



Red Hat Decision Manager 7.12

Red Hat Decision Manager と他の製品およびコンポーネントの統合

法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

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

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

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

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書では、Spring Boot、Red Hat Single Sign-On などの他の製品やコンポーネントと Red Hat Decision Manager を統合する方法を説明します。

目次

はじめに	4
多様性を受け入れるオープンソースの強化	5
パート I. SPRING BOOT での RED HAT DECISION MANAGER ビジネスアプリケーションの作成	6
第1章 RED HAT DECISION MANAGER SPRING BOOT ビジネスアプリケーション	7
第2章 ビジネスアプリケーションの作成	8
第3章 APACHE MAVEN および RED HAT DECISION MANAGER SPRING BOOT アプリケーション	9
3.1. オンライン MAVEN リポジトリ用の RED HAT DECISION MANAGER SPRING BOOT プロジェクトの設定	9
3.2. RED HAT DECISION MANAGER MAVEN リポジトリのダウンロードおよび設定	10
第4章 RED HAT DECISION MANAGER での SPRING SECURITY	13
4.1. SPRING SECURITY を使用した承認による認証	14
4.2. RED HAT DECISION MANAGER ビジネスアプリケーションでの SPRING SECURITY の無効化	15
4.3. 事前認証での SPRING セキュリティーの使用	16
4.4. RED HAT SINGLE SIGN-ON を使用したビジネスアプリケーションの設定	18
第5章 RED HAT DECISION MANAGER SPRING BOOT の設定	21
5.1. SPRING BOOT アプリケーションの REST エンドポイントの設定	21
5.2. KIE SERVER アイデンティティーの設定	21
5.3. ランタイム時に起動する KIE SERVER コンポーネントの設定	22
5.4. ビジネスアプリケーションのユーザーグループプロバイダーの設定	23
5.5. SWAGGER ドキュメントの有効化	24
第6章 自己完結型 RED HAT DECISION MANAGER SPRING BOOT JAR ファイルの作成	26
第7章 ビジネスアプリケーションの実行	31
7.1. スタンドアロンモードでのビジネスアプリケーションの実行	31
7.2. 開発モードでのビジネスアプリケーションの実行	32
第8章 RED HAT OPENSIFT CONTAINER PLATFORM での SPRINGBOOT ビジネスアプリケーションの実行 ..	34
第9章 BUSINESS CENTRAL へのビジネスアセットプロジェクトのインポートおよびデプロイ	36
第10章 JMS メッセージブローカーで監査データの複製	38
10.1. SPRING BOOT JMS 監査レプリケーションパラメーター	40
パート II. RED HAT DECISION MANAGER と RED HAT FUSE の統合	42
第11章 RED HAT FUSE および RED HAT DECISION MANAGER	43
第12章 APACHE KARAF 上に FUSE を統合した RED HAT DECISION MANAGER のデシジョンエンジン	44
12.1. KARAF 上の古くなった RED HAT DECISION MANAGER 機能 XML ファイルの削除	44
12.2. XML ファイルを使用した KARAF への RED HAT DECISION MANAGER 機能のインストール	45
12.3. MAVEN を使用した KARAF への RED HAT DECISION MANAGER 機能のインストール	46
12.4. RED HAT DECISION MANAGER の KARAF 機能	47
第13章 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM への FUSE のインストール	49
第14章 KIE-CAMEL コンポーネント	51
パート III. RED HAT DECISION MANAGER と RED HAT SINGLE SIGN-ON の統合	52

第15章 統合オプション	53
第16章 RH-SSO のインストールおよび設定	54
第17章 RED HAT DECISION MANAGER ロールおよびユーザー	55
17.1. RED HAT DECISION MANAGER ユーザーの追加	55
第18章 RH-SSO を使用した BUSINESS CENTRAL の認証	57
18.1. RH-SSO への BUSINESS CENTRAL クライアントの作成	57
18.2. BUSINESS CENTRAL への RH-SSO クライアントアダプターのインストール	58
18.3. RH-SSO による BUSINESS CENTRAL の外部ファイルシステムおよび GIT リポジトリサービスへのアクセスを可能にする	62
第19章 RH-SSO を使用した KIE SERVER の認証	64
19.1. RH-SSO で KIE SERVER クライアントの作成	64
19.2. クライアントアダプターを使用する KIE SERVER のインストールおよび設定	65
19.3. KIE SERVER のトークンベースの認証	67
第20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	69
20.1. BASIC 認証	69
20.2. トークンベースの認証	69
付録A バージョン情報	71
付録B お問い合わせ先	72

はじめに

開発者またはシステム管理者は、Spring Boot、Red Hat Single Sign-On などの他の製品やコンポーネントと Red Hat Decision Manager を統合できます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みにより、これらの変更は今後の複数のリリースに対して段階的に実施されます。詳細は、[弊社の CTO である Chris Wright のメッセージ](#)を参照してください。

