# Red Hat Decision Manager 7.2

## Decision Server の管理とモニターリング

ガイド

Last Updated: 2023-02-21

# Red Hat Decision Manager 7.2 Decision Server の管理とモニターリング

ガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

## 法律上の通知



## 概要

本書では、Red Hat Decision Manager 7.2 をインストール、設定し、パフォーマンスのチューニングをする方法について説明します。

# 目次

**はじめに** ..... 3

**第1章 RED HAT DECISION MANAGER のコンポーネント** ..... 4

**第2章 MAVEN を使用したシステム統合** ..... 5
2.1. ローカルプロジェクトの PREEMPTIVE (先行) 認証 5
2.2. DECISION CENTRAL における重複した GAV の検出 6
2.3. DECISION CENTRAL における重複した GAV 検出設定の管理 6

**第3章 RED HAT DECISION MANAGER へのパッチ更新およびマイナーリリースアップグレードの適用** ..... 7

**第4章 DECISION SERVER の設定と起動** ..... 11

**第5章 統合 DECISION MANAGER コントローラー での DECISION SERVER の設定** ..... 13

**第6章 ヘッドレス DECISION MANAGER コントローラーのインストールおよび実行** ..... 15
6.1. DECISION MANAGER コントローラーを使った DECISION SERVER のインストーラーによる設定 15
6.2. ヘッドレス DECISION MANAGER コントローラーのインストール 16
6.2.1. ヘッドレス Decision Manager コントローラーの作成 17
6.2.2. Decision Server とヘッドレス Decision Manager コントローラー の設定 17
6.3. ヘッドレス DECISION MANAGER コントローラーの実行 18
6.4. ヘッドレス DECISION MANAGER コントローラーを使用したクラスタリング 20

**第7章 DECISION CENTRAL に接続する DECISION SERVER の設定** ..... 22

**第8章 DECISION CENTRAL により管理される DECISION SERVER の設定** ..... 24
8.1. TLS 対応の SMART ROUTER の設定 26

**第9章 管理対象 DECISION SERVER** ..... 27

**第10章 非管理対象 DECISION SERVER** ..... 28

**第11章 デプロイメント記述子** ..... 29
11.1. デプロイメント記述子の設定 29
設定内容 30
11.2. デプロイメント記述子の管理 31
11.3. ランタイムエンジンへのアクセス制限 31

**第12章 OPENSHIFT 接続タイムアウトの設定** ..... 32

**第13章 LDAP ログインドメインの定義** ..... 33

**第14章 RH-SSO を使用したサードパーティークライアントの認証** ..... 34
14.1. BASIC 認証 34

**第15章 DECISION SERVER のシステムプロパティー** ..... 35

**第16章 DECISION SERVER の機能と拡張** ..... 41
16.1. カスタム REST API エンドポイントを使用した既存の DECISION SERVER 機能の拡張 42
16.2. カスタムデータトランスポートを使用するための DECISION SERVER の拡張 48
16.3. カスタムクライアント API を使用した DECISION SERVER のクライアント拡張 55

**第17章 関連情報** ..... 60

**付録A バージョン情報** ..... 61

# はじめに

システム管理者は、Red Hat Decision Manager を実稼働環境にインストール、設定、アップグレードし、システム障害にすばやくかつ容易に対応できるようになり、システムが最適に稼働するようにできます。

**前提条件**

- Red Hat JBoss Enterprise Application Platform 7.2 がインストールされている。詳細は、**Red Hat JBoss EAP 7.2 インストールガイド**を参照してください。
- Red Hat Decision Manager がインストールされている。詳細は、**Red Hat Decision Manager インストールの計画** を参照してください。
- Red Hat Decision Manager が実行していて、**admin** ロールで Decision Central にログインできる。詳細は、**Red Hat Decision Manager インストールの計画** を参照してください。

# 第1章 RED HAT DECISION MANAGER のコンポーネント

Red Hat Decision Manager は、Decision Central と Decision Server で設定されます。

- Decision Central は、ビジネスルールを作成および管理するためのグラフィカルインターフェイスです。Decision Central は、Red Hat JBoss EAP インスタンスまたは Red Hat OpenShift Container Platform (OpenShift) にインストールできます。
  Decision Central は、スタンドアロンの JAR ファイルとしても使用できます。Decision Central スタンドアロンの JAR ファイルとして使用して、アプリケーションサーバーにデプロイせずに Decision Central を実行できます。

- Decision Server では、プロセス、ルール、およびその他のアーティファクトが実行されます。これは、プロセスとルールをインスタンス化して実行し、計画の問題を解決するために使用されます。Decision Server は、Red Hat JBoss EAP インスタンス、OpenShift、Oracle WebLogic Server インスタンス、IBM WebSphere Application Server インスタンスに、または Spring Boot アプリケーションの一部としてインストールできます。
  Decision Server は、管理モードまたは非管理モードで動作するように設定できます。非管理モードの場合は、手動で KIE コンテナー (デプロイメントユニット) を作成および維持する必要があります。KIE コンテナーは、プロジェクトの特定のバージョンです。管理モードの場合は、Decision Manager コントローラーが Decision Server の設定を管理し、ユーザーはコントローラーと対話形式で KIE コンテナーを作成、維持します。

# 第2章 MAVEN を使用したシステム統合

Red Hat Decision Manager は、Red Hat JBoss Middleware Maven Repository と Maven Central リポジトリーを依存関係ソースとして使用するように作られています。これら両方の依存関係がプロセスビルドに利用可能になるようにしてください。

ご自分のプロジェクトがアーティファクトの特定バージョンに依存していることを確認してください。**LATEST** または **RELEASE** は、一般的に、アプリケーションの依存関係バージョンの特定と管理に使用されます。

- **LATEST** は、アーティファクトの最新デプロイ (スナップショット) バージョンになります。
- **RELEASE** は、リポジトリー内の最新の非スナップショットバージョンリリースになります。

**LATEST** または **RELEASE** を使用することで、サードパーティーのライブラリーの新リリース時にバージョン番号を更新する必要がなくなります。ただし、ソフトウェアリリースに影響を受けるビルドに対するコントロールができなくなることになります。

## 2.1. ローカルプロジェクトの PREEMPTIVE (先行) 認証

お使いの環境にインターネットアクセスがない場合には、Maven Central や他のパブリックリポジトリーの代わりに社内リポジトリーを設定します。Red Hat Decision Manager サーバーのリモート Maven リポジトリーからローカル Mave プロジェクトに JAR をインポートするには、リポジトリーサーバーの先行認証をオンにします。**pom.xml** ファイルの **guvnor-m2-repo** 用の認証を設定することでこれが実行できます。以下に例を示します。

```
<server>
  <id>guvnor-m2-repo</id>
  <username>admin</username>
  <password>admin</password>
  <configuration>
    <wagonProvider>httpclient</wagonProvider>
     <httpConfiguration>
       <all>
         <usePreemptive>true</usePreemptive>
       </all>
     </httpConfiguration>
  </configuration>
</server>
```

別の方法では、Authorization HTTP ヘッダーを Base64 でエンコードされた認証情報で設定できます。

```
<server>
  <id>guvnor-m2-repo</id>
  <configuration>
   <httpHeaders>
      <property>
        <name>Authorization</name>
        <!-- Base64-encoded "admin:admin" -->
        <value>Basic YWRtaW46YWRtaW4=</value>
      </property>
    </httpHeaders>
  </configuration>
</server>
```

## 2.2. 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、プロジェクトの **GroupId**、**ArtifactId**、および **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。

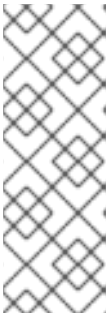
- **pom.xml** ファイルの **<repositories>** 要素および **<distributionManagement>** 要素で指定されたリポジトリー。
- Maven の **settings.xml** 設定ファイルに指定されたリポジトリー。

## 2.3. 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、および **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。

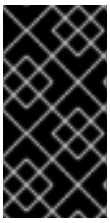
```
$ ~/EAP_HOME/bin/standalone.sh -c standalone-full.xml
-Dorg.guvnor.project.gav.check.disabled=true
```

# 第3章 RED HAT DECISION MANAGER へのパッチ更新およびマイナーリリースアップグレードの適用

大抵の場合は、Decision Central、Decision Server、ヘッドレス Decision Manager コントローラーなど、Red Hat Decision Manager の特定コンポーネントの更新を容易にする自動更新ツールが Red Hat Decision Manager のパッチ更新と新規マイナーバージョンで提供されます。デシジョンエンジンやスタンドアロンの Decision Central など、その他の Red Hat Decision Manager アーティファクトは、各マイナーリリースが含まれる新しいアーティファクトとしてリリースされるため、再インストールして更新を適用する必要があります。

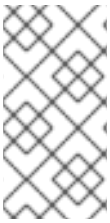
この自動更新ツールを使用してパッチ更新とマイナーリリースアップグレードの両方を Red Hat Decision Manager 7.2 に適用することができます。バージョン 7.2 から 7.2.1 への更新といった Red Hat Decision Manager のパッチ更新には、最新のセキュリティー更新とバグ修正が含まれます。バージョン 7.2.x から 7.3 へのアップグレードといった Red Hat Decision Manager のマイナーリリースアップグレードには、機能強化、セキュリティー更新、バグ修正が含まれます。

新たなマイナーリリースにアップグレードする前に、お使いの Red Hat Decision Manager に最新のパッチ更新を適用してください。



**重要**

Red Hat Decision Manager 7.1 から 7.2 にアップグレードするには、最初に Red Hat Decision Manager 7.1.1 (最新のパッチ更新) に更新してから、この手順を再度実行して Red Hat Decision Manager 7.2 にアップグレードします。必要に応じて、このセクションに示されているアップグレードバージョンの例を調整してください。



**注記**

Red Hat Decision Manager 更新ツールに含まれるのは、Red Hat Decision Manager の更新のみです。Red Hat JBoss EAP への更新は、Red Hat JBoss EAP パッチ配信を使用して適用する必要があります。詳細は、**Red Hat JBoss EAP パッチおよびアップグレードガイド** を参照してください。

**前提条件**

- Red Hat Decision Manager インスタンスおよび Decision Server インスタンスが稼働していない。Red Hat Decision Manager インスタンスまたは Decision Server インスタンスが実行している間は更新を適用しないでください。

**手順**

1. Red Hat カスタマーポータルの Software Downloads ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
   以下に例を示します。

   - **Product:** Decision Manager
   - **Version:** 7.2.1

   バージョン 7.2.x から 7.3 などのように Red Hat Decision Manager のマイナーリリースにアップグレードする場合は、お使いの Red Hat Decision Manager に最新のパッチ更新を適用してから、以下の手順に従って、新たなマイナーリリースにアップグレードしてください。

2. **Patches** をクリックし、**Red Hat Decision Manager [VERSION] Update Tool**をダウンロードし、ダウンロードした **rhdm-$VERSION-update.zip** ファイルを一時ディレクトリーに展開します。
この更新ツールは、Decision Central、Decision Server、ヘッドレス Decision Manager コントローラーなど、Red Hat Decision Manager の一定のコンポーネントの更新を自動化します。このツールを使用して最初に更新を適用し、Red Hat Decision Manager ディストリビューションに関連するその他の更新、または新しいリリースアーティファクトをインストールします。

3. 更新ツールがファイルを更新しないようにするには、展開した **rhdm-$VERSION-update** ディレクトリーに移動し、**blacklist.txt** ファイルを開き、更新しないファイルの相対パスを追加します。
ファイルが **blacklist.txt** ファイルの一覧に追加されていると、更新スクリプトは、そのファイルを新しいバージョンに置き換えずにそのまま残し、新しいバージョンのファイルに **.new** 接尾辞を付けて追加します。ブラックリストのファイルが配布されなくなると、更新ツールは、**.removed** 接尾辞の付いた、空のマーカーファイルを作成します。次に、これらの新しいファイルを手動で保持、マージ、または削除することを選択できます。

   **blacklist.txt** ファイルで除外されるファイルの例:

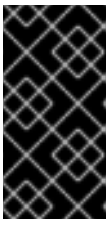
```
WEB-INF/web.xml  // Custom file
styles/base.css  // Obsolete custom file kept for record
```

   更新後の、ブラックリストに指定されたファイルディレクトリー内のコンテンツ:

```
$ ls WEB-INF
web.xml web.xml.new
```

```
$ ls styles
base.css base.css.removed
```

4. コマンドの端末で、**rhdm-$VERSION-update.zip** ファイルから展開した一時ファイルに移動し、以下の形式で **apply-updates** スクリプトを実行します。



**重要**

更新を適用する前に、Red Hat Decision Manager インスタンスおよび Decision Server インスタンスが実行していないことを確認します。Red Hat Decision Manager インスタンスまたは Decision Server インスタンスが実行している間は更新を適用しないでください。

