Red Hat Decision Manager 7.1

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

Last Updated: 2019-04-01

Red Hat Decision Manager 7.1 Red Hat Decision 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 Decision Manager 7.1 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 Decision Manager に作成したサービスの使用を開始するために、開発したプロジェクトを Decision 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、Decision 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、Decision Central のプロジェクトを開発して、Decision Server REST API を使用してデプロイしたり、Decision Central で設定した Maven にプロジェクトを開発して、Decision 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。Decision 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 DECISION MANAGER プロジェクトのパッケージ化

プロジェクトには、Red Hat Decision Manager で開発するビジネスアセットが含まれます。Red Hat Decision 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や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Decision 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。

Decision 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。Decision 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 DECISION CENTRAL でのプロジェクトデプロイメント

Decision 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 Decision Server を管理できます。プロジェクトを開発する際に、Decision Central にプロジェクトをビルドして、Decision 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、Decision Central には Maven リポジトリーが組み込まれています。Decision 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 Decision Server を同じ Decision 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

Decision Central を使用して、で聖ジョンサーバー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. DECISION CENTRAL に接続するように DECISION SERVER の設定

Red Hat Decision Manager 環境で Decision Serverがすでに設定されていない場合や、Red Hat Decision Manager 環境に追加の Decision Server が必要な場合には、Decision Server が Decision Central に接続するように設定する必要があります。

注記

Red Hat OpenShift Container Platform に Decision Server をデプロイする場合は、**『Red Hat OpenShift Container Platform への Red Hat Decision 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

Decision Central および Decision Server がインストールされている。詳細は、**RED HAT DECISION MANAGER インストールのプラニング** を参照してください。

手順

1. Red Hat Decision 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 Decision 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:** Decision Central にログインするユーザーのユーザー名。
   - **org.kie.server.controller.pwd:** Decision Central にログインするユーザーのパスワード。
   - **org.kie.server.controller:** Decision Central の API に接続する URL。通常は **http://<centralhost>:<centralport>/decision-central/rest/controller** (**<centralhost>** および **<centralport>** はそれぞれ Decision Central のホスト名およびポート) になります。Decision Central を OpenShift にデプロイしている場合は、URL から **decision-central/** を削除します。
   - **org.kie.server.location:** Decision 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 Decision 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 Decision Central に存在しない場合は、Decision Server が Decision 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/decision-
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. Decision Server を起動または再起動します。

2.2. DECISION CENTRAL および DECISION SERVER への外部 MAVEN リポジトリーの設定

組み込みリポジトリーの代わりに、Nexus などの外部 Maven リポジトリーを使用するように Decision Central および Decision Server を設定できます。この場合、Decision Central でプロジェクトをビルドする際に、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ー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ーからこのファイルを処理し、Decision Central または Decision 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

Decision Central および Decision Server がインストールされている。詳細は、**RED HAT DECISION MANAGER インストールのプラニング** 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 Decision 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 Decision 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。
例:

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

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

2.3. DECISION 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、Decision Central でプロジェクトをビルドして、設定した Decision 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、Decision 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. Decision 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. 右上にある **Deploy** をクリックして、プロジェクトをビルドしてデプロイします。

注記

Decision Server にプロジェクトをデプロイせずにコンパイルするには、**Build** をクリックします。

Decision Central に Decision Server を 1 つだけ接続する場合、または接続したすべての Decision 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 Decision 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:** Decision 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。

- **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。

2.4. DECISION 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 Decision Server 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** を介してランタイム時に使用されます。Decision 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、Decision 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、Decision Central に設定した既存または新しい Decision 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。

2.4.1. Decision Central でのデプロイメントユニットの作成

お使いの Red Hat Decision 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、Decision 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 Decision Central にビルドされている。

手順

1. Decision 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** をクリックして新しい設定を作成します。
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 Decision 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。

2.4.2. Decision 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。Decision Central に Decision Server を 1 つだけ接続する場合、または接続したすべての Decision 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 Decision 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. Decision 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。

2.5. DECISION 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。Decision Central と Decision 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、Decision Central と Decision 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

開発目的の場合に限り、プロジェクトバージョンに **SNAPSHOT** 接尾辞を追加して、Maven ポリシーに従って新しいスナップショットの更新を取得できるようにすることもできます。実稼働環境に **SNAPSHOT** 接尾辞を使用しないでください。

