Red Hat Process Automation Manager 7.7

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

Last Updated: 2020-04-01

Red Hat Process Automation Manager 7.7 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

概要

本書は、Red Hat Process Automation Manager 7.7 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

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 KIE 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て使用できます。たとえば、Business Central のプロジェクトを開発して、KIE 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

- 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 KIE 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 コンテナー、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 リポジトリーに保存され、**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 KIE 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、KIE 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 リポジトリー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 コンテナー) を起動、停止、または削除できます。

(**Menu → Deploy → Execution Servers** で) 複数の KIE 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。

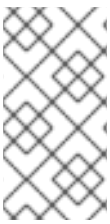
たとえば、**Test** 設定のテストサーバーと、**Production** 設定の実稼働サーバーを使用できます。ビジネスアセットとサービスをプロジェクトに開発し、**Test** サーバー設定にプロジェクトをデプロイしてから、十分にテストしたプロジェクトのバージョンを **Production** サーバー設定にデプロイできます。このとき、プロジェクトの開発を継続するには、プロジェクト設定でバージョンを変更します。これにより、組み込み Maven リポジトリーで、新しいバージョンと古いバージョンが別のアーティファクトと見なされるため、**Test** サーバー設定に新しいバージョンをデプロイし、**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、KIE 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 KIE SERVER の設定

KIE 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 KIE Server を追加する必要がある場合は、KIE 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 KIE Server をデプロイする場合は、『Red Hat OpenShift Container Platform への Red Hat Process Automation Manager freeform 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

- KIE 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、以下のプロパティーを設定します。

   - **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:** KIE 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 KIE 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、KIE 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.pwd" 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. KIE Server を起動または再起動します。

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

KIE 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 コンテナー) 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、Business Central で **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 KIE 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全インスタンスを置き換えることもできます。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 コンテナー) が同じターゲット KIE 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 KIE Server 上の新規デプロイメントユニット (KIE コンテナー) にデプロイすることのみが可能です。

手順

1. KIE 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ーを **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。

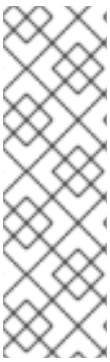
注記

デフォルトでは、KIE 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 KIE 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 KIE SERVER への外部 MAVEN リポジトリーの設定

Business Central および KIE Server が、内部のリポジトリーではなく、Nexus や Artifactory などの外部の Maven リポジトリーを使用するように設定できます。このように設定することで、Business Central と KIE Server は外部の Maven リポジトリーで管理されているアーティファクトにアクセスしてダウンロードできます。



注記

Red Hat OpenShift Container Platform のオーサリング環境向けに外部の Maven リポジトリーを設定する方法については、以下のドキュメントを参照してください。

- 『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』
- 『Operator を使用した Red Hat OpenShift Container Platform への Red Hat Process Automation Manager 環境のデプロイメント』

前提条件

- Business Central および KIE 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. 外部リポジトリー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** プロパティーに **settings.xml** ファイルのフルパス名を設定します。
以下は例を示しています。

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

-

5. Business Central と KIE Server を起動または再起動します。

次のステップ

外部の Maven リポジトリーに KJAR アーティファクトとしてエクスポートまたはプッシュする Business Central プロジェクトごとに、プロジェクトの **pom.xml** ファイルにリポジトリーの情報を追加する必要があります。詳細は、「Business Central プロジェクトの外部 Maven リポジトリーへのエクスポート」を参照してください。

2.4. BUSINESS CENTRAL プロジェクトの外部 MAVEN リポジトリーへのエクスポート

Business Central および KIE Server 向けの外部 Maven リポジトリーを設定した場合には、外部のリポジトリーに KJAR アーティファクトとしてエクスポートまたはプッシュする Business Central プロジェクトごとに、リポジトリー情報を **pom.xml** ファイルに追加する必要があります。こうすることで、必要に応じてプロジェクトの KJAR ファイルをリポジトリーに移動して統合プロセスを実装し、Business Central または KIE 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

- Business Central と KIE Server が外部の Maven リポジトリーを使用するように設定されている。オンプレミスで Business Central をデプロイした場合の外部 Maven リポジトリーの設定に関する詳細は、「Business Central および KIE Server への外部 Maven リポジトリーの設定」を参照してください。Red Hat OpenShift Container Platform にオーサリング環境をデプロイした場合の詳細は、以下のドキュメントを参照してください。
  - 『**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**』
  - 『**Operator を使用した Red Hat OpenShift Container Platform への Red Hat Process Automation Manager 環境のデプロイメント**』