   Linux システムまたは Unix ベースのシステムの場合:

```
$ ./apply-updates.sh $DISTRO_PATH $DISTRO_TYPE
```

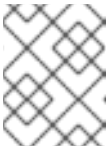
   Windows の場合:

```
$ .\apply-updates.bat $DISTRO_PATH $DISTRO_TYPE
```

   **$DISTRO_PATH** の部分は、関連するディストリビューションディレクトリーへのパスで、**$DISTRO_TYPE** の部分は、更新しているディストリビューションの種類となります。

   Red Hat Decision Manager 更新ツールでは、以下のディストリビューションの種類がサポートされます。

- **rhdm-decision-central-eap7-deployable**: Decision Central を更新します (**decision-central.war**)。
- **rhdm-kie-server-ee8**: Decision Server を更新します (**kie-server.war**)



**注記**

この更新ツールで、Red Hat JBoss EAP EE7 から Red Hat JBoss EAP EE8 に更新されます。

- **rhdm-kie-server-jws**: Red Hat JBoss Web Server で Decision Server を更新します ( **kie-server.war**)
- **rhdm-controller-ee7**: ヘッドレス Decision Manager コントローラー (**controller.war**) を更新します。
- **rhdm-controller-jws**: Red Hat JBoss Web Server でヘッドレスの Decision Manager コントローラーを更新します (**controller.war**)。

Red Hat JBoss EAP で、Red Hat Decision Manager の完全ディストリビューションに対する Decision Central および Decision Server への更新の例:

```
./apply-updates.sh ~EAP_HOME/standalone/deployments/decision-central.war rhdm-decision-central-eap7-deployable

./apply-updates.sh ~EAP_HOME/standalone/deployments/kie-server.war rhdm-kie-server-ee7
```

ヘッドレス Decision Manager コントローラーへの更新例 (使用している場合):

```
./apply-updates.sh ~EAP_HOME/standalone/deployments/controller.war rhdm-controller-ee7
```

更新スクリプトは、展開した **rhdm-$VERSION-update** ディレクトリーに、指定したディストリビューションのコピーを含む **backup** ディレクトリーを作成してから、更新を行います。

5. 更新ツールが完了したら、Red Hat カスタマーポータルで、更新ツールをダウンロードした**Software Downloads**ページに戻り、Red Hat Decision Manager ディストリビューションに関するその他の更新または新しいリリースアーティファクトをインストールします。デシジョンエンジンまたはその他のアドオンに関する **.jar** など、Red Hat Decision Manager ディストリビューションにすでに存在しているファイルについては、ファイルの既存のバージョンを、Red Hat カスタマーポータルから取得した新しいバージョンと取り替えます。
6. エアギャップ環境など、スタンドアロンの **Red Hat Decision Manager 7.2.0 Maven Repository** アーティファクト (**rhdm-7.2.0-maven-repository.zip**) を使用している場合は、**Red Hat Decision Manager [VERSION] Incremental Maven Repository**をダウンロードし、ダウンロードした **rhdm-$VERSION-incremental-maven-repository.zip** ファイルを既存の **~/maven-repository** ディレクトリーに展開して、関連するコンテンツを更新します。Maven リポジトリーの更新例:

```
$ unzip -o rhdm-7.2.1-incremental-maven-repository.zip -d $REPO_PATH/rhdm-7.2.0-maven-repository/maven-repository/
```

7. 関連する更新をすべて適用したら、Red Hat Decision Manager および Decision Server を起動して、Decision Central にログインします。

8. Decision Central 内のすべてのプロジェクトデータが存在して正確であることを確認し、Decision Central ウィンドウの右上隅でプロファイル名をクリックし、**About** をクリックして、更新した製品バージョン番号を確認します。
   Decision Central でエラーが発生したり、データが不足していることが通知されたら、**rhdm-$VERSION-update** ディレクトリーの **backup** ディレクトリーにコンテンツを復元し、更新ツールへの変更を戻します。Red Hat カスタマーポータルで Red Hat Decision Manager の以前のバージョンから、関連するリリースアーティファクトを再インストールできます。以前のディストリビューションを復元したら、更新を再実行してください。

# 第4章 DECISION SERVER の設定と起動

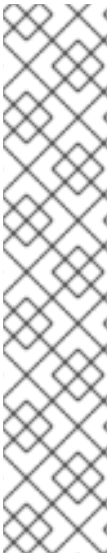
Decision Server の場所、ユーザー名、パスワード、その他の関連プロパティーは、Decision Server の起動時に必要な設定を定義することで設定できます。

### 手順

Red Hat Decision Manager 7.2 の **bin** ディレクトリーに移動し、以下のプロパティーで新しい Decision Server を起動します。お使いの環境に応じて特定のプロパティーを調整します。

```
$ ~/EAP_HOME/bin/standalone.sh --server-config=standalone-full.xml (1)
-Dorg.kie.server.id=myserver (2)
-Dorg.kie.server.user=decision_server_username (3)
-Dorg.kie.server.pwd=decision_server_password (4)
-Dorg.kie.server.controller=http://localhost:8080/decision-central/rest/controller (5)
-Dorg.kie.server.controller.user=controller_username (6)
-Dorg.kie.server.controller.pwd=controller_password (7)
-Dorg.kie.server.location=http://localhost:8080/kie-server/services/rest/server (8)
-Dorg.kie.server.persistence.dialect=org.hibernate.dialect.PostgreSQLDialect (9)
-Dorg.kie.server.persistence.ds=java:jboss/datasources/psjbpmDS (10)
```

1. **standalone-full.xml** サーバープロファイルの開始コマンド
2. サーバー ID (Decision Central で定義したサーバー設定名に一致させる必要がある)
3. Decision Manager コントローラー から Decision Server に接続するユーザー名
4. Decision Manager コントローラー から Decision Server に接続するパスワード
5. Decision Manager コントローラーの場所 (**/rest/controller** 接尾辞が付いた Decision Central の URL)
6. Decision Manager コントローラー REST API に接続するユーザー名
7. Decision Manager コントローラー REST API に接続するパスワード
8. Decision Server の場所 (この例では Decision Central と同じ場所)
9. 使用する Hibernate の方言
10. 以前の Red Hat JBoss BRMS データベースに使用されるデータソースの JNDI 名



**注記**

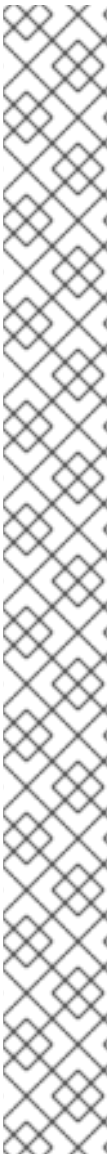
Decision Central と Decision Server が別々のアプリケーションサーバーインスタンス (Red Hat JBoss EAP など) にインストールされている場合は、Decision Central とポートが競合しないように、DecisionServer の場所には別のポートを使用します。別の Decision Server ポートが設定されていない場合は、ポートオフセットを追加して、Decision Server プロパティーに従って Decision Server のポート値を調整します。

以下に例を示します。

```
-Djboss.socket.binding.port-offset=150
-Dorg.kie.server.location=http://localhost:8230/kie-server/services/rest/server
```

この例のように、Decision Central ポートが 8080 の場合は、定義したオフセットが 150 の Decision Server ポートは 8230 です。

Decision Server は、新しい Decision Central に接続し、デプロイするデプロイメントユニット (KIE コンテナー) の一覧を収集します。



**注記**

依存関係の JAR ファイルでクラスを使用して Decision Server クライアントから Decision Server にアクセスすると、Decision Central では **ConversionException** および **ForbiddenClassException** が発生します。Decision Central でこれらの例外を発生させないようにするには、次のいずれかを実行します。

- クライアント側で例外が発生する場合は、kie-server クライアントに次のシステムプロパティーを追加します。

```
System.setProperty("org.kie.server.xstream.enabled.packages", "org.example.**");
```

- サーバー側で例外が発生する場合は、Red Hat Decision Manager インストールディレクトリーから **standalone-full.xml** を開き、<system-properties> タグに以下のプロパティーを設定します。

```
<property name="org.kie.server.xstream.enabled.packages" value="org.example.**"/>
```

- 以下の JVM プロパティーを設定します。

```
-Dorg.kie.server.xstream.enabled.packages=org.example.**
```

KJAR に存在するクラスは、これらのシステムプロパティーを使用して設定しないように想定されています。システムプロパティーでは既知のクラスのみを使用し、脆弱性を回避するようにしてください。

**org.example** はパッケージ例で、使用するパッケージを何でも定義できます。**org.example1.* * , org.example2.* * , org.example3.* *** などのように、コンマ区切りで、複数のパッケージを指定できます。

**org.example1.Mydata1, org.example2.Mydata2** など、特定のクラスも追加できます。

# 第5章 統合 DECISION MANAGER コントローラー での DECISION SERVER の設定

**注記**

本セクションの変更は、Decision Server を Decision Central で管理し、Red Hat Decision Manager を ZIP ファイルからインストールしている場合にのみ、実行してください。Decision Central をインストールしていない場合は、*6章 ヘッドレス Decision Manager コントローラーのインストールおよび実行* の記載どおりに、ヘッドレス Decision Manager コントローラーを使用して Decision Server を管理することができます。

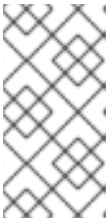
Decision Server は、管理モードまたは非管理モードで動作できます。非管理モードの場合は、手動で KIE コンテナー (デプロイメントユニット) を作成および維持する必要があります。管理モードの場合は、Decision Manager コントローラーが Decision Server の設定を管理し、ユーザーはコントローラーと対話形式で KIE コンテナーを作成、維持します。

Decision Manager コントローラーは Decision Central と統合します。Decision Central をインストールしている場合は、Decision Central の **Execution Server** ページを使用して Decision Manager コントローラーと対話します。

ZIP ファイルから Red Hat Decision Manager をインストールした場合は、Decision Server および Decision Central の両方のインストールの **standalone-full.xml** ファイルを編集して、統合 Decision Manager コントローラーで Decision Server を設定する必要があります。

**前提条件**

- Decision Central と Decision Server が Red Hat JBoss EAP インストールのベースディレクトリー (**EAP_HOME**) にインストールされている。



**注記**

Decision Central と Decision Server は、実稼働環境では異なるサーバーにインストールすることが推奨されます。ただし、たとえば開発環境で、Decision Server と Decision Central を同じサーバーにインストールする場合は、本セクションの説明に従って、共有の **standalone-full.xml** ファイルを変更します。

- Decision Central サーバーノードに、**rest-all** ロールを持つユーザーが作成されている。

**手順**

1. Decision Central の **EAP_HOME/standalone/configuration/standalone-full.xml** ファイルにおいて、**<system-properties>** セクションの以下のプロパティーのコメントを解除し、**<USERNAME>** および **<USER_PWD>** を **kie-server** ロールを持つユーザーの認証情報に置き換えます。

```
<property name="org.kie.server.user" value="<USERNAME>"/>
<property name="org.kie.server.pwd" value="<USER_PWD>"/>
```

2. Decision Server の **EAP_HOME/standalone/configuration/standalone-full.xml** ファイルで、**<system-properties>** セクションの以下のプロパティーのコメントを解除します。

```
<property name="org.kie.server.controller.user" value="<CONTROLLER_USER>"/>
```

```
  <property name="org.kie.server.controller.pwd" value="<CONTROLLER_PWD>"/>
  <property name="org.kie.server.id" value="<KIE_SERVER_ID>"/>
  <property name="org.kie.server.location" value="http://<HOST>:<PORT>/kie-
server/services/rest/server"/>
  <property name="org.kie.server.controller" value="<CONTROLLER_URL>"/>
```

3. 以下の値を置き換えます。

- **<CONTROLLER_USER>** および **<CONTROLLER_PWD>** を **rest-all** ロールを持つユーザーの認証情報に置き換えます。
- **<KIE_SERVER_ID>** を Decision Server システムの ID または名前に置き換えます (例: **rhdm-7.2.0-decision_server-1** )。
- **<HOST>** を Decision Server ホストの ID または名前に置き換える (例: **localhost** または **192.7.8.9**)。
- **<PORT>** を Decision Server ホストのポートに置き換える (例: **8080**)。

**注記**

**org.kie.server.location** プロパティーで Decision Server の場所を指定します。

- **<CONTROLLER_URL>** を Decision Central の URL に置き換えます。Decision Server は、起動時にこの URL に接続します。
  - インストーラーまたは Red Hat JBoss EAP zip ファイルを使用して Decision Central をインストールした場合、**<CONTROLLER_URL>** は以下のようになります。
    **http://<HOST>:<PORT>/decision-central/rest/controller**
  - **standalone.jar** ファイルを使用して Decision Central を実行している場合、**<CONTROLLER_URL>** は以下のようになります。
    **http://<HOST>:<PORT>/rest/controller**

