KIE ベース、または KIE コンテナから直接作成されます。KIE ベースおよび KIE セッションを **kmodule.xml** ファイルに定義すると、特定の属性および値を指定して、KIE セッション設定をさらにカスタマイズできます。

### kmodule.xml ファイルの KIE セッション設定例

```
<kmodule>
...
<kbase>
...
  <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
...
</kbase>
...
</kmodule>
```

以下は、プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) でサポートされている **ksession** 属性および値です。

表3.2 KIE モジュールでサポートされる KIE セッション属性

属性	サポートされている値	説明
<b>name</b>	すべての名前	<b>KieContainer</b> から <b>KieSession</b> を取得する名前を定義します。この属性は必須です。
<b>type</b>	<b>stateful</b> 、 <b>stateless</b> デフォルト値: <b>stateful</b>	KIE セッションの呼び出しと呼び出しの間にデータを保持 ( <b>stateful</b> ) するか、破棄 ( <b>stateless</b> ) するかを指定します。 <b>stateful</b> に設定したセッションでは、作業メモリーを繰り返し使用できますが、 <b>stateless</b> に設定したセッションでは、通常、アセットを1回だけ実行するために使用されます。 <b>stateless</b> セッションは、新しいファクトが追加、更新、または削除され、ルールが実行されるたびに変更するナレッジの状態を保存します。 <b>stateless</b> セッションの実行には、ルール実行など、以前のアクションに関する情報はありません。
<b>default</b>	<b>true</b> 、 <b>false</b> デフォルト値: <b>false</b>	KIE セッションをモジュールのデフォルトセッションにし、名前を渡さずに KIE コンテナから作成できるようにするかどうかを指定します。各モジュールにはデフォルトの KIE セッションを1つだけ指定できます。



属性	サポートされている値	説明
<b>clockType</b>	<b>realtime、pseudo</b> デフォルト値: <b>realtime</b>	イベントのタイムスタンプをシステムクロック、または疑似クロックから割り当てられるかどうかを指定します。このクロックは、一時的なルールをテストするユニットで特に便利です。
<b>beliefSystem</b>	<b>simple、jts、defeasible</b> デフォルト値: <b>simple</b>	KIE セッションが使用する信念体系の種類を定義します。信念体系は、ナレッジ (ファクト) から事実を推測します。たとえば、後ほどデシジョンエンジンから削除される別のファクトに基づいて新しいファクトが挿入されると、この体系では、新たに挿入されたファクトも削除する必要があると判断します。

## 3.2. RED HAT PROCESS AUTOMATION MANAGER プロジェクトの MAVEN でのパッケージ化およびデプロイ

Business Central 以外の Maven プロジェクトを、設定した Process Server にデプロイする場合は、プロジェクトの **pom.xml** ファイルを編集して、プロジェクトを KJAR ファイルとしてパッケージ化し、プロジェクトのアセットに対する KIE ベースおよび KIE セッションの設定が含まれる **kmodule.xml** ファイルを追加します。

### 前提条件/事前作業

- Red Hat Process Automation Manager ビジネスアセットを含む Maven 化したプロジェクトがある。
- Process Server がインストールされており、**kie-server** ユーザーアクセスが設定されている。インストールオプションは『[Red Hat Process Automation Manager インストールの計画](#)』を参照してください。

### 手順

- Maven プロジェクトの **pom.xml** ファイルで、パッケージタイプを **kjar** に設定し、**kie-maven-plugin** ビルドコンポーネントを追加します。

```
<packaging>kjar</packaging>
...
<build>
  <plugins>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>${rhpm.version}</version>
      <extensions>true</extensions>
    </plugin>
  </plugins>
</build>
```



**kjar** パッケージングタイプは、**kie-maven-plugin** コンポーネントをアクティブにして、アーティファクトリソースを検証してプリコンパイルします。**<version>** は、プロジェクトで現在使用される Red Hat Process Automation Manager の Maven アーティファクトのバージョン (例: 7.18.0.Final-redhat-00002) で、デプロイメントに Maven プロジェクトを適切にパッケージ化するのに必要です。

## 注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用します。BOM ファイルを追加すると、指定の Maven リポジトリからの一時的な依存関係の内、正しいバージョンが、このプロジェクトに追加されます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.3.0.GA-redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、[「What is the mapping between Red Hat Process Automation Manager and the Maven library version?」](#) を参照してください。