パート I. SPRING BOOT での RED HAT DECISION MANAGER ビジネスアプリケーションの作成

開発者は、[business application](#) の Web サイトから Spring Boot スターターを使用して、Red Hat Decision Manager ビジネスアプリケーションをすばやく作成して設定し、既存のサービスやクラウドにデプロイできます。

第1章 RED HAT DECISION MANAGER SPRING BOOT ビジネスアプリケーション

Spring Framework は、Java アプリケーション開発用に包括的なインフラストラクチャーサポートを提供する Java プラットフォームです。Spring Boot は、Spring Boot スターターをベースにした、軽量フレームワークです。Spring Boot スターターは、**pom.xml** ファイルで、このファイルには Spring Boot プロジェクトに追加可能な依存関係の記述子セットが含まれます。

Red Hat Decision Manager Spring Boot ビジネスアプリケーションは、特定のビジネス機能を提供する個別サービスを柔軟に、UI に依存せず、論理的にグループ化します。ビジネスアプリケーションは、Spring Boot スターターをベースにしています。通常、個別にデプロイされ、個別にバージョン管理できます。完全なビジネスアプリケーションを使用すると、ドメインは特定のビジネスゴール (たとえば、順序管理や補助管理など) を達成できます。

[business application](#) の Web サイトで、Process Automation Manager、Decision Manager、または Red Hat ビルドの OptaPlanner ビジネスアプリケーションを作成できます。ビジネスアプリケーションを作成して設定した後は、OpenShift で、既存のサービスやクラウドにデプロイできます。

ビジネスアプリケーションには、以下のプロジェクト1つ以上と、同じ型のプロジェクトを複数含めることができます。

- ビジネスアセット (KJAR): ビジネスプロセス、ルール、フォームが含まれており、Business Central へのインポートが簡単です。
- データモデル: データモデルのプロジェクトでは、サービスプロジェクトやビジネスアセットプロジェクト間で共有される、共通のデータ構造を提供しています。これにより、適切なカプセル化、再利用の促進、ショートカットの軽減が可能になります。各サービスプロジェクトは、独自の公開データモデルを公開できます。
- サービス: 実際のサービスにさまざまな機能を提供する、展開可能なプロジェクト。これには、ビジネスを管理するビジネスロジックが含まれます。多くの場合に、サービスプロジェクトにはビジネスアセットとデータモデルプロジェクトが含まれます。ビジネスアプリケーションは、サービスをより小さなコンポーネントサービスプロジェクトに分割して、管理性を向上できます。

第2章 ビジネスアプリケーションの作成

[business application](#) の Web サイトで、Spring Boot フレームワークを使用してすばやく簡単にビジネスアプリケーションを作成できます。これにより、Red Hat Decision Manager のインストールと設定の必要性がなくなります。

手順

1. Web ブラウザーで、以下の URL を入力します。

<https://start.jbpm.org>

2. **Configure your business application** をクリックします。
3. **Decision Management** をクリックし、**Next** をクリックします。
4. パッケージとアプリケーション名を入力します。
5. **Version** メニューから **Enterprise 7.12** を選択し、**Next** をクリックします。



注記

Enterprise 7.12 を選択して、Red Hat Decision Manager ビジネスアプリケーションを作成する必要があります。

6. プロジェクトに追加するプロジェクトの種類を選択します。プロジェクトタイプは、複数追加できます。
 - **ビジネスアセット**: ビジネスプロセス、ルール、フォームが含まれており、Business Central に簡単にインポートできます。また、ケースなどのように適応性があり、動的なアセットを追加する場合は、**動的アセット** を使用します。
 - **データモデル**: サービスプロジェクトやビジネスアセットプロジェクト間で共有される、共通のデータ構造を提供します。これにより、適切なカプセル化、再利用の促進、ショートカットの軽減が可能になります。各サービスプロジェクトは、独自の公開データモデルを公開できます。
 - **サービス**: ビジネスを動かすビジネスロジックが含まれます。
7. **Generate business application** をクリックします。
<BUSINESS-APPLICATION>.zip ファイルがダウンロードされます。**<BUSINESS-APPLICATION>** は Application Name のボックスで入力した名前に置き換えてください。
8. **<BUSINESS-APPLICATION>.zip** ファイルを展開します。

第3章 APACHE MAVEN および RED HAT DECISION MANAGER SPRING BOOT アプリケーション

Apache Maven は分散型構築自動化ツールで、ソフトウェアプロジェクトの作成、ビルド、および管理を行うために Java アプリケーション開発で使用されます。Maven は Project Object Model (POM) ファイルと呼ばれる標準の設定ファイルを使用して、プロジェクトの定義や構築プロセスの管理を行います。POM ファイルは、モジュールおよびコンポーネントの依存関係、ビルドの順番、結果となるプロジェクトパッケージのターゲットを記述し、XML ファイルを使用して出力します。これにより、プロジェクトが適切かつ統一された状態でビルドされるようになります。