# 第6章 ヘッドレス DECISION MANAGER コントローラーのインストールおよび実行

Decision Server は、管理モードまたは非管理モードで動作するように設定できます。非管理モードの場合は、手動で KIE コンテナー (デプロイメントユニット) を作成および維持する必要があります。管理モードの場合は、Decision Manager コントローラーが Decision Server の設定を管理し、ユーザーはコントローラーと対話形式で KIE コンテナーを作成、維持します。

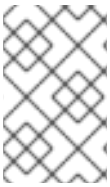
Decision Central には Decision Manager コントローラーが組み込まれています。Decision Central をインストールしている場合は、**Execution Server** ページを使用して KIE コンテナーを作成および維持します。Decision Central を使用せずに Decision Server の管理を自動化するには、ヘッドレス Decision Manager コントローラーを使用します。

## 6.1. DECISION MANAGER コントローラーを使った DECISION SERVER のインストーラーによる設定

Decision Server は、Decision Manager コントローラーで管理するか、非管理モードで動作できます。非管理モードの場合は、手動で KIE コンテナー (デプロイメントユニット) を作成および維持する必要があります。管理モードの場合は、Decision Manager コントローラーが Decision Server の設定を管理し、ユーザーはコントローラーと対話形式で KIE コンテナーを作成、維持します。

Decision Manager コントローラーは Decision Central と統合します。Decision Central をインストールしている場合は、Decision Central の **Execution Server** ページを使用して Decision Manager コントローラーと対話します。

インストーラーはインタラクティブモードまたは CLI モードで使用し、Decision Central と Decision Server をインストールして、Decision Manager コントローラーで Decision Server を設定します。



**注記**

Decision Central をインストールしない場合は、*6章 ヘッドレス Decision Manager コントローラーのインストールおよび実行* でヘッドレス Decision Manager の使用方法を参照してください。

**前提条件**

- バックアップを作成してある Red Hat JBoss EAP 7.2 以降のサーバーインストールを持つ 2 台のコンピューターが利用できる。
- インストールを完了するのに必要なユーザーパーミッションが付与されている。

**手順**

1. 1 台目のコンピューターで、インタラクティブモードまたは CLI モードでインストーラーを実行します。詳細は **Red Hat JBoss EAP への Red Hat Decision Manager のインストールおよび設定** を参照してください。
2. **Component Selection** ページで、**Decision Server** チェックボックスを外します。
3. Decision Central インストールを完了します。
4. 2 台目のコンピューターで、インタラクティブモードまたは CLI モードでインストーラーを実行します。

5. **Component Selection** ページで、**Decision Central** チェックボックスを外します。
6. **Configure Runtime Environment** ページで **Perform Advanced Configuration** を選択します。
7. **Customize Decision Server properties** を選択し、**Next** をクリックします。
8. **Process Server Properties Configuration** ページで、**New Server Configuration** をクリックして Decision Server を追加し、その Decision Server に一意の名前を指定します。この名前は Decision Central に表示され、複数の Decision Server を区別できるようになります。

## 6.2. ヘッドレス DECISION MANAGER コントローラーのインストール

ヘッドレス Decision Manager コントローラーをインストールし、REST API または Decision Server Java Client API を使用してコントローラーを操作することができます。

**前提条件**

- バックアップを作成してある Red Hat JBoss EAP システム (バージョン 7.2 またはそれ以降) が利用できる。Red Hat JBoss EAP システムのベースディレクトリーを **EAP_HOME** とする。
- インストールを完了するのに必要なユーザーパーミッションが付与されている。

**手順**

1. Red Hat カスタマーポータルの Software Downloads ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
   - **Product:** Decision Manager
   - **バージョン:** 7.2
2. **Red Hat Decision Manager 7.2.0 Add Ons**(**rhdm-7.2.0-add-ons.zip** ファイル) をダウンロードします。
3. **rhdm-7.2.0-add-ons.zip** ファイルを展開します。**rhdm-7.2-controller-ee7.zip** ファイルは展開したディレクトリーにあります。
4. **rhdm-7.2-controller-ee7.zip** アーカイブを一時ディレクトリーに展開します。以下の例では、この名前を **TEMP_DIR** とします。
5. **TEMP_DIR/rhdm-7.2-controller-ee7/controller.war** ディレクトリーを **EAP_HOME/standalone/deployments/** にコピーします。

**警告**

コピーするヘッドレス Decision Manager コントローラーデプロイメントの名前が、Red Hat JBoss EAP インスタンスの既存デプロイメントと競合しないことを確認します。

6. **TEMP_DIR/rhdm-7.2-controller-ee7/SecurityPolicy/** ディレクトリーの中身を **EAP_HOME/bin** にコピーします。ファイルの上書きを確認するメッセージが表示されたら、**Yes** を選択します。

7. **EAP_HOME/standalone/deployments/** ディレクトリーに、**controller.war.dodeploy** という名前で空のファイルを作成します。このファイルにより、サーバーが起動するとヘッドレス Decision Manager コントローラーが自動的にデプロイされます。

### 6.2.1. ヘッドレス Decision Manager コントローラーの作成

ヘッドレス Decision Manager コントローラーを使用する前に、**kie-server** ロールを持つユーザーを作成する必要があります。

**前提条件**

- ヘッドレス Decision Manager コントローラーが Red Hat JBoss EAP インストールのベースディレクトリー (**EAP_HOME**) にインストールされている。

**手順**

1. 端末アプリケーションで **EAP_HOME/bin** ディレクトリーに移動します。
2. 以下のコマンドを入力し、**<USER_NAME>** および **<PASSWORD>** を、作成するユーザー名およびパスワードに置き換えます。

```
$ ./add-user.sh -a --user <username> --password <password> --role kie-server
```

**注記**

必ず、既存のユーザー、ロール、またはグループとは異なるユーザー名を指定してください。たとえば、**admin** という名前のユーザーは作成しないでください。

パスワードは 8 文字以上で、数字と、英数字以外の文字をそれぞれ 1 文字以上使用する必要があります。ただし & の文字は使用できません。

3. ユーザー名とパスワードを書き留めておきます。

### 6.2.2. Decision Server とヘッドレス Decision Manager コントローラー の設定

Decision Server をヘッドレス Decision Manager コントローラーで管理する場合は、本セクションの説明に従って Decision Server インストールの **standalone-full.xml** とヘッドレス Decision Manager コントローラーの **standalone.xml** ファイルを編集する必要があります。

**前提条件**

- Decision Server が Red Hat JBoss EAP インストールのベースディレクトリー (**EAP_HOME**) にインストールされている。
- ヘッドレス Decision Manager コントローラーが **EAP_HOME** にインストールされている。

**注記**

実稼働環境では Decision Server およびヘッドレス Decision Manager コントローラーを異なるサーバーにインストールすることを推奨します。ただし、開発環境のように Decision Server およびヘッドレス Decision Manager コントローラーを同じサーバーにインストールする場合は、併せて共有の **standalone-full.xml** ファイルを変更します。

- Decision Server サーバーノードに、**kie-server** ロールをのあるユーザーが作成されている。
- サーバーノードに、**kie-server** ロールのあるユーザーが作成されている。

**手順**

1. **EAP_HOME/standalone/configuration/standalone-full.xml** ファイルの **<system-properties>** セクションに以下のプロパティーを追加し、**<USERNAME>** および **<USER_PWD>** を、**kie-server** ロールを持つユーザーの認証情報に置き換えます。

```
<property name="org.kie.server.user" value="<USERNAME>"/>
<property name="org.kie.server.pwd" value="<USER_PWD>"/>
```

2. Decision Server の **EAP_HOME/standalone/configuration/standalone-full.xml** ファイルの **<system-properties>** セクションに以下のプロパティーを追加します。

```
<property name="org.kie.server.controller.user" value="<CONTROLLER_USER>"/>
<property name="org.kie.server.controller.pwd" value="<CONTROLLER_PWD>"/>
<property name="org.kie.server.id" value="<KIE_SERVER_ID>"/>
<property name="org.kie.server.location" value="http://<HOST>:<PORT>/kie-server/services/rest/server"/>
<property name="org.kie.server.controller" value="<CONTROLLER_URL>"/>
```

3. このファイルで、以下の値を置き換えます。

- **<CONTROLLER_USER>** および **<CONTROLLER_PWD>** を **kie-server** ロールを持つユーザーの認証情報に置き換えます。
- **<KIE_SERVER_ID>** を Decision Server システムの ID または名前に置き換えます (例: **rhdm-7.2.0-decision_server-1**)。
- **<HOST>** を Decision Server ホストの ID または名前に置き換える (例: **localhost** または **192.7.8.9**)。
- **<PORT>** を Decision Server ホストのポートに置き換える (例: **8080**)。

**注記**

**org.kie.server.location** プロパティーで Decision Server の場所を指定します。

- **<CONTROLLER_URL>** をヘッドレス Decision Manager コントローラー の URL で置き換えます。

1. Decision Server は、起動時にこの URL に接続します。

## 6.3. ヘッドレス DECISION MANAGER コントローラーの実行

ヘッドレス Decision Manager コントローラーを Red Hat JBoss EAP にインストールしたら、以下の手順に従ってヘッドレス Decision Manager コントローラーを実行します。

**前提条件**

- ヘッドレス Decision Manager コントローラーが Red Hat JBoss EAP インストールのベースディレクトリー (**EAP_HOME**) にインストールされている。

**手順**

1. ターミナルアプリケーションで **EAP_HOME/bin** に移動します。
2. ヘッドレス Decision Manager コントローラーを、Decision Server をインストールした Red Hat JBoss EAP インスタンスと同じ Red Hat JBoss EAP インスタンスにインストールしている場合は、以下のいずれかのコマンドを実行します。
   - Linux または UNIX ベースのシステムの場合:

     ```
     $ ./standalone.sh -c standalone-full.xml
     ```

   - Windows の場合:

     ```
     standalone.bat -c standalone-full.xml
     ```

3. ヘッドレス Decision Manager コントローラーを、Decision Server をインストールした Red Hat JBoss EAP インスタンスとは別の Red Hat JBoss EAP インスタンスにインストールしている場合は、**standalone.sh** スクリプトで Decision Manager コントローラーを開始できます。

   **注記**

   この場合は、**standalone.xml** ファイルに必要な設定変更を加えます。

   - Linux または UNIX ベースのシステムの場合:

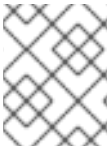
     ```
     $ ./standalone.sh
     ```

   - Windows の場合:

     ```
     standalone.bat
     ```

4. ヘッドレス Decision Manager コントローラーが Red Hat JBoss EAP 上で動作していることを確認するには、以下のコマンドを入力します。ここで、**<CONTROLLER>** と **<CONTROLLER_PWD>** は、ユーザー名とパスワードの組み合わせです。このコマンドにより、Decision Server インスタンスに関する情報が出力されます。

   ```
   curl -X GET "http://<HOST>:<PORT>/controller/rest/controller/management/servers" -H "accept: application/xml" -u '<CONTROLLER>:<CONTROLLER_PWD>'
   ```



**注記**

別の方法では、Decision Server Java API Client を使用してヘッドレス Decision Manager コントローラーにアクセスすることもできます。

## 6.4. ヘッドレス DECISION MANAGER コントローラーを使用したクラスタリング

Decision Manager コントローラーは Decision Central と統合します。ただし、Decision Central をインストールしない場合は、ヘッドレス Decision Manager コントローラーをインストールし、REST API または Decision Server Java Client API を使用してそのコントローラーと対話します。

**前提条件**

- バックアップを作成してある Red Hat JBoss EAP システム (バージョン 7.2 またはそれ以降) が利用できる。Red Hat JBoss EAP システムのベースディレクトリーを **EAP_HOME** とする。
- インストールを完了するのに必要なユーザーパーミッションが付与されている。
- マウントしたパーティションが存在する NFS サーバーが利用できる。

**手順**

1. Red Hat カスタマーポータルの Software Downloads ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
   - **Product: Decision Manager**
   - **Version: 7.2**
2. **Red Hat Decision Manager 7.2.0 Add Ons**(**rhdm-7.2.0-add-ons.zip** ファイル) をダウンロードします。
3. **rhdm-7.2.0-add-ons.zip** ファイルを展開します。**rhdm-7.2-controller-ee7.zip** ファイルは展開したディレクトリーにあります。
4. **rhdm-7.2-controller-ee7.zip** アーカイブを一時ディレクトリーに展開します。以下の例では、この名前を **TEMP_DIR** とします。
5. **TEMP_DIR/rhdm-7.2-controller-ee7/controller.war** ディレクトリーを **EAP_HOME/standalone/deployments/** にコピーします。