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、プロジェクト内の任意のアセットを選択します。
2. 画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。
3. プロジェクトの **pom.xml** ファイルの最後 (**</project>** の終了タグの前) に以下の設定を追加します。値は、**settings.xml** ファイルに定義した設定に対応する必要があります。

```
<distributionManagement>
<repository>
<id>${maven-repo-id}</id>
<url>${maven-repo-url}</url>
<layout>default</layout>
</repository>
</distributionManagement>
```

4. **Save** をクリックして **pom.xml** ファイルの変更を保存します。

外部の Maven リポジトリーに KJAR アーティファクトとしてエクスポートまたはプッシュする Business Central プロジェクトごとに、この手順を繰り返してください。

2.5. 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 KIE 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、KIE Server にデプロイします。KIE 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 KIE Server にデプロイせずに設定済みの Maven リポジトリー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、ビルドされた KJAR ファイルを KIE Server に、(該当する場合) 実行中のインスタンスを停止せずにデプロイできます。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すべてのインスタンスを置き換えることもできます。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 コンテナー) が同じターゲット KIE 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、ビルドされた KJAR ファイルを KIE Server 上の新規デプロイメントユニット (KIE コンテナー) にデプロイすることのみが可能です。

KIE 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ーを **org.kie.server.mode=development** または **org.kie.server.mode=production** に設定します。Business Central でそれぞれのプロジェクト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、KIE 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 KIE 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したりすることはできません。

Business Central に KIE Server を 1 つだけ接続する場合、または接続したすべての KIE 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 コンテナー) 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:** KIE 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 コンテナー) 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。KIE コンテナーのエイリアスの詳細は、「KIE コンテナーのエイリアス」を参照してください。

- **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 コンテナー) 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。

プロジェクトのデプロイメントに関する詳細を確認するには、画面の上部にあるデプロイメントバナーの **View deployment details** か、**Deploy** のドロップダウンメニューをクリックします。このオプションを使用すると、**Menu → Deploy → Execution Servers** ページに移動します。

2.6. BUSINESS CENTRAL のデプロイメントユニット

設定した KIE Server 上のインスタンス化された KIE コンテナー、または **デプロイメントユニット** を介して、プロジェクト内のサービスは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 KIE 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。

2.6.1. 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

- 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. 必要に応じて、**Alias** フィールドにエイリアスを追加します。
5. 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。
6. **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。
7. **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 KIE Server に配置されました。**Start Deployment Unit?**を選択した場合は、サービスが起動します。

2.6.2. 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 KIE Server を 1 つだけ接続する場合、または接続したすべての KIE 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。

2.6.3. KIE コンテナーのエイリアス

KIE コンテナー (デプロイメントユニット) のエイリアスは、KIE Server インスタンスのプロキシーとして、同じコンテナーをデプロイする場合に異なるバージョンを処理しやすくします。コンテナーがアップグレードされると、リンクされているエイリアスは新しいバージョンのコンテナーを自動的に参照します。KIE コンテナーのエイリアスの詳細は、「Business Central でのデプロイメントユニットの作成」を参照してください。

たとえば、新しいバージョンのコンテナーをデプロイするたびにクライアントアプリケーションが変わる場合には、クライアントアプリケーションがコンテナーのエイリアスを参照するようにできます。新しいコンテナーのバージョンがデプロイされると、関連付けられたエイリアスが自動的に更新され、クライアントアプリケーションを変更しなくても、全要求が自動的に新規コンテナーにルーティングされるようになります。

プロセスが 1 つで、以下のプロパティーを使用するプロジェクト例を見ていきます。

- **GroupId**: org.jbpm
- **ArtifactId**: my-project
- **Version**: 1.0
- **containerID**: my-project

上記のプロジェクトを更新、ビルド、デプロイする場合に、関連付けられたプロジェクトは KIE Server 内で最新のバージョンに更新され、以下のプロパティーを含みます。

- **GroupId**: org.jbpm
- **ArtifactId**: my-project

- **Version**: 2.0

プロジェクトの最新バージョンをデプロイする場合には、**my-project** は以前のバージョンを参照しているので、**containerID** を **my-project2** に更新する必要があります。