Maven リポジトリには、Java ライブラリー、プラグイン、およびその他のビルドアーティファクトが格納されます。デフォルトのパブリックリポジトリは Maven 2 Central Repository ですが、複数の開発チームの間で共通のアーティファクトを共有する目的で、社内のプライベートおよび内部リポジトリとすることが可能です。また、サードパーティーのリポジトリも利用できます。

Spring Boot プロジェクトでオンライン Maven リポジトリを使用するか、Red Hat Decision Manager Maven リポジトリをダウンロードできます。Spring Boot プロジェクトでオンライン Maven リポジトリを使用することが推奨されます。リポジトリマネージャーまたは共有サーバー上のリポジトリと使用する Maven 設定は、プロジェクトの制御および管理性を向上させます。

3.1. オンライン MAVEN リポジトリ用の RED HAT DECISION MANAGER SPRING BOOT プロジェクトの設定

Red Hat Decision Manager Spring Boot プロジェクトを作成したら、アプリケーションデータを保存するためにオンライン Maven リポジトリで設定します。

前提条件

- Red Hat Decision Manager Spring Boot プロジェクトを作成している。

手順

- Red Hat Decision Manager Spring Boot アプリケーションが含まれるディレクトリで、テキストエディターまたは IDE で **<BUSINESS-APPLICATION>-service/pom.xml** ファイルを開きます。**<BUSINESS-APPLICATION>** は Spring Boot プロジェクトの名前に置き換えてください。
- 以下のリポジトリを **repositories** 要素に追加します。

```
<repository>
  <id>jboss-enterprise-repository-group</id>
  <name>Red Hat JBoss Enterprise Maven Repository</name>
  <url>https://maven.repository.redhat.com/ga/</url>
  <layout>default</layout>
  <releases>
    <updatePolicy>never</updatePolicy>
  </releases>
  <snapshots>
    <updatePolicy>daily</updatePolicy>
  </snapshots>
</repository>
```

- 以下のプラグインを **pluginRepositories** 要素に追加します。



注記

pom.xml ファイルに **pluginRepositories** 要素がない場合は、これも追加します。

```
<pluginRepository>
  <id>jboss-enterprise-repository-group</id>
  <name>Red Hat JBoss Enterprise Maven Repository</name>
  <url>https://maven.repository.redhat.com/ga/</url>
  <layout>default</layout>
  <releases>
    <updatePolicy>never</updatePolicy>
  </releases>
  <snapshots>
    <updatePolicy>daily</updatePolicy>
  </snapshots>
</pluginRepository>
```

これにより、お使いのビジネスアプリケーションに、製品化した Maven リポジトリが追加されます。

3.2. RED HAT DECISION MANAGER MAVEN リポジトリのダウンロードおよび設定

オンライン Maven リポジトリを使用しない場合は、Red Hat Decision Manager Maven リポジトリをダウンロードして設定できます。Red Hat Decision Manager Maven リポジトリには、Java 開発者がアプリケーションの構築に使用する要件の多くが含まれています。この手順では、Maven の **settings.xml** ファイルを編集し、Red Hat Decision Manager Maven リポジトリを設定する方法を説明します。



注記

Maven の **settings.xml** ファイルを変更してリポジトリを設定する場合、変更はすべての Maven プロジェクトに適用されます。

前提条件

- Red Hat Decision Manager Spring Boot プロジェクトを作成している。

手順

- 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以下の製品およびバージョンを選択します。
 - Product: Decision Manager
 - バージョン: 7.12
- Red Hat Decision Manager 7.12 Maven リポジトリ (**rhdm-7.12.0-maven-repository.zip**) をダウンロードします。
- ダウンロードしたアーカイブを展開します。

4. ~/.m2/ ディレクトリーに移動し、テキストエディターまたは統合開発環境 (IDE) で Maven の **settings.xml** ファイルを開きます。
5. 以下の行を Maven **settings.xml** ファイルの **<profiles>** 要素に追加します。ここで、**<MAVEN_REPOSITORY>** はダウンロードした Maven リポジトリーのパスです。**<MAVEN_REPOSITORY>** の形式は **file://\$PATH** の形式 (例: **file:///home/userX/rhdm-7.12.0.GA-maven-repository/maven-repository**) にする必要があります。

```
<profile>
  <id>red-hat-enterprise-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-enterprise-maven-repository</id>
      <url><MAVEN_REPOSITORY></url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

6. 以下の行を Maven の **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

```
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```



重要

Maven リポジトリに古いアーティファクトが含まれる場合は、プロジェクトをビルドまたはデプロイしたときに以下のいずれかの Maven エラーメッセージが表示されることがあります。ここで、**<ARTIFACT_NAME>** は不明なアーティファクトの名前で、**<PROJECT_NAME>** は構築を試みているプロジェクトの名前になります。