**警告**

コピーするヘッドレス Decision Manager コントローラーデプロイメントの名前が、Red Hat JBoss EAP インスタンスの既存デプロイメントと競合しないことを確認します。

6. **TEMP_DIR/rhdm-7.2-controller-ee7/SecurityPolicy/** ディレクトリーの中身を **EAP_HOME/bin** にコピーします。ファイルの上書きを確認するメッセージが表示されたら、**Yes** を選択します。
7. **EAP_HOME/standalone/deployments/** ディレクトリーに、**controller.war.dodeploy** という名前で空のファイルを作成します。このファイルにより、サーバーが起動するとヘッドレス Decision Manager コントローラーが自動的にデプロイされます。

8. テキストエディターで **EAP_HOME/standalone/configuration/standalone.xml** ファイルを開きます。

9. 以下のプロパティーを **<system-properties>** 要素に追加し、**<NFS_STORAGE>** を、テンプレート設定が保存されている NFS ストレージへの絶対パスに置き換えます。

```
<system-properties>
  <property name="org.kie.server.controller.templatefile.watcher.enabled" value="true"/>
  <property name="org.kie.server.controller.templatefile" value="<NFS_STORAGE>"/>
</system-properties>
```

   **org.kie.server.controller.templatefile.watcher.enabled** プロパティーの値を true に設定すると、別のスレッドが開始してテンプレートファイルの修正を監視します。この確認の間隔はデフォルトで 30000 ミリ秒になり、**org.kie.server.controller.templatefile.watcher.interval** システムプロパティーで制御できます。このプロパティーの値を false に設定すると、テンプレートファイルへの変更の検出が、サーバーの再起動時に制限されます。

10. ヘッドレス Decision Manager コントローラーを開始するには、**EAP_HOME/bin** に移動して、以下のコマンドを実行します。

    - Linux または UNIX ベースのシステムの場合:

```
$ ./standalone.sh
```

    - Windows の場合:

```
standalone.bat
```

Red Hat JBoss Enterprise Application Platform のクラスターリング環境で Red Hat Decision Manager を稼働する方法の詳細情報は、**Red Hat JBoss EAP クラスター環境への Red Hat Decision Manager のインストールおよび設定** を参照してください。

# 第7章 DECISION CENTRAL に接続する 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 オーサリングまたは管理サーバー環境のデプロイメント** で、Decision Central に接続する設定手順を参照してください。

### 前提条件

Decision 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 Decision Manager インストールの計画** 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** にアクセスします。

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、URL は **http://<centralhost>:<centralport>/decision-central/rest/controller** です。**<centralhost>** と **<centralport>** はそれぞれ Decision Central のホスト名とポートになります。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 に接続する場合に自動的に作成されます。

   以下に例を示します。

```
<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/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。

# 第8章 DECISION CENTRAL により管理される DECISION SERVER の設定

**警告**

このセクションでは、テスト目的で使用可能なサンプルの設定を紹介します。一部の値は、実稼働環境には適しておらず、その旨を記載しています。

以下の手順を使用して、Decision Server インスタンスを管理するように Decision Central を設定します。

### 前提条件

以下のロールを持つユーザーが存在している

- Decision Central: **rest-all** ロールを持つユーザー
- Decision Server: **kie-server** ロールを持つユーザー

**注記**

実稼働環境では、2 人の異なるユーザーを使用し、それぞれロールを 1 つ割り当ててください。このサンプルでは、**rest-all** と **kie-server** の両ロールを持つ **controllerUser** という名前のユーザー 1 人のみを使用します。

### 手順

1. 以下の JVM プロパティーを設定します。
   Decision Central と Decision Server の場所は異なる可能性があります。このような場合、正しいサーバーインスタンスのプロパティーを設定するようにしてください。

   - Red Hat JBoss EAP で、以下のファイルの **<system-properties>** セクションを変更します。
     - スタンドアロンモードの場合: **EAP_HOME/standalone/configuration/standalone*.xml**
     - ドメインモードの場合: **EAP_HOME/domain/configuration/domain.xml**

**表8.1 管理対象 Decision Server インスタンスの JVM プロパティー**

| プロパティー | 値 | 注記 |
|---|---|---|
| **org.kie.server.id** | **default-kie-server** | Decision Server の ID |
| **org.kie.server.controller** | **http://localhost:8080/decision-central/rest/controller** | Decision Central の場所 |

| プロパティー | 値 | 注記 |
| --- | --- | --- |
| **org.kie.server.controller.user** | **controllerUser** | 前のステップで説明した **rest-all** ロールを持つユーザーの名前 |
| **org.kie.server.controller.pwd** | **controllerUser1234;** | 前のステップで説明したユーザーのパスワード |
| **org.kie.server.location** | **http://localhost:8080/kie-server/services/rest/server** | Decision Server の場所 |

表8.2 Decision Central インスタンスの JVM プロパティー

| プロパティー | 値 | 注記 |
| --- | --- | --- |
| **org.kie.server.user** | **controllerUser** | 前のステップで説明した **kie-server** ロールを持つユーザーの名前 |
| **org.kie.server.pwd** | **controllerUser1234;** | 前のステップで説明したユーザーのパスワード |

2. **http://SERVER:PORT/kie-server/services/rest/server/** に GET リクエストを送信して Decision Server が正常に起動したことを確認します。認証が終わると、以下のような XML 応答が返されます。

```
<response type="SUCCESS" msg="Kie Server info">
  <kie-server-info>
    <capabilities>KieServer</capabilities>
    <capabilities>BRM</capabilities>
    <capabilities>BPM</capabilities>
    <capabilities>CaseMgmt</capabilities>
    <capabilities>BPM-UI</capabilities>
    <capabilities>BRP</capabilities>
    <capabilities>DMN</capabilities>
    <capabilities>Swagger</capabilities>
    <location>http://localhost:8230/kie-server/services/rest/server</location>
    <messages>
      <content>Server KieServerInfo{serverId='first-kie-server', version='7.5.1.Final-redhat-1', location='http://localhost:8230/kie-server/services/rest/server', capabilities=[KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger]}started successfully at Mon Feb 05 15:44:35 AEST 2018</content>
      <severity>INFO</severity>
      <timestamp>2018-02-05T15:44:35.355+10:00</timestamp>
    </messages>
    <name>first-kie-server</name>
    <id>first-kie-server</id>
```

```
        <version>7.5.1.Final-redhat-1</version>
    </kie-server-info>
</response>
```

3. 登録が正常に完了したことを確認します。

    a. Decision Central にログインします。

    b. **Menu → Deploy → Execution Servers** の順にクリックします。
    正常に登録されている場合には、登録されたサーバーの ID が表示されます。

## 8.1. TLS 対応の SMART ROUTER の設定

TLS 対応の Smart Router (以前の KIE Server Router) 設定か可能となり、HTTPS トラフィックが使用できます。

**手順**

- ターミナルを開いて以下のコマンドを実行し、TLS 対応の Smart Router を起動します。

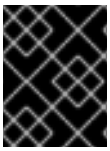
```
java -Dorg.kie.server.router.tls.keystore=PATH_TO_YOUR_KEYSTORE
    -Dorg.kie.server.router.tls.keystore.password=YOUR_KEYSTORE_PASSWD
    -Dorg.kie.server.router.tls.keystore.keyalias=YOUR_KEYSTORE_ALIAS
    -jar kie-server-router-proxy-YOUR_VERSION.jar
```

**PATH_TO_YOUR_KEYSTORE**、**YOUR_KEYSTORE_PASSWD**、**YOUR_KEYSTORE_ALIAS**、および **YOUR_VERSION** をそれぞれ関連データで置き換えます。

# 第9章 管理対象 DECISION SERVER

管理対象インスタンスには、Decision Server を起動するために利用可能な Decision Manager コントローラーが必要です。

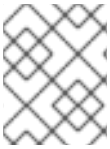
Decision Manager コントローラーは、Decision Server の設定を一元的に管理します。各 Decision Manager コントローラーは複数の設定を一度に管理でき、環境内に複数の Decision Manager コントローラーを配置することができます。管理対象 Decision Server に複数の Decision Manager コントローラーを設定できますが、一度に接続することができるのは 1 台だけです。



**重要**

どの Decision Manager コントローラーに接続されても同じ設定セットがサーバーに提供されるように、Decision Manager コントローラーはすべて同期する必要があります。

Decision Server に複数の Decision Manager コントローラーが設定されている場合には、いずれかのコントローラーとの接続が正常に確立されるまで、起動時に各コントローラーに対して接続を試みます。接続を確立できない場合は、設定でローカルのストレージが利用可能な場合でもサーバーは起動しません。こうすることで、整合性を保ち、冗長設定でサーバーが実行されるのを回避します。



**注記**

Decision Manager コントローラーに接続せずにスタンドアロンモードで Decision Server を実行する方法については、10章 *非管理対象 Decision Server* を参照してください。

# 第10章 非管理対象 DECISION SERVER

管理対象外の Decision Server はスタンドアロンインスタンスであるため、Decision Server 自体から REST/JMS API を使用して個別に設定する必要があります。再起動時には、サーバーが自動的に設定をファイルに永続化し、そのファイルが内部のサーバーの状態として使用されます。

以下の操作を実行中に、設定が更新されます。

- KIE コンテナーのデプロイ
- KIE コンテナーのデプロイ解除
- KIE コンテナーの起動
- KIE コンテナーの停止

**注記**

Decision Server が再起動すると、シャットダウン前に永続化された状態を再度確立しようと試みます。そのため、実行していた KIE コンテナー (デプロイメントユニット) は起動しますが、停止していたコンテナーは起動しません。

# 第11章 デプロイメント記述子

プロセスとルールは Apache Maven ベースのパッケージに保存され、ナレッジアーカイブ、または KJAR と呼ばれます。ルール、プロセス、アセット、およびその他のプロジェクトアーティファクトは、Maven がビルドおよび管理する JAR ファイルの一部です。**kmodule.xml** と呼ばれる、KJAR の **META-INF** ディレクトリー内に保存されるファイルを使用して、KIE ベースとセッションを定義できます。デフォルトでは、この **kmodule.xml** ファイルは空です。

Decision Central のようなランタイムコンポーネントが KJAR を処理しようとする際には、ランタイム表記のビルドのために **kmodule.xml** を検索します。

デプロイメント記述子は **kmodule.xml** ファイルを補い、デプロイメントにおいてより詳細な制御を提供します。このような記述子は任意で、記述子がなくてもデプロイメントは正常に行われます。記述子を使用して、persistence、auditing、runtime strategy といったメタ値を含む技術的属性を設定することができます。

記述子を使用すると、(サーバーレベルのデフォルト、KJAR ごとに異なるデプロイメント記述子、その他のサーバー設定という) 複数レベルで Decision Server を設定できるようになります。こうすることで、デフォルトの Decision Server 設定にシンプルなカスタマイズが可能になります (KJAR ごとなど)。

記述子は **kie-deployment-descriptor.xml** と呼ばれるファイルで定義し、**META-INF** ディレクトリーの **kmodule.xml** ファイルの隣に置くことができます。このデフォルトの場所とファイル名は、システムパラメーターとして指定すると変更できます。

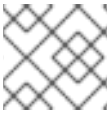
```
-Dorg.kie.deployment.desc.location=file:/path/to/file/company-deployment-descriptor.xml
```

## 11.1. デプロイメント記述子の設定

デプロイメント記述子を使用すると、ユーザーは以下の複数レベルで実行サーバーを設定することができるようになります。

- **サーバーレベル**: メインのレベルで、サーバーにデプロイされているすべての KJAR に適用されます。
- **KJAR レベル**: このレベルでは、KJAR ベースで記述子を設定できます。
- **デプロイ時レベル**: KJAR のデプロイ時に適用される記述子です。

デプロイメント記述子で指定されたより詳細な設定アイテムは、マージされるコレクションベースの設定アイテムを除いて、サーバーレベルのものよりも優先されます。優先順位は、**デプロイ時設定 > KJAR 設定 > サーバー設定** となります。



**注記**

デプロイ時の設定は、REST API によるデプロイメントに適用されます。

たとえば、サーバーレベルで定義された (設定可能なアイテムの 1 つである) persistence mode が **NONE** で、同じモードが KJAR レベルでは **JPA** と指定されている場合、その KJAR の実際のモードは **JPA** になります。その KJAR についてデプロイメント記述子で persistence mode に何も指定されていない場合 (またはデプロイメント記述子がない場合) は、サーバーレベルの設定にフォールバックします。このケースでは、**NONE** (またはサーバーレベルのデプロイメント記述子がない場合は **JPA**) になります。

**設定内容**
デプロイメント記述子では、高度な技術的設定が可能です。以下の表では、設定可能な詳細と、それぞれの許容値とデフォルト値を掲載しています。

**表11.1 デプロイメント記述子**

| 設定 | XML エントリー | 許容値 | デフォルト値 |
|---|---|---|---|
| ランタイムデータの永続ユニット名 | persistence-unit | 有効な永続パッケージ名 | org.jbpm.domain |
| 監査データの永続ユニット名 | audit-persistence-unit | 有効な永続パッケージ名 | org.jbpm.domain |
| 永続モード | persistence-mode | JPA, NONE | JPA |
| 監査モード | audit-mode | JPA、JMS、または NONE | JPA |
| ランタイムストラテジー | runtime-strategy | SINGLETON、PER_REQUEST、または PER_PROCESS_INSTANCE | SINGLETON |
| 登録するイベントリスナー一覧 | event-listeners | **ObjectModel** のような有効なリスナークラス名 | デフォルト値なし |
| 登録するタスクイベントリスナー一覧 | task-event-listeners | **ObjectModel** のような有効なリスナークラス名 | デフォルト値なし |
| 登録する作業アイテムハンドラー一覧 | work-item-handlers | **NamedObjectHandler** のような有効な作業アイテムハンドラークラス | デフォルト値なし |
| 登録するグローバル一覧 | globals | **NamedObjectModel** のような有効なグローバル変数 | デフォルト値なし |
| 登録するマーシャリングストラテジー (プラグ可能変数永続) | marshalling-strategies | 有効な **ObjectModel** クラス | デフォルト値なし |
| KJAR のリソースにアクセス可能となるために必要なロール | required-roles | 文字列のロール名 | デフォルト値なし |
| KIE セッションの追加の環境エントリー | environment-entries | 有効な **NamedObjectModel** | デフォルト値なし |
| KIE セッションの追加の設定オプション | configurations | 有効な **NamedObjectModel** | デフォルト値なし |

| 設定 | XML エントリー | 許容値 | デフォルト値 |
| --- | --- | --- | --- |
| リモートサービスのシリアル化に使用するクラス | remoteable-class | 有効な **CustomClass** | デフォルト値なし |

## 11.2. デプロイメント記述子の管理

デプロイメント記述子を設定するには、Decision Central で **Menu → Design → $PROJECT_NAME → Settings → Deployments** と移動します。

プロジェクトが作成されるたびに、ストックの **kie-deployment-descriptor.xml** ファイルがデフォルト値で生成されます。

すべての KJAR で完全なデプロイメント記述子を提供する必要はありません。部分的なデプロイメント記述子の提供は可能で、かつ推奨されるものです。たとえば、異なる監査モードを使用する必要がある場合は、その KJAR のみにそれを指定し、残りの属性はサーバーレベルのデフォルト値で定義します。

**OVERRIDE_ALL** マージモードの使用時には、すべての設定アイテムを指定する必要があります。関連する KJAR は常に指定された設定を使用し、階層内の他のデプロイメント記述子とマージしないためです。

## 11.3. ランタイムエンジンへのアクセス制限

**required-roles** 設定アイテムは、デプロイメント記述子で編集できます。このプロパティーが定義するグループに属するユーザーにのみ特定プロセスへのアクセスを付与することで、KJAR ごとまたはサーバーレベルごとにランタイムエンジンへのアクセスを制限します。

セキュリティーロールを使用してプロセス定義へのアクセスを制限したり、ランタイムでのアクセスを制限することができます。

リポジトリーの制限に基づいてこのプロパティーに必要なロールを追加するのがデフォルトの動作になります。必要な場合は、セキュリティーレルムで定義されている実際のロールに合致するロールを提供することで、このプロパティーを手動で変更できます。

**手順**

1. プロジェクトのデプロイメント記述子設定を開くには、Decision Central で **Menu → Design → $PROJECT_NAME → Settings → Deployments** の順に選択します。
2. 設定一覧から、**Required Roles** をクリックし、次に **Add Required Role** をクリックします。
3. **Add Required Role** ウィンドウで、このデプロイメントにアクセスをするパーミッションのロール名を入力し、**Add** をクリックします。
4. デプロイメントにアクセスする権限を持つロールをさらに追加するには、前の手順を繰り返します。
5. すべてのロールを追加したら、**Save** をクリックします。

# 第12章 OPENSHIFT 接続タイムアウトの設定

デフォルトでは、OpenShift のルートは 30 秒を超えた HTTP リクエストをタイムアウトするように設定されています。これにより Decision Central でセッションタイムアウト問題が発生し、以下の動作につながるおそれがあります。

- "Unable to complete your request.The following exception occurred: (TypeError) : Cannot read property 'indexOf' of null."
- "Unable to complete your request.The following exception occurred: (TypeError) : b is null."
- Decision Central で **Project** リンクまたは **Server** リンクをクリックすると、空白ページが表示される。

すべての Decision Central テンプレートには拡張タイムアウト設定が含まれています。

Decision Central OpenShift ルートのタイムアウトを長く設定するには、ターゲットルートに **haproxy.router.openshift.io/timeout: 60s** の注釈を追加します。