注記

プロジェクトバージョンはすべて、**containerID** 名前が含まれます。関連付けられたクライアントアプリケーションでは、対話するプロジェクトの全バージョンを認識しておく必要があります。

コンテナーエイリアスを使用すると、KIE コンテナーを管理しやすくなります。コンテナーの作成時にコンテナーのエイリアスを明示的に設定することも、関連付けられた **ArtifactId** 名をもとに暗黙的に設定することもできます。必要に応じて、複数のコンテナーにエイリアスを 1 つ追加できます。コンテナーエイリアスを指定しない場合には、プロジェクトの **ArtifactId** はデフォルトで、コンテナーエイリアスとして設定されます。

GroupId と **ArtifactId** の名前を含む複数のコンテナーにエイリアスを設定する場合には、KIE Server と対話するたびに同じエイリアスを使用できます。

通常、コンテナエイリアスは以下のユースケースで使用します。

- プロセスバージョンが最新のクライアントアプリケーションで、**新しいプロセスインスタンスを起動する**
- 特定のバージョンの **既存のプロセスを操作する**
- プロセス内の **既存のタスクを操作する**
- **プロセス定義の** イメージやフォームを操作する

たとえば、プロジェクトのバージョン 1.0 をデプロイした後に、POST 要求を以下の KIE Server REST API エンドポイントに送信してプロジェクトのプロセスを開始してみます。

/http://localhost:8230/kie-server/services/rest/server/containers/my-project/processes/evaluation/instances

送信要求は、**org.jbpm:my-project:1.0** から新しいプロセスインスタンスを起動します。org.jbpm:my-project:1.0 の中で **my-project** はコンテナーのエイリアスとして定義しています。後に、プロジェクトのバージョン 2.0 をデプロイする場合に、同じ要求を送信すると、新規インスタンスが **org.jbpm:my-project:2.0** から起動します。**containerID** 名を追加せずに、プロセスの最新バージョンをデプロイできます。

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

GroupId、**ArtifactId**、および **Version** (GAV) 値は、Maven リポジトリーのプロジェクトを識別します。Business Central と KIE Server が同じファイルシステムにあり、同じ Maven リポジトリーを使用する場合は、新しいバージョンのプロジェクトをビルドするたびに、リポジトリーでプロジェクトが自動的に更新されます。ただし、Business Central と KIE Server が別のファイルシステムにあり、ローカルの Maven リポジトリー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。

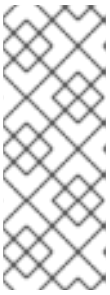
注記

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。

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

Business Central のすべての Maven リポジトリーで、**GroupId**、**ArtifactId**、**Version** (GAV) の各値が重複しているかどうかが確認されます。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す。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 リポジトリーで重複の GAV が確認されます。

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

2.8.1. 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ー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す。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ーの一覧を開きます。
3. 一覧表示したリポジトリー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ー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ーを **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 コンテナー (デプロイメントユニット) のプロジェクトを、設定した KIE Server にデプロイします。KIE 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 コンテナーを起動、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ーですが、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 コンテナー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 コンテナー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="jtms" />
  </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) ロガー、**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ーを定義します。

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 コンテナーから 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 コンテナーから取得できます。

```
KieContainer kContainer = ...

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

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

Red Hat Process Automation Manager ディストリビューションの以下のシステムプロパティーの値を変更して、デシジョンエンジンにキャッシュする KIE モジュールまたはアーティファクトバージョンの最大数を増減できます。

- **kie.repository.project.cache.size**: デシジョンエンジンにキャッシュする最大 KIE モジュール数。デフォルト値: **100**
- **kie.repository.project.versions.cache.size**: デシジョンエンジンにキャッシュする同一のアーティファクトに対して最大指定可能なバージョン数。デフォルト値: **10**

KIE リポジトリー設定の完全一覧については、Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.7.0 Source Distribution** ZIP ファイルをダウンロードして、**~/rhpam-7.7.0-sources/src/drools-$VERSION/drools-compiler/src/main/java/org/drools/compiler/kie/builder/impl/KieRepositoryImpl.java** に移動してください。

kmodule.xml ファイルの詳細は、(まだダウンロードしていない場合には) **Red Hat Process Automation Manager 7.7.0 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ー

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、プロパティーの **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ーの例