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

手順

1. Decision 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

開発目的の場合に限り、プロジェクトバージョンに **SNAPSHOT** 接尾辞を追加して、Maven ポリシーに従って新しいスナップショットの更新を取得できるようにすることもできます。実稼働環境に **SNAPSHOT** 接尾辞を使用しないでください。

4. **Save** をクリックして終了します。

2.6. DECISION CENTRAL における重複した GAV の検出

Decision 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。

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** ファイルを保存します。
- プロジェクトのインストール、ビルドまたはデプロイ。

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>** and **<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。

2.6.1. Decision Central における重複した GAV 検出設定の管理

admin ロールを持つ Decision 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。

手順

1. Decision 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 Decision 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 DECISION CENTRAL を使用しないプロジェクトデプロイメント

Decision 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 Decision 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 Decision Server にデプロイします。Decision 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 Decision Manager 機能を使用するビジネスアセットを開発できます。

Decision 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。Decision 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 Decision 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 Decision Manager のルール、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。

Decision 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** 例で、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();
```

kmodule.xml ファイルの詳細は、**Red Hat Decision 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 Decision 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 Decision Manager のルール、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
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。
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 Decision 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** に設定した場合は、**equals()** 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 **FactHandle** が作成されます。

属性	サポートされている値	説明
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。

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。**

属性	サポートされている値	説明
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、ワーキング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。
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 DECISION MANAGER プロジェクトの MAVEN でのパッケージ化およびデプロイ

Decision Central 以外の Maven プロジェクトを、設定した Decision 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 Decision Manager ビジネスアセットを含む Maven 化されたプロジェクトがあること

- Decision 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 Decision 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>${rhdm.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 Decision Manager の Maven アーティファクトのバージョン (例: 7.11.0.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 Decision 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>7.1.0.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 Decision Manager and the Maven library version? を参照してください。

2. 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 Decision 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Red Hat Decision 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>${rhdm.version}</version>
</dependency>

<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhdm.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 Decision 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 コンテナーと 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 Decision Manager アセットが多い場合は、特に有効です。

実行可能ルールについての詳細は、**Designing a decision service using DRL rules** の "Executable rule models" を参照してください。

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 Decision 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.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
```

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 リポジトリーにデプロイします。

```
mvn deploy
```

3.3. RED HAT DECISION 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 Decision 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 Decision Manager ビジネスアセットを含む Java アプリケーションがあること
- Decision 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 Decision Manager インストールの計画』**を参照してください。

手順

1. 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 Decision 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Red Hat Decision 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>${rhdm.version}</version>
</dependency>

<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhdm.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 Decision 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 コンテナーと 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 Decision Manager アセットが多い場合は、特に有効です。

   実行可能ルールについての詳細は、**Designing a decision service using DRL rules** の "Executable rule models" を参照してください。

<version> は、プロジェクトで現在使用する Red Hat Decision Manager の Maven アーティファクトバージョンです (例: 7.11.0.Final-redhat-00002)。

注記

個別の依存関係に対して Red Hat Decision 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>7.1.0.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 Decision 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 Decision 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. 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 コンテナーを作成して、デプロイします。

```
2import 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と、ビルドプロセスの詳細を示しています。

3.4. DECISION SERVER でのサービスの起動

Decision Central 以外の Maven または Java プロジェクトから Red Hat Decision Manager アセットをデプロイした場合は、Decision 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。Decision Server REST API を使用して、デプロイメントの種類 (Decision Central からのデプロイメントを含む) に拘わらず、サービスを起動できますが、Decision 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、Decision Central インターフェース内で起動できます。

前提条件

Decision 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 Decision 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、Decision 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>:** Decision Server のホスト名 (Decision Server と同じホストでコマンドを実行する場合は **localhost**)。

- **<serverport>:** Decision Server のポート番号。

例:

```
curl --user "rhdmAdmin: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. DECISION SERVER でのサービスの停止および削除

Decision Central 以外の Maven または Java プロジェクトから Red Hat Decision Manager サービスを起動した場合は、Decision 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。Decision Server REST API を使用して、デプロイメントの種類 (Decision 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、Decision Central からのサービスは Decision Central インターフェース内で停止できます。

前提条件

Decision 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 Decision 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、Decision 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>:** Decision Server のホスト名 (Decision Server と同じホストでコマンドを実行する場合は **`localhost`**)。
- **<serverport>:** Decision Server のポート番号。

例:

```
curl --user "rhdmAdmin: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 Decision Manager 管理サーバー環境のデプロイメント』**

付録A バージョン情報

本書の最終更新日: 2018 年 11 月 1 日