```
- kind: Route
  apiVersion: v1
  id: "$APPLICATION_NAME-rhdmcentr-http"
  metadata:
    name: "$APPLICATION_NAME-rhdmcentr"
    labels:
      application: "$APPLICATION_NAME"
    annotations:
      description: Route for Decision Central's http service.
      haproxy.router.openshift.io/timeout: 60s
  spec:
    host: "$DECISION_CENTRAL_HOSTNAME_HTTP"
    to:
      name: "$APPLICATION_NAME-rhdmcentr"
```

グローバルのルート固有のタイムアウト注釈の完全一覧は、OpenShift ドキュメント を参照してください。

# 第13章 LDAP ログインドメインの定義

Red Hat Decision Manager が認証と承認に LDAP を使用するように設定する際には、LDAP ログインドメインを定義します。これは、Git SSH 認証が別のセキュリティードメインを使用している可能性があるためで、その場合は認証に失敗する可能性があります。

LDAP ログインドメインを定義するには、**org.uberfire.domain** システムプロパティーを使用します。たとえば、Red Hat JBoss Enterprise Application Platform 上でこのプロパティーを以下のように **standalone.xml** ファイルに追加します。

```
<system-properties>
  <!-- other system properties -->
  <property name="org.uberfire.domain" value="LDAPAuth"/>
</system-properties>
```

認証されたユーザーが、LDAP で適切なロール (**admin**、**analyst**、**reviewer**) に関連付けられているようにしてください。

# 第14章 RH-SSO を使用したサードパーティークライアントの認証

Decision Central または Decision Server が提供するさまざまなリモートサービスを使用するには、curl、wget、Web ブラウザー、カスタムの REST クライアントなどのクライアントが、RH-SSO サーバー経由で認証を受け、要求を実行するために有効なトークンを取得する必要があります。リモートのサービスを使用するには、認証済みのユーザーに以下のロールを割り当てる必要があります。

- **rest-all**: Decision Central リモートサービスを使用する場合
- **kie-server**: Decision Server のリモートサービスを使用する場合

RH-SSO 管理コンソールを使用してこれらのロールを作成し、リモートサービスを使用するユーザーに割り当てます。

クライアントは、以下のオプションのいずれかを使用して RH-SSO 経由で認証できます。

- クライアントでサポートされている場合は Basic 認証
- トークンベースの認証

## 14.1. BASIC 認証

Decision Central および Decision Server の両方に対して RH-SSO クライアントアダプターの設定で Basic 認証を有効にした場合には、以下の例のようにトークンの付与/更新の呼び出しをせずにサービスを呼び出すことができます。

- Web ベースのリモートリポジトリーエンドポイントの場合:

```
curl http://admin:password@localhost:8080/decision-central/rest/repositories
```

- Decision Server の場合:

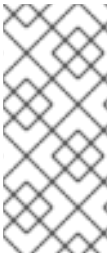
```
curl http://admin:password@localhost:8080/kie-execution-server/services/rest/server/
```

# 第15章 DECISION SERVER のシステムプロパティー

Decision Server では、以下のシステムプロパティー (ブートストラップスイッチ) を使用してサーバーの動作を設定できます。

**表15.1 Decision Server の拡張機能を無効にするシステムプロパティー**

| プロパティー | 値 | デフォルト | 説明 |
|---|---|---|---|
| **org.drools.server.ext.disabled** | **true**、**false** | **false** | **true** に設定した場合は、(ルールのサポートなど) Business Rule Management (BRM) のサポートが無効になります。 |
| **org.optaplanner.server.ext.disabled** | **true**、**false** | **false** | **true** に設定した場合は、Red Hat Business Optimizer のサポートが無効になります。 |
| **org.kie.dmn.server.ext.disabled** | **true**、**false** | **false** | **true** に設定した場合には、Decision Server DMN サポートが無効になります。 |
| **org.kie.swagger.server.ext.disabled** | **true**、**false** | **false** | **true** に設定した場合、Decision Server swagger のドキュメントサポートが無効になります。 |



**注記**

以下の表に記載した Decision Manager コントローラーのプロパティーの中で、必須と印がついているものがあります。Decision Central で Decision Server を作成または削除する場合に、このプロパティーを設定してください。Decision Central との対話なしに Decision Server を別個で使用する場合には、必須のプロパティーを設定する必要はありません。

**表15.2 Decision Manager コントローラーに必要なシステムプロパティー**

| プロパティー | 値 | デフォルト | 説明 |
|---|---|---|---|
| **org.kie.server.id** | String | 該当なし | サーバーに割り当てる任意の ID。ヘッドレス Decision Manager コントローラーが Decision Central 外に設定されている場合は、サーバーがこの ID を使用してヘッドレス Decision Manager コントローラーと接続し、KIE コンテナー設定をフェッチします。指定されていない場合、ID は自動で生成されます。 |
| **org.kie.server.user** | String | **kieserver** | Decision Manager コントローラーから Decision Server への接続に使用するユーザー名。管理モードで実行する場合には必要です。Decision Manager コントローラーを使用する場合は、このプロパティーを設定します。 |

| プロパティー | 値 | デフォルト | 説明 |
| --- | --- | --- | --- |
| **org.kie.server.pwd** | String | **kieserver1!** | コントローラーから Decision Server への接続に使用するパスワード。管理モードで実行する場合には必要です。Decision Manager コントローラーを使用する場合は、このプロパティーを設定します。 |
| **org.kie.server.token** | String | 該当なし | このプロパティーにより、Decision Manager コントローラーと Decision Server 間の認証に、ユーザー名/パスワードを使用する Basic 認証ではなく、トークンベースの認証を使用できます。Decision Manager コントローラーは、要求ヘッダーのパラメーターとしてトークンを送信します。トークンは更新されないため、サーバーには有効期限の長いアクセストークンが必要です。 |
| **org.kie.server.location** | URL | 該当なし | Decision Manager コントローラーが Decision Server インスタンスをコールバックするのに使用する URL (例: **http://localhost:8230/kie-server/services/rest/server**)。Decision Manager コントローラーを使用する場合は、このプロパティーの設定が必須です。 |
| **org.kie.server.controller** | コンマ区切りのリスト | 該当なし | Decision Manager コントローラー REST エンドポイントへの URL のコンマ区切りリスト (例: **http://localhost:8080/decision-central/rest/controller**)。Decision Manager コントローラーを使用する場合は、このプロパティーの設定が必須です。 |
| **org.kie.server.controller.user** | String | **kieserver** | Decision Manager コントローラー REST API に接続するためのユーザー名。Decision Manager コントローラーを使用する場合は、このプロパティーの設定が必須です。 |
| **org.kie.server.controller.pwd** | String | **kieserver1!** | Decision Manager コントローラー REST API に接続するためのパスワード。Decision Manager コントローラーを使用する場合は、このプロパティーの設定が必須です。 |

| プロパティー | 値 | デフォルト | 説明 |
| --- | --- | --- | --- |
| **org.kie.server.controller.token** | String | 該当なし | このプロパティーにより、Decision Manager コントローラーと Decision Server 間の認証に、ユーザー名/パスワードを使用する Basic 認証ではなく、トークンベースの認証を使用できます。このサーバーは、要求ヘッダーのパラメーターとしてトークンを送信します。トークンは更新されないため、サーバーには有効期限の長いアクセストークンが必要です。 |
| **org.kie.server.controller.connect** | Long | **10000** | サーバーの起動時に Decision Server を Decision Manager コントローラーに接続することを試み、次に試みるまでの待機時間 (ミリ秒)。 |

表15.3 キーストアを読み込むためのシステムプロパティー

| プロパティー | 値 | デフォルト | 説明 |
| --- | --- | --- | --- |
| **kie.keystore.keyStoreURL** | URL | 該当なし | Java Cryptography Extension KeyStore (JCEKS) の読み込みに使用する URL。例: **file:///home/kie/keystores/keystore.jceks** |
| **kie.keystore.keyStorePwd** | String | 該当なし | JCEKS に使用するパスワード |
| **kie.keystore.key.server.alias** | String | 該当なし | パスワードの保存先となる REST サービスのキーのエイリアス名 |
| **kie.keystore.key.server.pwd** | String | 該当なし | REST サービスのエイリアスのパスワード |
| **kie.keystore.key.ctrl.alias** | String | 該当なし | デフォルトの REST Decision Manager コントローラー用のキーのエイリアス |
| **kie.keystore.key.ctrl.pwd** | String | 該当なし | デフォルトの REST Decision Manager コントローラー用のエイリアスのパスワード |

表15.4 その他のシステムプロパティー

| プロパティー | 値 | デフォルト | 説明 |
| --- | --- | --- | --- |
| **kie.maven.settings.custom** | パス | 該当なし | Maven 設定のカスタム **settings.xml** ファイルの場所。 |

| プロパティー | 値 | デフォルト | 説明 |
| --- | --- | --- | --- |
| **kie.server.jms.queues.response** | String | **queue/KIE.SERVER.RESPONSE** | JMS に対する応答キューの JNDI 名。 |
| **org.drools.server.filter.classes** | **true**、**false** | **false** | **true** に設定した場合、Drools Decision Server の拡張機能が受け入れるのは **XmlRootElement** または **Remotable** のアノテーションが付いたカスタムクラスのみです。 |
| **org.kie.server.domain** | String | 該当なし | JMS を使用する場合にユーザーの認証に使う JAAS **LoginContext** ドメイン。 |
| **org.kie.server.repo** | パス | **をクリックします。** | Decision Server の状態ファイルが保存される場所 |

| プロパティー | 値 | デフォルト | 説明 |
| --- | --- | --- | --- |
| **org.kie.server.sync.deploy** | **true**、**false** | **false** | Decision Server に対して、Decision Manager コントローラーがコンテナーのデプロイメント設定を提供するまでデプロイメントを保持するように指示します。このプロパティーは、管理モードで実行するサーバーのみが対象です。以下のオプションが利用できます。<br><br>* **false**: Decision Manager コントローラーへの接続は非同期です。アプリケーションが起動して、Decision Manager コントローラーに接続し、成功すると、コンテナーをデプロイします。アプリケーションはコンテナーが利用可能になる前でもリクエストを受け付けます。* **true**: サーバーアプリケーションのデプロイメントは、Decision Manager コントローラーの接続スレッドと、メインのデプロイメントを結合し、完了するまで待機します。このオプションを使用すると、複数のアプリケーションが同じサーバー上にある場合に、デッドロックになる可能性があります。1 台のサーバーで使用するアプリケーションは 1 つだけにしてください。 |
| **org.kie.server.startup.strategy** | **ControllerBasedStartupStrategy**、**LocalContainersStartupStrategy** | **ControllerBasedStartupStrategy** | デプロイした KIE コンテナーの制御に使用する Decision Server の起動ストラテジーおよび、デプロイする順番 |
| **org.kie.server.mgmt.api.disabled** | **true**、**false** | **false** | **true** に設定した場合には、Decision Server 管理 API が無効になります。 |

| プロパティー | 値 | デフォルト | 説明 |
| --- | --- | --- | --- |
| **org.kie.server.xstream.enabled.packages** | **org.kie.example** などの Java パッケージ。**org.kie.example.*** のようにワイルドカード表現を指定することも可能です。 | 該当なし | XStream を使用してマーシャリングのホワイトリスト化を行うための追加パッケージを指定するプロパティー |
| **org.kie.store.services.class** | String | **org.drools.persistence.jpa.KnowledgeStoreServiceImpl** | KieSession インスタンスのブートストラップを行う **KieStoreServices** を実装する完全修飾クラス名 |

# 第16章 DECISION SERVER の機能と拡張

Decision Server の機能は、ビジネスニーズに合わせて有効化、無効化、または拡張可能なプラグインにより決まります。Decision Server は以下の機能および拡張をサポートします。

**表16.1 Decision Server の機能と拡張**

| 機能名 | 拡張名 | 説明 |
|---|---|---|
| **KieServer** | **KieServer** | サーバーインスタンスでの KIE コンテナーの作成や削除など、Decision Server のコア機能を提供します。 |
| **BRM** | **Drools** | ファクトの挿入やビジネスルールの実行など、ビジネスルール管理 (BRM) 機能を提供します。 |
| **BRP** | **OptaPlanner** | ソルバーの実装など、ビジネスリソースプランニング (BRP) 機能を提供します。 |
| **DMN** | **DMN** | DMN データ型の管理や DMN モデルの実行など、Decision Model and Notation (DMN) 機能を提供します。 |
| **Swagger** | **Swagger** | Decision Server REST API と対話するための Swagger の Web インターフェイス機能を提供します。 |

実行中の Decision Server インスタンスに対応する拡張を表示するには、以下の REST API エンドポイントに **GET** 要求を送信して、XML または JSON サーバーの要求を確認します。

### Decision Server の情報に対する GET 要求のベース URL

```
http://SERVER:PORT/kie-server/services/rest/server
```

### Decision Server の情報を含む JSON 応答の例

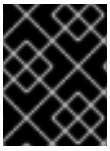
```
{
  "type": "SUCCESS",
  "msg": "Kie Server info",
  "result": {
    "kie-server-info": {
      "id": "test-kie-server",
      "version": "7.26.0.20190818-050814",
      "name": "test-kie-server",
      "location": "http://localhost:8080/kie-server/services/rest/server",
      "capabilities": [
        "KieServer",
        "BRM",
        "BRP",
        "DMN",
        "Swagger"
      ],
      "messages": [
        {
          "severity": "INFO",
```

```
        "timestamp": {
          "java.util.Date": 1566169865791
        },
        "content": [
          "Server KieServerInfo{serverId='test-kie-server', version='7.26.0.20190818-050814',
name='test-kie-server', location='http:/localhost:8080/kie-server/services/rest/server', capabilities=
[KieServer, BRM, BRP, DMN, Swagger]', messages=null', mode=DEVELOPMENT}started
successfully at Sun Aug 18 23:11:05 UTC 2019"
        ]
       }
      ],
      "mode": "DEVELOPMENT"
    }
  }
}
```

Decision Server 拡張機能を有効または無効にするには、関連する Decision Server システムプロパティー (***.server.ext.disabled**) を設定します。たとえば、**BRM** 機能を無効にするには、**org.drools.server.ext.disabled=true** システムプロパティーを設定します。全 Decision Server システムプロパティーについては、15章*Decision Server のシステムプロパティー* を参照してください。

デフォルトでは、Decision Server 拡張機能は REST または JMS データトランスポートで公開され、事前定義済みのクライアント API を使用します。追加の REST エンドポイントで既存の Decision Server 機能を拡張するか、REST または JMS 以外の対応するトランスポートメソッドを拡張するか、Decision Server クライアントの機能を拡張できます。

Decision Server 機能は柔軟であるため、デフォルトの Decision Server 機能にビジネスニーズを合わせるのではなく、Decision Server インスタンスをビジネスニーズに適合できます。



**重要**

Decision Server 機能を拡張した場合には、Red Hat では、カスタムの実装や拡張の一部として使用したカスタムコードをサポートしません。

## 16.1. カスタム REST API エンドポイントを使用した既存の DECISION SERVER 機能の拡張

Decision Server REST API を使用すると、Decision Central ユーザーインターフェイスを使わずに Red Hat Decision Manager の KIE コンテナーやビジネスアセット (ビジネスルールやプロセス、ソルバーなど) を操作することができます。利用可能な REST エンドポイントは、Decision Server システムプロパティーで有効にした機能により決まります (例: **BRM** 機能は **org.drools.server.ext.disabled=false**)。既存の Decision Server 機能は、カスタムの REST API エンドポイントで拡張し、ビジネスニーズに合わせて Decision Server REST API を適合できます。

たとえば、この手順では、以下のカスタム REST API エンドポイントで **Drools** Decision Server 機能 (**BRM** 機能向け) を拡張します。

**カスタム REST API エンドポイントの例**

```
/server/containers/instances/{containerId}/ksession/{ksessionId}
```

このカスタムのエンドポイントの例では、{DECISION_ENGINE} の作業メモリーに挿入するファクト一覧を受け入れ、自動的に全ルールを実行して、指定の KIE コンテナーで KIE セッションからのオブジェクトをすべて取得します。

**手順**

1. 空の Maven プロジェクトを作成して、以下のパッケージタイプと依存関係を、プロジェクトの **pom.xml** ファイルに定義します。

**サンプルプロジェクトの pom.xml ファイルの例**

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.14.0.Final-redhat-00002</version.org.kie>
</properties>

<dependencies>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-internal</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-common</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-drools</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-rest-common</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-core</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
```

```
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
  </dependency>
</dependencies>
```

2. 以下の例のように、プロジェクトの Java クラスに **org.kie.server.services.api.KieServerApplicationComponentsService** インターフェイスを実装します。

**KieServerApplicationComponentsService インターフェイスの実装例**

```
public class CusomtDroolsKieServerApplicationComponentsService implements
KieServerApplicationComponentsService { (1)

    private static final String OWNER_EXTENSION = "Drools"; (2)

    public Collection<Object> getAppComponents(String extension, SupportedTransports
type, Object... services) { (3)
        // Do not accept calls from extensions other than the owner extension:
        if ( !OWNER_EXTENSION.equals(extension) ) {
            return Collections.emptyList();
        }

        RulesExecutionService rulesExecutionService = null; (4)
        KieServerRegistry context = null;

        for( Object object : services ) {
            if( RulesExecutionService.class.isAssignableFrom(object.getClass()) ) {
                rulesExecutionService = (RulesExecutionService) object;
                continue;
            } else if( KieServerRegistry.class.isAssignableFrom(object.getClass()) ) {
                context = (KieServerRegistry) object;
                continue;
            }
        }

        List<Object> components = new ArrayList<Object>(1);
        if( SupportedTransports.REST.equals(type) ) {
            components.add(new CustomResource(rulesExecutionService, context)); (5)
        }

        return components;
    }

}
```

(1) アプリケーションの起動時にデプロイされる Decision Server インフラストラクチャーに REST エンドポイントを提供します。

(2) この例の **Drools** 拡張など、拡張する機能を指定します。

(3) REST コンテナーがデプロイする必要のある全リソースを返します。Decision Server インスタンスで有効化した各拡張で、**getAppComponents** メソッドを呼び出して、指定した **OWNER_EXTENSION** 以外の拡張の空のコレクションを、**if ( !OWNER_EXTENSION.equals(extension) )** の呼び出しで返します。

4 この例の **Drools** 拡張の **RulesExecutionService** や **KieServerRegistry** サービスなど、指定の拡張から使用するサービスを表示します。

5 **components** リストの一部としてリソースを返す **CustomResource** クラスと、拡張のトランスポートタイプを **REST** または **JMS** に指定します (この例では **REST**)。

3. 以下の例のように、Decision Server を使用して新規の REST リソースの機能を追加する **CustomResource** クラスを実装します。

**CustomResource クラスの実装例**

```
// Custom base endpoint:
@Path("server/containers/instances/{containerId}/ksession")
public class CustomResource {

    private static final Logger logger = LoggerFactory.getLogger(CustomResource.class);

    private KieCommands commandsFactory = KieServices.Factory.get().getCommands();

    private RulesExecutionService rulesExecutionService;
    private KieServerRegistry registry;

    public CustomResource() {

    }

    public CustomResource(RulesExecutionService rulesExecutionService, KieServerRegistry
registry) {
        this.rulesExecutionService = rulesExecutionService;
        this.registry = registry;
    }

    // Supported HTTP method, path parameters, and data formats:
    @POST
    @Path("/{ksessionId}")
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Response insertFireReturn(@Context HttpHeaders headers,
            @PathParam("containerId") String id,
            @PathParam("ksessionId") String ksessionId,
            String cmdPayload) {

        Variant v = getVariant(headers);
        String contentType = getContentType(headers);

        // Marshalling behavior and supported actions:
        MarshallingFormat format = MarshallingFormat.fromType(contentType);
        if (format == null) {
            format = MarshallingFormat.valueOf(contentType);
        }
        try {
            KieContainerInstance kci = registry.getContainer(id);

            Marshaller marshaller = kci.getMarshaller(format);
```

```
            List<?> listOfFacts = marshaller.unmarshall(cmdPayload, List.class);

            List<Command<?>> commands = new ArrayList<Command<?>>();
            BatchExecutionCommand executionCommand =
commandsFactory.newBatchExecution(commands, ksessionId);

            for (Object fact : listOfFacts) {
               commands.add(commandsFactory.newInsert(fact, fact.toString()));
            }
            commands.add(commandsFactory.newFireAllRules());
            commands.add(commandsFactory.newGetObjects());

            ExecutionResults results = rulesExecutionService.call(kci, executionCommand);

            String result = marshaller.marshall(results);


            logger.debug("Returning OK response with content '{}'", result);
            return createResponse(result, v, Response.Status.OK);
         } catch (Exception e) {
            // If marshalling fails, return the `call-container` response to maintain backward
compatibility:
            String response = "Execution failed with error : " + e.getMessage();
            logger.debug("Returning Failure response with content '{}'", response);
            return createResponse(response, v,
Response.Status.INTERNAL_SERVER_ERROR);
         }

      }
}
```