```
<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>** プロパティー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。このプロパティーを **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ー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ーで、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"
```

```
sequential="false">
    ...
  </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
name	すべての名前	**KieContainer** から **KieBase** を取得する名前を定義します。**この属性は必須です。**
includes	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 **pom.xml** 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。
packages	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: **all**	(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、~/**resources** 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。
default	**true**、**false** デフォルト値: **false**	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 コンテナーから作成できます。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。

属性	サポートされている値	説明
equalsBehavior	**identity**、**equality** デフォルト値: **identity**	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。**identity** 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 **FactHandle** が作成されます。**equality** に設定時には、挿入した **equals()** メソッドに従い、新しく挿入したオブジェクトが既存のファクトと同じでない場合に新しい **FactHandle** のみが作成されます。オブジェクトを明示的な ID ではなく、機能の同等性をもとに評価する場合には、**equality** モードを使用します。
eventProcessingMode	**cloud**、**stream** デフォルト値: **cloud**	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 **cloud** 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。**stream** 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。
declarativeAgenda	**disabled**、**enabled** デフォルト値: **disabled**	宣言型アジェンダが有効かどうかを指定します。
sequential	**true**、**false** デフォルト値: **false**	シーケンシャルモードが有効化どうかを判断します。シーケンシャルモードの場合は、デシジョンエンジンは、作業メモリー内の変更を考慮せずにデシジョンエンジンのアジェンダに表示されている順番に、ルールを一度評価します。ステートレスの KIE セッションを使用しており、ルールを実行することで、アジェンダで次に出てくるルールに影響を与えないようにするには、このプロパティーを有効にします。

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 コンテナー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
name	すべての名前	**KieContainer** から **KieSession** を取得する名前を定義します。**この属性は必須です。**
type	**stateful**、**stateless** デフォルト値: **stateful**	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 (**stateful**) するか、破棄 (**stateless**) するかを指定します。**stateful** 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、**stateless** 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。**stateless** 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。**stateless** 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。
default	**true**、**false** デフォルト値: **false**	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 コンテナー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。
clockType	**realtime**、**pseudo** デフォルト値: **realtime**	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。

属性	サポートされている値	説明
beliefSystem	**simple**、**jtms**、**defeasible** デフォルト値: **simple**	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 KIE 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

- Red Hat Process Automation Manager ビジネスアセットを含む Maven プロジェクトがある。
- KIE 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

1. 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>${rhpam.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: 7.33.0.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がするのに必要です。

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 Process 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 リポジトリー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>7.7.0.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. (オプション) プロジェクトに Decision Model and Notation (DMN) アセットが含まれる場合には、**pom.xml** ファイルに以下の依存関係を追加して、DMN 実行可能モデルを有効にしてください。DMN 実行可能モデルを使用すると、DMN プロジェクトの DMN デシジョンテーブルロジックがより効果的に評価できるようになります。

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-core</artifactId>
  <scope>provided</scope>
  <version>${rhpam.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 コンテナーおよび 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 コンテナーは、**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 コンテナー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.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public void testApp() {

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

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

}
```

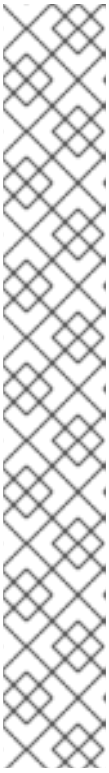
5. コマンドターミナルで Maven プロジェクトディレクトリーに移動して、以下のコマンドを実行し、プロジェクトをビルドします。

```
mvn clean install
```

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検証を行います。



注記

デフォルトで Maven プロジェクト内のルールアセットが実行可能なルールモデルからビルドされない場合には、以下の依存関係がプロジェクトの **pom.xml** ファイルに含まれているか確認して、プロジェクトを再構築してください。

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

この依存関係は、デフォルトで Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるようにするために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

実行可能なルールモデルの詳細は、「実行可能ルールモデル」を参照してください。

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 リポジトリーにデプロイします。

```
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 KIE 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

- 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。
- KIE 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

1. (オプション) プロジェクトに Decision Model and Notation (DMN) アセットが含まれる場合には、Java プロジェクトの該当のクラスパスに、以下の依存関係を追加して、DMN 実行可能モデルを有効にしてください。DMN 実行可能モデルを使用すると、DMN プロジェクトの DMN デシジョンテーブルロジックがより効果的に評価できるようになります。

```
<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: 7.33.0.Final-redhat-00003)。

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 Process 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 リポジトリー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>7.7.0.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());
```