- 以下の依存関係を **pom.xml** ファイルに追加して、ルールアセットから実行可能なルールモデルを生成します。

- **drools-canonical-model**: Red Hat Process Automation Manager から独立するルールセットモデルの実行可能な正規表現を有効にします。
- **drools-model-compiler**: デンジョンエンジンで実行できるように Red Hat Process Automation Manager の内部データ構造に実行可能なモデルをコンパイルします。

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-canonical-model</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

実行可能ルールモデルは埋め込み可能なモデルで、ビルド時に実行するルールセットの Java ベース表記を提供します。実行可能モデルは Red Hat Process Automation Manager の標準アセットパッケージングの代わりとなるもので、より効率的です。KIE コンテナと KIE ベース

の作成がより迅速にでき、DRL (Drools Rule Language) ファイルリストや他の Red Hat Process Automation Manager アセットが多い場合は、特に有効です。

実行可能なルールモデルの詳細は、[『DRL ルールを使用したデシジョンサービスの作成』](#)を参照してください。

プロジェクト内にある DMN アセットの Decision Model and Notation (DMN) 実行可能モデルを有効にするには (該当する場合)、**pom.xml** ファイルに **kie-dmn-core** の依存関係も追加します。

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-core</artifactId>
  <scope>provided</scope>
  <version>${rhpm.version}</version>
</dependency>
```

3. Maven プロジェクトの **~/resources** ディレクトリーに、最低でも以下の内容を含む **META-INF/kmodule.xml** メタデータファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
</kmodule>
```

この **kmodule.xml** ファイルは、すべての Red Hat Process Automation Manager プロジェクトに必要な KIE モジュール記述子です。KIE モジュールを使用して、1つ以上の KIE ベースを定義し、各 KIE ベースに1つ以上の KIE セッションを定義します。

**kmodule.xml** 設定の詳細は「[KIE モジュール記述子ファイルの設定](#)」を参照してください。

4. Maven プロジェクトの関連リソースで、**.java** クラスを設定して KIE コンテナおよび KIE セッションを作成して、KIE ベースをロードします。

```
import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public void testApp() {

    // Load the KIE base:
    KieServices ks = KieServices.Factory.get();
    KieContainer kContainer = ks.getKieClasspathContainer();
    KieSession kSession = kContainer.newKieSession();

}
```

この例では、KIE コンテナは、**testApp** プロジェクトのクラスパスからビルドしたファイルを読み込みます。**KieServices** API を使用すれば、KIE ビルド設定およびランタイム設定のすべてにアクセスできます。

プロジェクトの **ReleaseId** を **KieServices** API に渡して KIE コンテナを作成することもできます。**ReleaseId** は、プロジェクトの **pom.xml** ファイルの **GroupId** 値、**ArtifactId** 値、**Version** 値 (GAV) から生成します。

```
import org.kie.api.KieServices;
```

```
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public void testApp() {

    // Identify the project in the local repository:
    ReleaseId rid = new ReleaseId();
    rid.setGroupId("com.sample");
    rid.setArtifactId("my-app");
    rid.setVersion("1.0.0");

    // Load the KIE base:
    KieServices ks = KieServices.Factory.get();
    KieContainer kContainer = ks.newKieContainer(rid);
    KieSession kSession = kContainer.newKieSession();

}
```

5. コマンドターミナルで Maven プロジェクトディレクトリーに移動して、以下のコマンドを実行し、実行可能なモデルからプロジェクトをビルドします。

```
mvn clean install -DgenerateModel=<VALUE>
```

ルールアセットが実行可能なルールでビルドされるように、**-DgenerateModel=<VALUE>** プロパティーで、プロジェクトが DRL ベースの KJAR ではなく、モデルベースの KJAR としてビルドできるようにします。

**<VALUE>** は、3 つの値のいずれかに置き換えます。

- **YES:** オリジナルプロジェクトの DRL ファイルに対応する実行可能なモデルを生成し、生成した KJAR から DRL ファイルを除外します。
- **WITHDRL:** オリジナルプロジェクトの DRL ファイルに対応する実行可能なモデルを生成し、文書化の目的で、生成した KJAR に DRL ファイルを追加します (KIE ベースはいずれの場合でも実行可能なモデルからビルドされます)。
- **NO:** 実行可能なモデルは生成されません。