この例では、カスタムエンドポイントの **CustomResource** クラスで、以下のデータと動作を指定します。

- **server/containers/instances/{containerId}/ksession** のベースポイントを使用します。
- **POST** HTTP メソッドを使用します。
- REST 要求で以下のデータを指定する必要があります。
  - パスの引数として **containerId**
  - パスの引数として **ksessionId**
  - メッセージペイロードとしてファクトの一覧
- 全 Decision Server データ形式をサポートします。
  - XML (JAXB、XStream)
  - JSON
- **List<?>** コレクションにペイロードをアンマーシャリングして、リスト内のアイテムごとに、**InsertCommand** インスタンスを作成し、その後に **FireAllRules** と **GetObject** コマンドを追加します。

- {DECISION_ENGINE} を呼び出す **BatchExecutionCommand** インスタンスに全コマンドを追加します。

4. 新規エンドポイントを Decision Server で検出できるようにするには、Maven プロジェクト内に **META-INF/services/org.kie.server.services.api.KieServerApplicationComponentsService** ファイルを作成して、このファイルに **KieServerApplicationComponentsService** 実装クラスの完全修飾名を追加します。たとえば、このファイルには、**org.kie.server.ext.drools.rest.CusomtDroolsKieServerApplicationComponentsService** の 1 行が含まれます。

5. プロジェクトを構築して、作成された JAR ファイルをプロジェクトの **~/kie-server.war/WEB-INF/lib** ディレクトリーにコピーします。たとえば、Red Hat JBoss EAP ではこのディレクトリーへのパスは **EAP_HOME/standalone/deployments/kie-server.war/WEB-INF/lib** です。

6. Decision Server を起動して、実行中の Decision Server に構築したプロジェクトをデプロイします。プロジェクトは、Decision Central インターフェイスまたは Decision Server REST API (**http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}** への **PUT** 要求) のいずれかを使用してデプロイできます。
実行中の Decision Server にプロジェクトを追加した後に、新しい REST エンドポイントとの対話を開始します。

   今回の例では、以下の情報を使用して新規エンドポイントを呼び出すことができます。

   - **要求 URL 例: http://localhost:8080/kie-server/services/rest/server/containers/instances/demo/ksession/defaultKieSession**
   - **HTTP メソッド: POST**
   - **HTTP ヘッダー:**
     - **Content-Type: application/json**
     - **Accept: application/json**
   - **メッセージペイロードの例:**

```
[
  {
    "org.jbpm.test.Person": {
      "name": "john",
      "age": 25
    }
  },
  {
    "org.jbpm.test.Person": {
      "name": "mary",
      "age": 22
    }
  }
]
```

   - **サーバーの応答例: 200** (success)
   - **サーバーのログ出力例:**

```
13:37:20,347 INFO  [stdout] (default task-24) Hello mary
13:37:20,348 INFO  [stdout] (default task-24) Hello john
```

## 16.2. カスタムデータトランスポートを使用するための DECISION SERVER の拡張

デフォルトでは、Decision Server の拡張が REST または JMS データトランスポートを使用して公開されます。Decision Server を拡張して、カスタムのデータトランスポートのサポートを追加し、Decision Server トランスポートプロトコルをビジネスニーズに適合します。

たとえば、以下の手順では、**Drools** 拡張を使用し、Apache MINA (オープンソースの Java ネットワークアプリケーションフレームワーク) をベースとする Decision Server にカスタムのデータトランスポートを追加します。カスタムの MINA トランスポートの例では、既存のマーシャリング操作に依存し、JSON 形式のみをサポートする文字列ベースのデータを変換します。

**手順**

1. 空の Maven プロジェクトを作成して、以下のパッケージタイプと依存関係を、プロジェクトの **pom.xml** ファイルに定義します。

   **サンプルプロジェクトの pom.xml ファイルの例**

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.14.0.Final-redhat-00002</version.org.kie>
</properties>

<dependencies>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-internal</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-common</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-drools</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
```

```
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-core</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
  </dependency>
  <dependency>
    <groupId>org.apache.mina</groupId>
    <artifactId>mina-core</artifactId>
    <version>2.1.3</version>
  </dependency>
</dependencies>
```

2. 以下の例のように、プロジェクトの Java クラスに **org.kie.server.services.api.KieServerExtension** インターフェイスを実装します。

**KieServerExtension インターフェイスの実装例**

```
public class MinaDroolsKieServerExtension implements KieServerExtension {

    private static final Logger logger =
LoggerFactory.getLogger(MinaDroolsKieServerExtension.class);

    public static final String EXTENSION_NAME = "Drools-Mina";

    private static final Boolean disabled =
Boolean.parseBoolean(System.getProperty("org.kie.server.drools-mina.ext.disabled",
"false"));
    private static final String MINA_HOST = System.getProperty("org.kie.server.drools-
mina.ext.port", "localhost");
    private static final int MINA_PORT =
Integer.parseInt(System.getProperty("org.kie.server.drools-mina.ext.port", "9123"));

    // Taken from dependency on the `Drools` extension:
    private KieContainerCommandService batchCommandService;

    // Specific to MINA:
    private IoAcceptor acceptor;

    public boolean isActive() {
        return disabled == false;
    }

    public void init(KieServerImpl kieServer, KieServerRegistry registry) {

        KieServerExtension droolsExtension = registry.getServerExtension("Drools");
        if (droolsExtension == null) {
```

```
            logger.warn("No Drools extension available, quiting...");
            return;
        }

        List<Object> droolsServices = droolsExtension.getServices();
        for( Object object : droolsServices ) {
            // If the given service is null (not configured), continue to the next service:
            if (object == null) {
                continue;
            }
            if( KieContainerCommandService.class.isAssignableFrom(object.getClass()) ) {
                batchCommandService = (KieContainerCommandService) object;
                continue;
            }
        }
        if (batchCommandService != null) {
            acceptor = new NioSocketAcceptor();
            acceptor.getFilterChain().addLast( "codec", new ProtocolCodecFilter( new
TextLineCodecFactory( Charset.forName( "UTF-8" ))));

            acceptor.setHandler( new TextBasedIoHandlerAdapter(batchCommandService) );
            acceptor.getSessionConfig().setReadBufferSize( 2048 );
            acceptor.getSessionConfig().setIdleTime( IdleStatus.BOTH_IDLE, 10 );
            try {
                acceptor.bind( new InetSocketAddress(MINA_HOST, MINA_PORT) );

                logger.info("{} -- Mina server started at {} and port {}", toString(), MINA_HOST,
MINA_PORT);
            } catch (IOException e) {
                logger.error("Unable to start Mina acceptor due to {}", e.getMessage(), e);
            }

        }
    }

    public void destroy(KieServerImpl kieServer, KieServerRegistry registry) {
        if (acceptor != null) {
            acceptor.dispose();
            acceptor = null;
        }
        logger.info("{} -- Mina server stopped", toString());
    }

    public void createContainer(String id, KieContainerInstance kieContainerInstance,
Map<String, Object> parameters) {
        // Empty, already handled by the `Drools` extension

    }

    public void disposeContainer(String id, KieContainerInstance kieContainerInstance,
Map<String, Object> parameters) {
      // Empty, already handled by the `Drools` extension

    }

    public List<Object> getAppComponents(SupportedTransports type) {
```

```
        // Nothing for supported transports (REST or JMS)
        return Collections.emptyList();
    }

    public <T> T getAppComponents(Class<T> serviceType) {

        return null;
    }

    public String getImplementedCapability() {
        return "BRM-Mina";
    }

    public List<Object> getServices() {
        return Collections.emptyList();
    }

    public String getExtensionName() {
        return EXTENSION_NAME;
    }

    public Integer getStartOrder() {
        return 20;
    }

    @Override
    public String toString() {
        return EXTENSION_NAME + " KIE Server extension";
    }
}
```

**KieServerExtension** インターフェイスは、新規の MINA トランスポートの機能を追加する時に Decision Server が使用する主要な拡張インターフェイスです。このインターフェイスには、以下のコンポーネントが含まれます。

**KieServerExtension インターフェイスの概要**

```
public interface KieServerExtension {

    boolean isActive();

    void init(KieServerImpl kieServer, KieServerRegistry registry);

    void destroy(KieServerImpl kieServer, KieServerRegistry registry);

    void createContainer(String id, KieContainerInstance kieContainerInstance, Map<String,
Object> parameters);

    void disposeContainer(String id, KieContainerInstance kieContainerInstance, Map<String,
Object> parameters);

    List<Object> getAppComponents(SupportedTransports type);

    <T> T getAppComponents(Class<T> serviceType);
```

```
    String getImplementedCapability(); (1)

    List<Object> getServices();

    String getExtensionName(); (2)

    Integer getStartOrder(); (3)
}
```

(1) この拡張で対応している機能を指定します。この機能は、Decision Server 内で一意でなければなりません。

(2) 拡張は、人間が解読可能な名前に定義します。

(3) 指定した拡張の起動のタイミングを決定します。他の拡張と依存関係がある拡張の場合、この設定は親の設定と競合しないようにしてください。たとえば、今回の場合、このカスタムの拡張は **Drools** 拡張に依存しており、Drool 拡張の **StartOrder** は **0** に設定されているため、このカスタムのアドオン拡張は **0** を超える値でなければなりません (サンプルの実装では **20** に設定)。

このインターフェイスの先程の **MinaDroolsKieServerExtension** 実装例では、**init** メソッドが主に、**Drools** 拡張からサービスを収集して、MINA サーバーをブートストラップ化する要素となっています。**KieServerExtension** インターフェイスの他のメソッドは、標準の実装のままで、インターフェイスの要件を満たします。

**TextBasedIoHandlerAdapter** クラスは、受信要求に対応する MINA サーバーにあるハンドラーです。

3. 以下の例のように、MINA サーバーの **TextBasedIoHandlerAdapter** ハンドラーを実装します。

**TextBasedIoHandlerAdapter ハンドラーの実装例**

```
public class TextBasedIoHandlerAdapter extends IoHandlerAdapter {

    private static final Logger logger =
LoggerFactory.getLogger(TextBasedIoHandlerAdapter.class);

    private KieContainerCommandService batchCommandService;

    public TextBasedIoHandlerAdapter(KieContainerCommandService
batchCommandService) {
        this.batchCommandService = batchCommandService;
    }

    @Override
    public void messageReceived( IoSession session, Object message ) throws Exception {
        String completeMessage = message.toString();
        logger.debug("Received message '{}'", completeMessage);
        if( completeMessage.trim().equalsIgnoreCase("quit") ||
completeMessage.trim().equalsIgnoreCase("exit") ) {
            session.close(false);
            return;
        }
```

```
        String[] elements = completeMessage.split("\\|");
        logger.debug("Container id {}", elements[0]);
        try {
            ServiceResponse<String> result = batchCommandService.callContainer(elements[0],
elements[1], MarshallingFormat.JSON, null);

            if (result.getType().equals(ServiceResponse.ResponseType.SUCCESS)) {
                session.write(result.getResult());
                logger.debug("Successful message written with content '{}'", result.getResult());
            } else {
                session.write(result.getMsg());
                logger.debug("Failure message written with content '{}'", result.getMsg());
            }
        } catch (Exception e) {