4. 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** プロパティーを **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));
```

5. **buildAll()** メソッド **KieBuilder** を併用して**KieFileSystem** のコンテンツをビルドし、KIE コンテナーを作成してデプロイします。

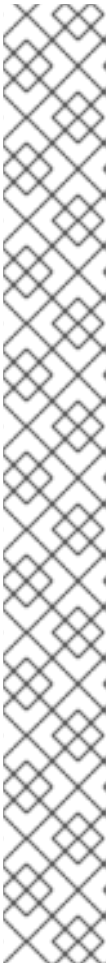
```
import org.kie.api.KieServices;
import org.kie.api.KieServices.Factory;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieBuilder;
import org.kie.api.runtime.KieContainer;

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

KieBuilder kieBuilder = ks.newKieBuilder( kfs );
kieBuilder.buildAll()
assertEquals(0, kieBuilder.getResults().getMessages(Message.Level.ERROR).size());

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

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と、ビルドプロセスの詳細を示しています。



注記

実行可能なルールモデルから Java アプリケーションプロジェクトでルールアセットをビルドするには、プロジェクトの **pom.xml** ファイルに以下の依存関係があることを確認します。

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

この依存関係は、Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるようにするために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

依存関係を確認したあとに、以下のように変更した **buildAll()** オプションを使用して実行可能なモデルを有効にします。

```
kieBuilder.buildAll(ExecutableModelProject.class)
```

実行可能なルールモデルの詳細は、「実行可能ルールモデル」を参照してください。

3.4. 実行可能ルールモデル

Red Hat Process Automation Manager のルールアセットは、標準の **kie-maven-plugin** を使用して、デフォルトで実行可能なルールモデルからビルドされます。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 ベースの表記を提供します。実行可能モデルは、以前の Red Hat Process Automation Manager バージョンの標準アセットパッケージングの代わるもので、より効率的です。KIE コンテナーと KIE ベースの作成をより迅速に実行でき、DRL (Drools Rule Language) ファイルや他の Red Hat Process Automation Manager アセットの一覧のサイズが大きい場合にとくに効果的です。このモデルは詳細レベルにわたり、インデックス評価の lambda 表記など、必要な実行情報すべてを提供します。

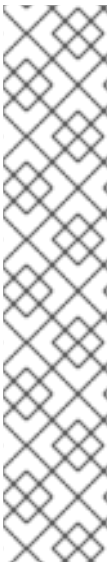
kie-maven-plugin プラグインを使用しない場合や、プロジェクト内に、必要な **drools-model-compiler** の依存関係がない場合には、ルールアセットは実行可能なモデルなしでビルドされます。

実行可能なルールモデルでは、プロジェクトにとって具体的に以下のような利点があります。

- **コンパイル時間:** 従来のパッケージ化された Red Hat Process Automation Manager プロジェクト (KJAR) には、制限や結果を実装する事前生成済みのクラスと合わせて、ルールベースを定義する DRL ファイルのリストやその他の Red Hat Process Automation Manager アーティファクトが含まれています。これらの DRL ファイルは、KJAR が Maven リポジトリーからダウンロードされて、KIE コンテナーにインストールされた時点で、解析してコンパイルする必要があります。特に大規模なルールセットの場合など、このプロセスは時間がかかる可能性があります。実行可能なモデルでは、プロジェクト KJAR 内で、Java クラスをパッケージして、プロ

ジェクトルールベースの実行可能なモデルを実装し、はるかに早い方法で KIE コンテナーと KIE ベースをを再作成することができます。Maven プロジェクトでは、**kie-maven-plugin** プラグインを使用してコンパイルプロセス中に DRL ファイルから 実行可能なモデルソースを自動的に生成します。

- **ランタイム:** 実行可能なモデルでは、制約はすべて、Java lambda 式で定義されます。同じ lambda 式も制約評価に使用するので、**mvel** ベースの制約をバイトコードに変換するのに、解釈評価用の **mvel** 式も、Just-In-Time (JIT) プロセスも使用しません。これにより、さらに迅速で効率的なランタイムを構築できます。
- **開発時間:** 実行可能なモデルでは、DRL 形式で直接要素をエンコードしたり、DRL パーサーを対応するように変更したりする必要なく、デシジョンエンジンの新機能で開発および試行することができます。