ビルドコマンドの例:

```
mvn clean install -DgenerateModel=YES
```

DMN 実行可能なモデルの場合には、以下のコマンドを実行します。

```
mvn clean install -DgenerateDMNModel=YES
```

ビルドに失敗したら、コマンドラインのエラーメッセージに記載されている問題に対応し、ビルドに成功するまでファイルの妥当性確認を行います。

6. プロジェクトをローカルで正常にビルドしてテストした後に、プロジェクトをリモートの Maven リポジトリにデプロイします。

```
mvn deploy
```

### 3.3. RED HAT PROCESS AUTOMATION MANAGER プロジェクトの JAVA アプリケーションでのパッケージ化およびデプロイ

お使いの Java アプリケーションから、設定した Process Server にプロジェクトをデプロイする場合は、**KieModuleModel** インスタンスを使用して **kmodule.xml** ファイルをプログラムで作成して KIE ベースおよび KIE セッションを定義し、プロジェクトのすべてのリソースを、KIE 仮想ファイルシステム **KieFileSystem** に追加します。

#### 前提条件/事前作業

- Red Hat Process Automation Manager ビジネスアセットを含む Java アプリケーションがある。
- Process Server がインストールされており、**kie-server** ユーザーアクセスが設定されている。インストールオプションは『[Red Hat Process Automation Manager インストールの計画](#)』を参照してください。

#### 手順

- クライアントアプリケーションで、Java プロジェクトの関連のクラスパスに、以下の依存関係を追加して、ルールアセットから実行可能なルールモデルを生成します。
  - drools-canonical-model**: Red Hat Process Automation Manager から独立するルールセットモデルの実行可能な正規表現を有効にします。
  - drools-model-compiler**: デシジョンエンジンで実行できるように Red Hat Process Automation Manager の内部データ構造に実行可能なモデルをコンパイルします。

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-canonical-model</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

実行可能ルールモデルは埋め込み可能なモデルで、ビルド時に実行するルールセットの Java ベース表記を提供します。実行可能モデルは Red Hat Process Automation Manager の標準アセットパッケージングの代わりとなるもので、より効率的です。KIE コンテナと KIE ベースの作成がより迅速にでき、DRL (Drools Rule Language) ファイルリストや他の Red Hat Process Automation Manager アセットが多い場合は、特に有効です。

実行可能なルールモデルの詳細は、『[DRL ルールを使用したデシジョンサービスの作成](#)』を参照してください。

プロジェクト内にある DMN アセットの Decision Model and Notation (DMN) 実行可能モデルを有効にするには (該当する場合)、**pom.xml** ファイルに **kie-dmn-core** の依存関係も追加します。

```
<dependency>
  <groupId>org.kie</groupId>
```

```
<artifactId>kie-dmn-core</artifactId>
<scope>provided</scope>
<version>${rhpam.version}</version>
</dependency>
```

**<version>** は、プロジェクトで現在使用する Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.18.0.Final-redhat-00002)。

## 注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Decision Manager と Red Hat Process Automation Manager の両方に適用します。BOM ファイルを追加すると、指定の Maven リポジトリからの一時的な依存関係の内、正しいバージョンが、このプロジェクトに追加されます。

BOM 依存関係の例:

```
<dependency>
<groupId>com.redhat.ba</groupId>
<artifactId>ba-platform-bom</artifactId>
<version>7.3.0.GA-redhat-00002</version>
<scope>import</scope>
<type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、[「What is the mapping between Red Hat Process Automation Manager and the Maven library version?」](#) を参照してください。

2. **KieServices** API を使用して、必要な KIE ベースおよび KIE セッションを持つ **KieModuleModel** インスタンスを作成します。**KieServices** API を使用して、KIE ビルド設定およびランタイム設定にアクセスできます。**KieModuleModel** インスタンスは、プロジェクトの **kmodule.xml** ファイルを生成します。**kmodule.xml** 設定の詳細は [「KIE モジュール記述子ファイルの設定」](#) を参照してください。
3. **KieModuleModel** インスタンスを XML に変換し、XML を **KieFileSystem** に追加します。

プログラムを使用して **kmodule.xml** を作成し、**KieFileSystem** に追加