- **Missing artifact <PROJECT_NAME>**
- **[ERROR] Failed to execute goal on project <ARTIFACT_NAME>; Could not resolve dependencies for <PROJECT_NAME>**

この問題を解決するには、**~/.m2/repository** ディレクトリーにあるローカルリポジトリーのキャッシュバージョンを削除し、最新の Maven アーティファクトを強制的にダウンロードします。

第4章 RED HAT DECISION MANAGER での SPRING SECURITY

Spring Security は、[Spring Security ライブラリー](#) を設定するサーブレットフィルターのコレクションによって提供されます。これらのフィルターは、ユーザー名とパスワードを使用した認証およびロールを使用した承認を行います。Red Hat Decision Manager Spring Boot アプリケーションで生成されるデフォルトの Spring Security 実装は、認証なしで承認を行います。つまり、アプリケーションに対して有効なユーザー名とパスワードを持つユーザーは、ロールなしでアプリケーションにアクセスできます。

サーブレットフィルターは、CSRF (Cross-Site Request Forgery) や CORS (Cross-Origin Resource Sharing) などの一般的な脆弱性から Spring Boot アプリケーションを保護します。Spring Web は [DispatcherServlet](#) に依存し、受信した HTTP 要求を [@Controller](#) アノテーションが付けられた基盤となる java REST リソースにリダイレクトします。[DispatchServlet](#) は、セキュリティなどの要素に依存しません。ビジネスアプリケーションロジックの外部でセキュリティなどの実装の詳細を処理することは、優れた方法であり、より効率的です。そのため、Spring はフィルターを使用して HTTP 要求を傍受してから、それらを [DispatchServlet](#) にルーティングします。

典型的な Spring Security 実装は、複数のサーブレットフィルターを使用する以下の手順で設定されます。

1. HTTP 要求からユーザー認証情報を抽出し、デコードまたは復号します。
2. データベース、Web サービス、Red Hat Single Sign-On などの企業アイデンティティプロバイダーに対して認証情報を検証することにより、完全な認証を行います。
3. 承認されたユーザーが要求を実行する権限を持っているかどうかを決定することにより、完全な承認を行います。
4. ユーザーが認証および承認された場合は、要求を [DispatchServlet](#) に伝播します。

Spring はこれらの手順を個別のフィルターに分割し、FilterChain で連結させます。この連鎖メソッドは、ほとんどすべてのアイデンティティプロバイダーとセキュリティフレームワークと連携するために必要な柔軟性を提供します。Spring Security では、プログラムを使用してアプリケーションの FilterChain を定義できます。以下のセクションは、Web サイト <https://start.jbpm.org> で作成されたビジネスアプリケーションの一部として生成された `business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java` ファイルからのものです。

```
@Configuration("kieServerSecurity")
@EnableWebSecurity
public class DefaultWebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override (1)
    protected void configure(HttpSecurity http) throws Exception {
        http
            .cors().and()
            .csrf().disable() (2)
            .authorizeRequests() (3)
            .antMatchers("/rest/*").authenticated().and()
            .httpBasic().and() (4)
            .headers().frameOptions().disable(); (5)
    }
}
```

- (1) デフォルトの `configure(HttpSecurity http)` メソッドをオーバーライドし、Spring HttpClient fluent API/DSL を使用してカスタムの FilterChain を定義します。

- (2) ローカルテストにおける CORS トークンおよび CSRF トークンの一般的な脆弱性フィルターを無効にします。
- (3) パターン `rest/*` に送信された要求には認証が必要ですが、ロールは定義されていません。
- (4) 承認ヘッダーによる Basic 認証を許可します (ヘッダー `Authorization: Basic dGVzdF91c2VyOnBhc3N3b3Jk` など)。
- (5) 要求/レスポンスから `X-Frame-Options` ヘッダーを削除します。

この設定により、認証されたユーザーは KIE API を実行できます。

デフォルトの実装は外部アイデンティティプロバイダーに統合されないため、ユーザーはメモリー内の同じ **DefaultWebSecurityConfig** クラスで定義されます。以下のセクションは、Red Hat Decision Manager Spring Boot ビジネスアプリケーションの作成時に提供されるユーザーを示しています。