注記

実行可能なルールモデルのクエリー定義に使用できるのは、引数最大 10 個のみです。

実行可能なルールモデルのルール結果内にある変数については、使用できるバインド変数は、最大 24 個のみとなっています (同梱されている **drools** 変数を含む)。たとえば、以下のルールの結果では、バインド変数を 25 個以上使用しているので、コンパイルエラーが発生します。

```
...
then
  $input.setNo25Count(functions.sumOf(new Object[]{$no1Count_1, $no2Count_1,
$no3Count_1, ..., $no25Count_1}).intValue());
  $input.getFirings().add("fired");
  update($input);
```

3.4.1. Red Hat Process Automation Manager プロジェクトでの実行可能ルールモデルの変更または無効化

Red Hat Process Automation Manager のルールアセットは標準の **kie-maven-plugin** プラグインを使用してデフォルトで実行可能なルールモデルからビルドされます。実行可能モデルは、Red Hat Process Automation Manager のこれまでのバージョンで使用した標準アセットパッケージよりも効率的です。ただし、必要に応じて、実行可能ルールモデルを変更または無効にして、デフォルトのモデルベースの KJAR ではなく、DRL ベースの KJAR として Red Hat Process Automation Manager プロジェクトをビルドできます。

手順

通常の方法で Red Hat Process Automation Manager プロジェクトをビルドします。ただし、プロジェクトの種類に合わせて、別のビルドオプションを用意してください。

- Maven プロジェクトの場合には、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ます。

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

<VALUE> は、3 つの値のいずれかに置き換えます。

  - **YES_WITHDRL**: (デフォルト) 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)。

- **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。
- **NO**: 実行可能なモデルは生成されません。

デフォルトの実行可能モデルの動作を無効にするビルドコマンドの例:

```
mvn clean install -DgenerateModel=NO
```

- プログラムで設定した Java アプリケーションの場合には、実行可能なモデルはデフォルトで無効になっています。KIE 仮想ファイルシステム **KieFileSystem** にルールアセットを追加して、以下の **buildAll()** メソッドのいずれかと **KieBuilder** をあわせて使用してください。
  - **buildAll()** (デフォルト) または **buildAll(DrlProject.class)**: 実行可能なモデルは生成されません。
  - **buildAll(ExecutableModelProject.class)**: 元のプロジェクトにある DRL ファイルに対応する実行可能モデルが生成されます。

実行可能モデルの動作を有効にするコードの例:

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

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

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

3.5. KIE スキャナーを使用した KIE コンテナーの監視および更新

Red Hat Process Automation Manager の KIE スキャナーは、Red Hat Process Automation Manager プロジェクトに新しい **SNAPSHOT** バージョンがないか、Maven リポジトリーを監視して、指定の KIE コンテナーにプロジェクトの最新バージョンをデプロイします。開発環境に KIE スキャナーを使用して、新規バージョンが利用できるようになった時に、Red Hat Process Automation Manager プロジェクトのデプロイメントをより効率的に管理できます。

重要

実稼働環境では、誤ってまたは予期せずにプロジェクトが更新されてしまわないように、**SNAPSHOT** のプロジェクトバージョンで、KIE スキャナーを使用しないでください。KIE スキャナーは、**SNAPSHOT** プロジェクトバージョンを使用する開発環境向けに設計されています。

ビジネスプロセスと KIE スキャナーを併用するのは避けてください。プロセスで KIE スキャナーを使用すると、予期せぬ更新が発生し、プロセスインスタンスの実行と互換性のない変更が加えられた場合に、長時間実行中のプロセスでエラーが発生する可能性があります。

前提条件

- **kie-ci.jar** ファイルを Red Hat Process Automation Manager プロジェクトのクラスパスに用意しておく。

手順

1. プロジェクト内の、該当する **.java** クラスで、以下のコード例のように、KIE スキャナーを登録して起動します。

   KIE コンテナー向けの KIE スキャナーの登録および起動

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.builder.KieScanner;

...