        }
    }
}
```

この例では、ハンドラークラスはテキストメッセージを受信して、**Drools** サービスでこのメッセージを実行します。

**TextBasedIoHandlerAdapter** ハンドラー実装を使用する場合は、以下のハンドラー要件と動作を考慮してください。

- 各受信トランスポート要求が 1 行であるため、ハンドラーに送信する内容は、1 行でなければなりません。
- ハンドラーで **containerID|payload** の形式が想定されるように、この 1 行に KIE コンテナー ID を渡す必要があります。
- マーシャラーで生成される方法で応答を設定できます。応答は複数行にすることができます。
- このハンドラーは **stream mode** をサポートし、Decision Server セッションを切断せずにコマンドを送信できます。ストリームモードで Decision Server セッションを終了するには、サーバーに **exit** または **quit** コマンドを送信してください。

4. 新規のデータトランスポートを Decision Server で検出できるようにするには、Maven プロジェクトで **META-INF/services/org.kie.server.services.api.KieServerExtension** ファイルを作成し、このファイルに **KieServerExtension** 実装クラスの完全修飾名を追加します。たとえば、このファイルには **org.kie.server.ext.mina.MinaDroolsKieServerExtension** の 1 行が含まれます。

5. プロジェクトを構築して、作成された JAR ファイルと **mina-core-2.0.9.jar** ファイル (今回の例でこの拡張が依存) をプロジェクトの **~/kie-server.war/WEB-INF/lib** ディレクトリーにコピーします。たとえば、Red Hat JBoss EAP ではこのディレクトリーへのパスは **EAP_HOME/standalone/deployments/kie-server.war/WEB-INF/lib** です。

6. Decision Server を起動して、実行中の Decision Server に構築したプロジェクトをデプロイします。プロジェクトは、Decision Central インターフェイスまたは Decision Server REST API (**http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}** への **PUT** 要求) のいずれかを使用してデプロイできます。
プロジェクトを実行中の Decision Server にデプロイした後に、Decision Server ログで新規データトランスポートのステータスを表示して、新規データトランスポートの使用を開始できます。

**サーバーログの新規データトランスポート**

```
Drools-Mina KIE Server extension -- Mina server started at localhost and port 9123
Drools-Mina KIE Server extension has been successfully registered as server extension
```

この例では、Telnet を使用して Decision Server の 新しい MINA ベースのデータトランスポートと対話できます。

**コマンドターミナルでの Telnet の開始およびポート 9123 での Decision Server の接続**

```
telnet 127.0.0.1 9123
```

**コマンドターミナルでの Decision Server との対話例**

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

# Request body:
demo|{"lookup":"defaultKieSession","commands":[{"insert":{"object":{"org.jbpm.test.Person":
{"name":"john","age":25}}}},{"fire-all-rules":""}]}

# Server response:
{
  "results" : [ {
    "key" : "",
    "value" : 1
  } ],
  "facts" : [ ]
}

demo|{"lookup":"defaultKieSession","commands":[{"insert":{"object":{"org.jbpm.test.Person":
{"name":"mary","age":22}}}},{"fire-all-rules":""}]}
{
  "results" : [ {
    "key" : "",
    "value" : 1
  } ],
  "facts" : [ ]
}

demo|{"lookup":"defaultKieSession","commands":[{"insert":{"object":{"org.jbpm.test.Person":
{"name":"james","age":25}}}},{"fire-all-rules":""}]}
{
  "results" : [ {
    "key" : "",
    "value" : 1
  } ],
  "facts" : [ ]
}
exit
Connection closed by foreign host.
```

**サーバーログの出力例**

```
16:33:40,206 INFO  [stdout] (NioProcessor-2) Hello john
16:34:03,877 INFO  [stdout] (NioProcessor-2) Hello mary
16:34:19,800 INFO  [stdout] (NioProcessor-2) Hello james
```

## 16.3. カスタムクライアント API を使用した DECISION SERVER のクライアント拡張

Decision Server は、Decision Server サービスの使用時に対話可能な、事前定義済みのクライアント API を使用します。カスタムのクライアント API で Decision Server クライアントを拡張して、ビジネスのニーズに Decision Server サービスを適合させます。

たとえば、以下の手順では、カスタムのクライアント API を Decision Server に追加して、Apache MINA (オープンソースの Java ネットワークアプリケーションフレームワーク) をもとにした、カスタムのデータトランスポートに対応します (このシナリオ向けにすでに設定済み)。

**手順**

1. 空の Maven プロジェクトを作成して、以下のパッケージタイプと依存関係を、プロジェクトの **pom.xml** ファイルに定義します。

   **サンプルプロジェクトの pom.xml ファイルの例**

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.14.0.Final-redhat-00002</version.org.kie>
 </properties>

 <dependencies>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
     <groupId>org.kie.server</groupId>
     <artifactId>kie-server-client</artifactId>
     <version>${version.org.kie}</version>
   </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
 </dependencies>
```

2. 以下の例のように、プロジェクトの Java クラスに、関連する **ServicesClient** インターフェイスを実装します。

   **RulesMinaServicesClient インターフェイスの例**

```
public interface RulesMinaServicesClient extends RuleServicesClient {

}
```

インターフェイスをもとにクライアントの実装を登録する必要があるため、特定のインターフェイスが必要です。また、指定のインターフェイスには実装は 1 つしか指定できません。

この例では、カスタムの MINA ベースのデータトランスポートが **Drools** 拡張を使用し、この **RulesMinaServicesClient** インターフェイスの例は、**Drools** 拡張から、既存の **RuleServicesClient** クライアント API を拡張します。

3. 以下の例のように、新規の MINA トランスポートのクライアント機能を追加するのに Decision Server が使用可能な **RulesMinaServicesClient** インターフェイスを実装します。

**RulesMinaServicesClient インターフェイスの実装例**

```
public class RulesMinaServicesClientImpl implements RulesMinaServicesClient {

    private String host;
    private Integer port;

    private Marshaller marshaller;

    public RulesMinaServicesClientImpl(KieServicesConfiguration configuration, ClassLoader
classloader) {
        String[] serverDetails = configuration.getServerUrl().split(":");

        this.host = serverDetails[0];
        this.port = Integer.parseInt(serverDetails[1]);

        this.marshaller = MarshallerFactory.getMarshaller(configuration.getExtraJaxbClasses(),
MarshallingFormat.JSON, classloader);
    }

    public ServiceResponse<String> executeCommands(String id, String payload) {

        try {
            String response = sendReceive(id, payload);
            if (response.startsWith("{")) {
                return new ServiceResponse<String>(ResponseType.SUCCESS, null, response);
            } else {
                return new ServiceResponse<String>(ResponseType.FAILURE, response);
            }
        } catch (Exception e) {
            throw new KieServicesException("Unable to send request to KIE Server", e);
        }
    }

    public ServiceResponse<String> executeCommands(String id, Command<?> cmd) {
        try {
            String response = sendReceive(id, marshaller.marshall(cmd));
            if (response.startsWith("{")) {
                return new ServiceResponse<String>(ResponseType.SUCCESS, null, response);
            } else {
                return new ServiceResponse<String>(ResponseType.FAILURE, response);
```

```
            }
        } catch (Exception e) {
            throw new KieServicesException("Unable to send request to KIE Server", e);
        }
    }

    protected String sendReceive(String containerId, String content) throws Exception {

        // Flatten the content to be single line:
        content = content.replaceAll("\\n", "");

        Socket minaSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        StringBuffer data = new StringBuffer();
        try {
            minaSocket = new Socket(host, port);
            out = new PrintWriter(minaSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(minaSocket.getInputStream()));

            // Prepare and send data:
            out.println(containerId + "|" + content);
            // Wait for the first line:
            data.append(in.readLine());
            // Continue as long as data is available:
            while (in.ready()) {
                data.append(in.readLine());
            }

            return data.toString();
        } finally {
            out.close();
            in.close();
            minaSocket.close();
        }
    }
}
```

この実装例は、以下のデータおよび動作を指定します。

- ソケットベースの通信を使用して簡素化します。
- Decision Server クライアントのデフォルト設定に依存し、**ServerUrl** を使用して MINA サーバーのホストとポートを提供します。
- マーシャリング形式で JSON を指定します。
- 受信メッセージは左波括弧 **{** で始まる JSON オブジェクトでなければなりません。
- 応答の最初の行を待機中に、ブロッキング API と直接、ソケット通信を使用してから、利用可能なすべての行を読み取ります。
- **ストリームモード** を使用しないので、コマンドの呼び出し後に Decision Server セッションを切断します。

4. 以下の例のように、プロジェクトの Java クラスに **org.kie.server.client.helper.KieServicesClientBuilder** インターフェイスを実装します。

**KieServicesClientBuilder インターフェイスの実装例**

```
public class MinaClientBuilderImpl implements KieServicesClientBuilder { (1)

    public String getImplementedCapability() { (2)
        return "BRM-Mina";
    }

    public Map<Class<?>, Object> build(KieServicesConfiguration configuration, ClassLoader
classLoader) { (3)
        Map<Class<?>, Object> services = new HashMap<Class<?>, Object>();

        services.put(RulesMinaServicesClient.class, new
RulesMinaServicesClientImpl(configuration, classLoader));

        return services;
    }

}
```

(1) 一般の Decision Server クライアントインフラストラクチャーにクライアント API を追加できます。

(2) クライアントが使用する Decision Server 機能 (拡張) を定義します。

(3) クライアントの実装のマッピングを提供します。キーはインターフェイス、値は完全な初期実装です。

5. 新規のクライアント API を Decision Server クライアントで検出できるようにするには、Maven プロジェクトで **META-INF/services/org.kie.server.client.helper.KieServicesClientBuilder** ファイルを作成し、このファイルに **KieServicesClientBuilder** 実装クラスの完全修飾名を追加します。たとえば、このファイルには **org.kie.server.ext.mina.client.MinaClientBuilderImpl** の 1 行が含まれます。

6. プロジェクトを構築して、作成された JAR ファイルをプロジェクトの **~/kie-server.war/WEB-INF/lib** ディレクトリーにコピーします。たとえば、Red Hat JBoss EAP ではこのディレクトリーへのパスは **EAP_HOME/standalone/deployments/kie-server.war/WEB-INF/lib** です。

7. Decision Server を起動して、実行中の Decision Server に構築したプロジェクトをデプロイします。プロジェクトは、Decision Central インターフェイスまたは Decision Server REST API (**http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}** への **PUT** 要求) のいずれかを使用してデプロイできます。
実行中の Decision Server にプロジェクトをデプロイしたあとに、新規の Decision Server クライアントと対話を開始できます。標準の Decision Server クライアントと同じ方法で、クライアント設定とクライアントインスタンスを作成して、タイプ別にサービスクライアントを取得し、クライアントメソッドを呼び出して、新しいクライアントを使用します。

たとえば、**RulesMinaServiceClient** クライアントインスタンスを作成して、MINA トランスポートを使用して Decision Server で操作を呼び出すことができます。

**RulesMinaServiceClient クライアント作成の実装例**

```
protected RulesMinaServicesClient buildClient() {
    KieServicesConfiguration configuration =
KieServicesFactory.newRestConfiguration("localhost:9123", null, null);
    List<String> capabilities = new ArrayList<String>();
    // Explicitly add capabilities (the MINA client does not respond to `get-server-info`
requests):
    capabilities.add("BRM-Mina");

    configuration.setCapabilities(capabilities);
    configuration.setMarshallingFormat(MarshallingFormat.JSON);

    configuration.addJaxbClasses(extraClasses);

    KieServicesClient kieServicesClient =
KieServicesFactory.newKieServicesClient(configuration);

    RulesMinaServicesClient rulesClient =
kieServicesClient.getServicesClient(RulesMinaServicesClient.class);

    return rulesClient;
}
```

**MINA トランスポートを使用して Decision Server 上で操作を呼び出す設定例**

```
RulesMinaServicesClient rulesClient = buildClient();

List<Command<?>> commands = new ArrayList<Command<?>>();
BatchExecutionCommand executionCommand =
commandsFactory.newBatchExecution(commands, "defaultKieSession");

Person person = new Person();
person.setName("mary");
commands.add(commandsFactory.newInsert(person, "person"));
commands.add(commandsFactory.newFireAllRules("fired"));

ServiceResponse<String> response = rulesClient.executeCommands(containerId,
executionCommand);
Assert.assertNotNull(response);

Assert.assertEquals(ResponseType.SUCCESS, response.getType());

String data = response.getResult();

Marshaller marshaller = MarshallerFactory.getMarshaller(extraClasses,
MarshallingFormat.JSON, this.getClass().getClassLoader());

ExecutionResultImpl results = marshaller.unmarshall(data, ExecutionResultImpl.class);
Assert.assertNotNull(results);

Object personResult = results.getValue("person");
Assert.assertTrue(personResult instanceof Person);

Assert.assertEquals("mary", ((Person) personResult).getName());
Assert.assertEquals("JBoss Community", ((Person) personResult).getAddress());
Assert.assertEquals(true, ((Person) personResult).isRegistered());
```

# 第17章 関連情報

- Red Hat JBoss EAP への Red Hat Decision Manager のインストールおよび設定
- Red Hat Decision Manager インストールの計画
- Red Hat JBoss EAP への Red Hat Decision Manager のインストールおよび設定
- Red Hat OpenShift Container Platform への Red Hat Decision 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 Decision Manager 管理対象サーバー環境のデプロイ

# 付録A バージョン情報

本書の最終更新日: 2021 年 11 月 15 日 (月)