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().withUser("user").password("user").roles("kie-server");
    auth.inMemoryAuthentication().withUser("wbadmin").password("wbadmin").roles("admin");
    auth.inMemoryAuthentication().withUser("kieserver").password("kieserver1").roles("kie-server");
}
```

4.1. SPRING SECURITY を使用した承認による認証

デフォルトでは、Red Hat Decision Manager Spring Boot アプリケーションに対して有効なユーザー名とパスワードを持つユーザーは、ロールなしでアプリケーションにアクセスできます。Spring Security の認証および承認は、**HTTPSecurity** フィルターチェーン設定から取得します。特定のロールマッピングを持たないユーザーから REST API を保護するには、Spring Security **.authorizeRequests()** メソッドを使用して、承認する URL を一致させます。

前提条件

- Red Hat Decision Manager Spring Boot アプリケーションがある。

手順

1. Red Hat Decision Manager Spring Boot アプリケーションが含まれるディレクトリーで、テキストエディターまたは IDE で **business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java** ファイルを開きます。
2. 特定のロールがある場合にのみ認証されたユーザーによるアクセス要求を承認するには、以下のいずれかの方法で **.antMatchers("/rest/*").authenticated().and()** 行を編集します。
 - 1つのロールを承認するには、以下の例のように **antMatchers** メソッドを編集します。ここで、**<role>** はユーザーがアクセスするために必要なロールに置き換えます。

```
@Configuration("kieServerSecurity")
@EnableWebSecurity
public class DefaultWebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
```

```

        .cors().and().csrf().disable()
        .authorizeRequests()
            .antMatchers("/**").hasRole("<role>")
            .anyRequest().authenticated()
        .and().httpBasic()
        .and().headers().frameOptions().disable();
    }
    ...

```

- さまざまなロールのうちの1つを持つユーザーを承認するには、以下の例のように **antMatchers** メソッドを編集します。ここで、**<role>** および **<role1>** はそれぞれ、ユーザーがアクセスするために持つことができるロールです。

```

@Configuration("kieServerSecurity")
@EnableWebSecurity
public class DefaultWebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .cors().and().csrf().disable()
            .authorizeRequests()
                .antMatchers("/**").hasAnyRole("<role>", "<role1>")
                .anyRequest().authenticated()
            .and().httpBasic()
            .and().headers().frameOptions().disable();
    }
    ...

```

authorizeRequests メソッドでは、特定の式に対する要求の承認が必要です。すべての要求が正常に認証されている必要があります。認証は、HTTP Basic 認証を使用して実行されます。認証されたユーザーが、自分では持っていないロールに対して保護されているリソースにアクセスしようとする、そのユーザーは **HTTP 403 (Forbidden)** エラーを受け取ります。

4.2. RED HAT DECISION MANAGER ビジネスアプリケーションでの SPRING SECURITY の無効化

Red Hat Decision Manager ビジネスアプリケーションで Spring Security を設定して、認証なしでセキュリティコンテキストを提供することができます。

前提条件

- Red Hat Decision Manager Spring Boot アプリケーションがある。

手順

1. Red Hat Decision Manager Spring Boot アプリケーションが含まれるディレクトリーで、テキストエディターまたは統合開発環境 (IDE) で **business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java** ファイルを開きます。
2. 以下の例で示すように **.antMatchers** メソッドを編集します。

```

@Override

```

```
protected void configure(HttpSecurity http) throws Exception {

    http
        .cors().and().csrf().disable()
        .authorizeRequests()
        .antMatchers("/*")
            .permitAll()
        .and().headers().frameOptions().disable();

}
```

PermitAll メソッドは、指定された URL パターンに対するすべての要求を許可します。

注記

HttpServletRequest にセキュリティコンテキストが渡されないため、Spring は **AnonymousAuthenticationToken** を作成し、**ROLE_ANONYMOUS** ロール以外の指定されたロールを持たない **anonymousUser** ユーザーで **SecurityContext** を追加します。ユーザーは、アプリケーションの多くの機能にアクセスできません。たとえば、割り当てられたタスクをグループ化するためにアクションを割り当てることができなくなります。

4.3. 事前認証での SPRING セキュリティーの使用

PermitAll メソッドを使用して Spring Security 認証を無効にすると、すべてのユーザーはアプリケーションにログインできますが、ユーザーのアクセスと機能は限定されます。ただし、ユーザーを事前に認証できるため (指定されたサービスアカウントなど)、ユーザーのグループは同じログインを使用できますが、ユーザーのグループに必要なすべてのパーミッションが付与されます。このため、各ユーザーに認証情報を作成する必要はありません。

事前認証を実装する最も簡単な方法は、カスタムフィルターサーブレットを作成し、**DefaultWebSecurityConfig** クラスのセキュリティ FilterChain の前に追加することです。これにより、カスタマイズされたプロファイルベースのセキュリティコンテキストを挿入し、そのコンテンツを制御して、シンプルに保つことができます。

前提条件

- Red Hat Decision Manager Spring Boot アプリケーションがあり、Spring Security を「[Red Hat Decision Manager ビジネスアプリケーションでの Spring Security の無効化](#)」に従って無効化している。

手順

1. **AnonymousAuthenticationFilter** クラスを拡張する以下のクラスを作成します。

```
import org.springframework.security.authentication.AnonymousAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.AnonymousAuthenticationFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
```