```
import org.kie.api.KieServices;
import org.kie.api.builder.model.KieModuleModel;
import org.kie.api.builder.model.KieBaseModel;
import org.kie.api.builder.model.KieSessionModel;
import org.kie.api.builder.KieFileSystem;

KieServices kieServices = KieServices.Factory.get();
KieModuleModel kieModuleModel = kieServices.newKieModuleModel();

KieBaseModel kieBaseModel1 = kieModuleModel.newKieBaseModel("KBase1")
    .setDefault(true)
    .setEqualsBehavior(EqualityBehaviorOption.EQUALITY)
    .setEventProcessingMode(EventProcessingOption.STREAM);
```

```
KieSessionModel ksessionModel1 = kieBaseModel1.newKieSessionModel("KSession1")
    .setDefault(true)
    .setType(KieSessionModel.KieSessionType.STATEFUL)
    .setClockType(ClockTypeOption.get("realtime"));

KieFileSystem kfs = kieServices.newKieFileSystem();
kfs.writeKModuleXML(kieModuleModel.toXML());
```

- プロジェクトで使用する残りの Red Hat Process Automation Manager アセットをすべて **KieFileSystem** インスタンスに追加します。アーティファクトは、Maven プロジェクトファイル構造に含まれる必要があります。

```
import org.kie.api.builder.KieFileSystem;

KieFileSystem kfs = ...
kfs.write("src/main/resources/KBase1/ruleSet1.drl", stringContainingAValidDRL)
    .write("src/main/resources/dtable.xls",
        kieServices.getResources().newInputStreamResource(dtableFileStream));
```

この例では、プロジェクトアセットは、**String** 変数および **Resource** インスタンスの両方として追加されます。**Resource** インスタンスは **KieResources** ファクトリーを使用して作成され、**KieServices** インスタンスにより提供されます。**KieResources** クラスは、**InputStream** オブジェクト、**URL** オブジェクト、および **File** オブジェクト、またはファイルシステムのパスを示す **String** を、**KieFileSystem** が管理する **Resource** インスタンスに変換する factory メソッドを提供します。

プロジェクトのアーティファクトを **KieFileSystem** に追加する際に、**ResourceType** プロパティを **Resource** オブジェクトに明示的に割り当てることもできます。

```
import org.kie.api.builder.KieFileSystem;

KieFileSystem kfs = ...
kfs.write("src/main/resources/myDrl.txt",
    kieServices.getResources().newInputStreamResource(drlStream)
        .setResourceType(ResourceType.DRL));
```

- 実行可能なモデルから **KieFileSystem** のコンテンツをビルドするように、**buildAll(ExecutableModelProject.class)** を指定して **KieBuilder** を使用し、KIE コンテナを作成して、デプロイします。

```
2 import org.kie.api.KieServices;
import org.kie.api.KieServices.Factory;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieBuilder;
import org.kie.api.runtime.KieContainer;

KieServices kieServices = KieServices.Factory.get();
KieFileSystem kfs = ...

KieBuilder kieBuilder = ks.newKieBuilder( kfs );
// Build from an executable model
kieBuilder.buildAll( ExecutableModelProject.class )
assertEquals(0, kieBuilder.getResults().getMessages(Message.Level.ERROR).size());
```



```
KieContainer kieContainer = kieServices
    .newKieContainer(kieServices.getRepository().getDefaultReleaseId());
```

実行可能なモデルから **KieFileSystem** をビルドした後に、作成された **KieSession** は効率のあまりよくない **mvel** 式ではなく、**lambda** 式をもとにした制約を使用します。実行可能なモデルなしに、標準の手法でプロジェクトをビルドするには、**buildAll()** で引数を指定しないでください。

**ERROR** ビルドは、プロジェクトのコンパイルに失敗し、**KieModule** が作成されず、**KieRepository** シングルトンに何も追加されないことを示しています。**WARNING** または **INFO** の結果は、プロジェクトのコンパイルが成功したことで、ビルドプロセスの詳細を示しています。

### 3.4. PROCESS SERVER のサービスの起動

Business Central 以外の Maven または Java プロジェクトから Red Hat Process Automation Manager アセットをデプロイした場合は、Process Server REST API 呼び出しを使用し、KIE コンテナ (デプロイメントユニット) およびこのコンテナ内のサービスを起動できます。デプロイメントの種類 (Business Central からのデプロイメントを含む) にかかわらず、サービスは Process Server REST API を使用して起動できますが、Business Central からデプロイしたプロジェクトは、自動的に起動されるか、Business Central インターフェースで起動できます。

#### 前提条件/事前作業

Process Server がインストールされており、**kie-server** ユーザーアクセスが設定されている。インストールオプションは『[Red Hat Process Automation Manager インストールの計画](#)』を参照してください。

#### 手順

コマンドターミナルで以下の API 要求を実行し、Process Server の KIE コンテナにサービスをロードして起動します。