KieServices kieServices = KieServices.Factory.get();
ReleaseId releaseId = kieServices
  .newReleaseId("com.sample", "my-app", "1.0-SNAPSHOT");
KieContainer kContainer = kieServices.newKieContainer(releaseId);
KieScanner kScanner = kieServices.newKieScanner(kContainer);

// Start KIE scanner for polling the Maven repository every 10 seconds (10000 ms)
kScanner.start(10000L);
```

   この例では、KIE スキャナーは一定の間隔で実行されるように設定しています。KIE スキャナーの最小のポーリング間隔は 1 ミリ秒 (ms) で、最大のポーリング間隔はデータ型 **long** の最大値です。ポーリングの間隔が 0 以下の場合は、**java.lang.IllegalArgumentException: pollingInterval must be positive** エラーが発生します。また、KIE スキャナーを **scanNow()** メソッドで呼び出してオンデマンドで実行するように設定することも可能です。

   この例のプロジェクトグループ ID、アーティファクト ID およびバージョン (GAV) の設定は、**com.sample:my-app:1.0-SNAPSHOT** で設定されています。プロジェクトバージョンには、**-SNAPSHOT** のサフィックスを含めて、KIE スキャナーが指定のアーティファクトバージョンの最新ビルドを取得できるようにする必要があります。**1.0.1-SNAPSHOT** への更新など、スナップショットのプロジェクトバージョン番号を変更する場合には、KIE スキャナー設定の GAV 定義のバージョンも更新する必要があります。KIE スキャナーは、**com.sample:my-app:1.0** など、静的バージョンのプロジェクトの更新を取得しません。

2. Maven リポジトリーの **settings.xml** ファイルで、**updatePolicy** 設定を **always** に指定し、KIE スキャナーが正しく機能するようにします。

```
<profile>
  <id>guvnor-m2-repo</id>
  <repositories>
    <repository>
      <id>guvnor-m2-repo</id>
      <name>BA Repository</name>
      <url>http://localhost:8080/business-central/maven2/</url>
      <layout>default</layout>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
```

```
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
      </snapshots>
    </repository>
  </repositories>
</profile>
```

KIE スキャナーがポーリングを開始した後に、KIE スキャナーが指定の KIE コンテナーで **SNAPSHOT** プロジェクトの更新バージョンを検出した場合に、KIE スキャナーは自動的に新しいプロジェクトバージョンをダウンロードして、新規プロジェクトのインクリメンタルビルドをトリガーします。この時点以降、KIE コンテナーから作成された、新規の **KieBase** と **KieSession** オブジェクトでは、新規プロジェクトバージョンが使用されるようになります。

KIE Server API を使用して KIE スキャナーを開始または停止する方法については、『**KIE API を使用した Red Hat Process Automation Manager の操作**』を参照してください。

3.6. KIE SERVER でのサービスの起動

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、KIE Server REST API コールを使用して、KIE コンテナー (デプロイメントユニット) およびそのサービスを起動できます。KIE Server REST API を使用して、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を起動できますが、Business Central からデプロイしたプロジェクトは自動的に起動するか、Business Central インターフェース内で起動できます。

前提条件

- KIE 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、KIE Server の KIE コンテナー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 コンテナー (デプロイメントユニット) の識別子。ランダムの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>:** KIE Server のホスト名 (KIE Server と同じホストでコマンドを実行する場合は **localhost**)。

- **<serverport>:** KIE 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
```

3.7. KIE SERVER で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、KIE Server REST API コールを使用して、サービスを含む KIE コンテナー (デプロイメントユニット) を停止して削除できます。KIE Server REST API を使用して、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を停止できますが、Business Central からのサービスは Business Central インターフェース内で停止できます。

前提条件

- KIE 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、KIE Server のサービスで KIE コンテナー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 コンテナー (デプロイメントユニット) の識別子。ランダムの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>:** KIE Server のホスト名 (KIE Server と同じホストでコマンドを実行する場合は **localhost**)。
- **<serverport>:** KIE 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」
- 『KIE API を使用した Red Hat Process Automation Manager の操作』
- 『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』
- 『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』
- 『Red Hat OpenShift Container Platform への Red Hat Process Automation Manager フリーフォーム管理サーバー環境のデプロイ』
- 『Operator を使用した Red Hat OpenShift Container Platform への Red Hat Process Automation Manager 環境のデプロイメント』
- 『Red Hat OpenShift Container Platform への Red Hat Process Automation Manager 試用環境のデプロイ』

付録A バージョン情報

本ドキュメントの最終更新日: 2020 年 3 月 18 日 (水)