```

import javax.servlet.ServletException;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class <CLASS_NAME> extends AnonymousAuthenticationFilter {
    private static final Logger log = LoggerFactory.getLogger(<CLASS_NAME>.class);

    public AnonymousAuthFilter() {
        super("PROXY_AUTH_FILTER");
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
        throws IOException, ServletException {

        SecurityContextHolder.getContext().setAuthentication(createAuthentication((HttpServletRequest) req));
        log.info("SecurityContextHolder pre-auth user: {}", SecurityContextHolder.getContext());

        if (log.isDebugEnabled()) {
            log.debug("Populated SecurityContextHolder with authenticated user: {}",
                SecurityContextHolder.getContext().getAuthentication());
        }

        chain.doFilter(req, res);
    }

    @Override
    protected Authentication createAuthentication(final HttpServletRequest request)
        throws AuthenticationException {

        log.info("<ANONYMOUS_USER>");

        List<? extends GrantedAuthority> authorities = Collections
            .unmodifiableList(Arrays.asList(new SimpleGrantedAuthority("<ROLE>")
                ));
        return new AnonymousAuthenticationToken("ANONYMOUS", "<ANONYMOUS_USER>", authorities);
    }
}

```

2. 以下の変数を置き換えてください。

- **<CLASS_NAME>** を、このクラスの名前 (**AnonymousAuthFilter** など) に置き換えます。
- **<ANONYMOUS_USER>** をユーザー ID (**Service_Group** など) に置き換えます。
- **<ROLE>** を **<ANONYMOUS_USER>** に付与する必要がある権限を持つロールに置き換えます。

3. **<ANONYMOUS_USER>** に複数のロールを割り当てる必要がある場合は、以下の例に示すようにロールを追加します。

```
.unmodifiableList(Arrays.asList(new SimpleGrantedAuthority("<ROLE>")
, new SimpleGrantedAuthority("<ROLE2>"))
```

4. `.anonymous().authenticationFilter(new <CLASS_NAME>()).and()` を `business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java` ファイルに追加します。**<CLASS_NAME>** は作成したクラス名に置き換えます。

```
@Override
protected void configure(HttpSecurity http) throws Exception {

    http
        .anonymous().authenticationFilter(new <CLASS_NAME>()).and() // Override
        anonymousUser
            .cors().and().csrf().disable()
            .authorizeRequests()
            .antMatchers("/*").permitAll()
            .and().headers().frameOptions().disable();

}
```

4.4. RED HAT SINGLE SIGN-ON を使用したビジネスアプリケーションの設定

ほとんどの組織は、シングルサインオン (SSO) トークンを使用して、ユーザーおよびグループの詳細を提供します。Red Hat Single Sign-On (RHSSO) を使用して、サービス間のシングルサインオンを有効にし、一元的にユーザーとロールの設定や管理ができます。

前提条件

- [business applications](#) の Web サイトを使用して作成した Spring Boot アプリケーションの ZIP ファイルがある。

手順

1. Red Hat シングルサインオン (SSO) をダウンロードします。手順については、[Red Hat Single Sign-On Getting Started Guide](#) を参照してください。
2. RHSSO を設定します。
 - a. デフォルトのマスターレルムを使用するか、新しいレルムを作成します。
レルムは、一連のユーザー、認証情報、ロール、およびグループを管理します。ユーザーはレルムに属し、レルムにログインします。レルムは相互に分離され、制御するユーザーのみを管理および認証できます。
 - b. **springboot-app** クライアントを作成して、パブリックに **AccessType** を追加します。
 - c. 以下の例で示すように、ローカルの設定に合わせて有効なリダイレクト URI と Web オリジンを設定します。
 - 有効なリダイレクト URI: **http://localhost:8090/***

- Web オリジン: **http://localhost:8090**
 - d. アプリケーションで使用するレルムロールを作成します。
 - e. アプリケーションで使用するユーザーを作成してロールを割り当てます。
3. 以下の要素およびプロパティを Spring Boot プロジェクトの **pom.xml** ファイルに追加します。ここで、**<KEYCLOAK_VERSION>** は使用している Keycloak のバージョンに置き換えます。

```
<properties>
  <version.org.keycloak><KEYCLOAK_VERSION></version.org.keycloak>
</properties>
```

4. 以下の依存関係を Spring Boot プロジェクトの **pom.xml** ファイルに追加します。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.keycloak.bom</groupId>
      <artifactId>keycloak-adapter-bom</artifactId>
      <version>${version.org.keycloak}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

....

<dependency>
  <groupId>org.keycloak</groupId>
  <artifactId>keycloak-spring-boot-starter</artifactId>
</dependency>
```

5. Spring Boot プロジェクトディレクトリーで **business-application-service/src/main/resources/application.properties** ファイルを開き、以下の行を追加します。

```
# keycloak security setup
keycloak.auth-server-url=http://localhost:8100/auth
keycloak.realm=master
keycloak.resource=springboot-app
keycloak.public-client=true
keycloak.principal-attribute=preferred_username
keycloak.enable-basic-auth=true
```