```
$ curl --user "<username>:<password>" -H "Content-Type: application/json" -X PUT -d '{"container-id" : "<containerID>","release-id" : {"group-id" : "<groupID>","artifact-id" : "<artifactID>","version" : "<version>"}' http://<serverhost>:<serverport>/kie-server/services/rest/server/containers/<containerID>
```

以下の値を置き換えてください。

- **<username>**、**<password>**: **kie-server** ロールを持つユーザーのユーザー名およびパスワード。
- **<containerID>**: KIE コンテナ (デプロイメントユニット) の識別子。ランダムな識別子を使用することもできますが、コマンドの URL およびデータの両方で同じものを使用する必要があります。
- **<groupID>**、**<artifactID>**、**<version>**: プロジェクトの GAV 値。
- **<serverhost>**: Process Server のホスト名 (Process Server と同じホストでコマンドを実行する場合は **localhost**)。
- **<serverport>**: Process Server のポート番号。

例:

```
curl --user "rhpamAdmin:password@1" -H "Content-Type: application/json" -X PUT -d '{"container-id": "kie1", "release-id": {"group-id": "org.kie.server.testing", "artifact-id": "container-crud-tests1", "version": "2.1.0.GA"}}' http://localhost:39043/kie-server/services/rest/server/containers/kie1
```

### 3.5. PROCESS SERVER のサービスの停止および削除

Business Central 以外の Maven または Java プロジェクトから Red Hat Process Automation Manager サービスを起動した場合は、Process Server REST API 呼び出しを使用し、サービスを含む KIE コンテナ (デプロイメントユニット) を停止して削除できます。デプロイメントの種類 (Business Central からのデプロイメントを含む) にかかわらずサービスは、Process Server REST API を使用して停止できますが、Business Central のサービスは Business Central インターフェースで停止できます。

#### 前提条件/事前作業

Process Server がインストールされており、**kie-server** ユーザーアクセスが設定されている。インストールオプションは『[Red Hat Process Automation Manager インストールの計画](#)』を参照してください。

#### 手順

コマンドターミナルで、以下の API 要求を実行して、Process Server のサービスで KIE コンテナを停止および削除します。

```
$ curl --user "<username>:<password>" -X DELETE http://<serverhost>:<serverport>/kie-server/services/rest/server/containers/<containerID>
```

以下の値を置き換えてください。

- **<username>**、**<password>**: **kie-server** ロールを持つユーザーのユーザー名およびパスワード。
- **<containerID>**: KIE コンテナ (デプロイメントユニット) の識別子。ランダムな識別子を使用することもできますが、コマンドの URL およびデータの両方で同じものを使用する必要があります。
- **<serverhost>**: Process Server のホスト名 (Process Server と同じホストでコマンドを実行する場合は **localhost**)。
- **<serverport>**: Process Server のポート番号。

例:

```
curl --user "rhpamAdmin:password@1" -X DELETE http://localhost:39043/kie-server/services/rest/server/containers/kie1
```



## 第4章 関連資料

- 『DRL ルールを使用したデシジョンサービスの作成』の「ルールの実行」
- 『Red Hat OpenShift Container Platform への Red Hat Process Automation Manager オーサリング環境のデプロイメント』
- 『Red Hat OpenShift Container Platform への Red Hat Process Automation Manager 管理サーバー環境のデプロイメント』
- 『Red Hat OpenShift Container Platform への Red Hat Process Automation Manager イミュータブルサーバー環境のデプロイメント』

## 付録A バージョン情報

本ドキュメントの最終更新日: 2019 年 5 月 8 日 (水)