6. **business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java** ファイルを変更して、Spring Security が RHSSO で正しく機能することを確認します。

```
import org.keycloak.adapters.KeycloakConfigResolver;
import org.keycloak.adapters.springboot.KeycloakSpringBootConfigResolver;
import org.keycloak.adapters.springsecurity.authentication.KeycloakAuthenticationProvider;
import org.keycloak.adapters.springsecurity.config.KeycloakWebSecurityConfigurerAdapter;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerB
uilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.authority.mapping.SimpleAuthorityMapper;
import org.springframework.security.core.session.SessionRegistryImpl;
import
org.springframework.security.web.authentication.session.RegisterSessionAuthenticationStrateg
y;
import
org.springframework.security.web.authentication.session.SessionAuthenticationStrategy;

@Configuration("kieServerSecurity")
@EnableWebSecurity
public class DefaultWebSecurityConfig extends KeycloakWebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http
            .csrf().disable()
            .authorizeRequests()
                .anyRequest().authenticated()
                .and()
            .httpBasic();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        KeycloakAuthenticationProvider keycloakAuthenticationProvider =
keycloakAuthenticationProvider();
        SimpleAuthorityMapper mapper = new SimpleAuthorityMapper();
        mapper.setPrefix("");
        keycloakAuthenticationProvider.setGrantedAuthoritiesMapper(mapper);
        auth.authenticationProvider(keycloakAuthenticationProvider);
    }

    @Bean
    public KeycloakConfigResolver KeycloakConfigResolver() {
        return new KeycloakSpringBootConfigResolver();
    }

    @Override
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
    }
}

```


第5章 RED HAT DECISION MANAGER SPRING BOOT の設定

Spring Boot プロジェクトの作成後、複数のコンポーネントを設定してアプリケーションをカスタマイズできます。

5.1. SPRING BOOT アプリケーションの REST エンドポイントの設定

Spring Boot プロジェクトを作成したら、Spring Boot アプリケーションの REST エンドポイントのホスト、ポート、およびパスを設定できます。

前提条件

- [business application](#) の Web サイトを使用して作成した ZIP ファイルがある。

手順

1. Spring Boot プロジェクトの ZIP ファイルを **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources** フォルダーに展開します。**<BUSINESS-APPLICATION>** は Spring Boot プロジェクトの名前に置き換えます。
2. テキストエディターで **application.properties** ファイルを開きます。
3. REST エンドポイントのホスト、ポート、およびパスを設定します。**<ADDRESS>** はサーバーアドレスに、**<PORT>** はサーバーポートに置き換えます。

```
server.address=<ADDRESS>
server.port=<PORT>
cxf.path=/rest
```

以下の例では、REST エンドポイントをポート **8090** の **localhost** アドレスに追加します。

```
server.address=localhost
server.port=8090
cxf.path=/rest
```

5.2. KIE SERVER アイデンティティの設定

Spring Boot プロジェクトを作成したら、簡単に特定できるように KIE Server を設定できます。

前提条件

- [business application](#) の Web サイトを使用して作成した Spring Boot ビジネスアプリケーションの ZIP ファイルがある。

手順

1. Spring Boot プロジェクトの ZIP ファイルを **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources** フォルダーに展開します。**<BUSINESS-APPLICATION>** は Spring Boot プロジェクトの名前に置き換えます。
2. テキストエディターで **application.properties** ファイルを開きます。
3. 以下の例のように KIE Server パラメーターを設定します。

```
kieserver.serverId=<BUSINESS-APPLICATION>-service
kieserver.serverName=<BUSINESS-APPLICATION>-service
kieserver.location=http://localhost:8090/rest/server
kieserver.controllers=http://localhost:8080/decision-central/rest/controller
```

以下の表で、ビジネスプロジェクトに設定可能な KIE Server のパラメーターを紹介します。

表5.1 kieserver パラメーター

パラメーター	値	説明
kieserver.serverId	string	Decision Manager コントローラーに接続時にビジネスアプリケーションを識別するために使用する ID。
kieserver.serverName	string	Decision Manager コントローラーへの接続時にビジネスアプリケーションを識別するために使用する名前。 kieserver.serverId パラメーターに使用した文字と同じものを使用できます。
kieserver.location	URL	REST API を使用する他のコンポーネントがこのサーバーの場所を識別するために使用します。 server.address および server.port で定義されている場所は使用しないでください。
kieserver.controllers	URL	コントローラー URL のコンマ区切りリスト

5.3. ランタイム時に起動する KIE SERVER コンポーネントの設定

Spring Boot ビジネスアプリケーションの作成時に **Business Automation** を選択した場合は、ランタイムに起動する必要のある KIE Server のコンポーネントを指定することができます。

前提条件

- [business application](#) の Web サイトを使用して作成した Spring Boot ビジネスアプリケーションの ZIP ファイルがある。

手順

1. Spring Boot プロジェクトの ZIP ファイルを **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources** フォルダーに展開します。**<BUSINESS-APPLICATION>** は Spring Boot プロジェクトの名前に置き換えます。
2. テキストエディターで **application.properties** ファイルを開きます。
3. ランタイムにコンポーネントが起動するように設定するには、コンポーネントの値を **true** に設定します。
以下の表は、ランタイムに起動するように設定できるコンポーネントの一覧です。

表5.2 kieserver ケーパビリティパラメーター

パラメーター	値	説明
kieserver.drools.enabled	true, false	Decision Manager コンポーネントを有効または無効にします。
kieserver.dmn.enabled	true, false	DMN (Decision Model and Notation) コンポーネントを有効または無効にします。

5.4. ビジネスアプリケーションのユーザーグループプロバイダーの設定

Red Hat Decision Manager を使用すると、人間中心のアクティビティを管理できます。2 つの KIE API エントリーポイントを使用して、ユーザーリポジトリおよびグループリポジトリを統合できます。

- **UserGroupCallback**: ユーザーまたはグループが存在するかどうかを確認して、特定のユーザーのグループの情報を集めます。
- **UserInfo**: メールアドレスや設定言語など、ユーザーおよびグループの追加情報を収集します。

すぐに使用できるコードまたはカスタムの開発コードなど、代替りのコードを指定すると、これらの両コンポーネントを設定できます。

UserGroupCallback コンポーネントについては、デフォルトの実装はアプリケーションのセキュリティコンテキストをもとにしているため、この実装が保持されます。このように、どのバックエンドストアを認証や承認 (例: RH-SSO) に使用するかは重要ではありません。ユーザーやグループの情報収集の情報源として、この実装が自動的に使用されます。

UserInfo コンポーネントは、より詳細にわたる情報を収集するため、別のコンポーネントとなっています。

前提条件

- [business application](#) の Web サイトを使用して作成した ZIP ファイルがあり、このファイルにはビジネス自動化プロジェクトが含まれている。

手順

1. **UserGroupCallback** の別の実装を提供するには、以下のコードを、アプリケーションクラスに追加するか、**@Configuration** のアノテーションが付いた別のクラスに追加します。

```
@Bean(name = "userGroupCallback")
public UserGroupCallback userGroupCallback(IdentityProvider identityProvider) throws
IOException {
    return new MyCustomUserGroupCallback(identityProvider);
}
```

2. **UserInfo** の別の実装を提供するには、以下のコードを、アプリケーションクラスに追加するか、**@Configuration** のアノテーションが付いた別のクラスに追加します。

```
@Bean(name = "userInfo")
public UserInfo userInfo() throws IOException {
    return new MyCustomUserInfo();
}
```

```

| }

```

5.5. SWAGGER ドキュメントの有効化

Red Hat Decision Manager ビジネスアプリケーションのサービスプロジェクトで利用可能なすべてのエンドポイントに関する Swagger ベースのドキュメントを有効にできます。

前提条件

- [business applications](#) の Web サイトを使用して作成した Spring Boot アプリケーションの ZIP ファイルがある。

手順

1. **<BUSINESS-APPLICATION>.zip** ファイルを展開して、**<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service** ディレクトリーに移動します。**<BUSINESS-APPLICATION>** は Spring Boot プロジェクトの名前に置き換えます。
2. テキストエディターでサービスプロジェクト **pom.xml** ファイルを開きます。
3. サービスプロジェクトの **pom.xml** ファイルに以下の依存関係を追加して、このファイルを保存します。

```

<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-rs-service-description-swagger</artifactId>
  <version>3.2.6</version>
</dependency>
<dependency>
  <groupId>io.swagger</groupId>
  <artifactId>swagger-jaxrs</artifactId>
  <version>1.5.15</version>
  <exclusions>
    <exclusion>
      <groupId>javax.ws.rs</groupId>
      <artifactId>jsr311-api</artifactId>
    </exclusion>
  </exclusions>
</dependency>

```

4. Swagger UI を有効にするには (任意)、以下の依存関係を **pom.xml** ファイルに追加して、このファイルを保存します。

```

<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>swagger-ui</artifactId>
  <version>2.2.10</version>
</dependency>

```

5. テキストエディターで **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources/application.properties** ファイルを開きます。
6. Swagger サポートを有効にするには、以下の行を **application.properties** ファイルに追加します。

kieserver.swagger.enabled=true

ビジネスアプリケーションの起動後に、**<http://localhost:8090/rest/swagger.json>** で Swagger ドキュメントを表示できます。全エンドポイントについては、**<http://localhost:8090/rest/api-docs?url=http://localhost:8090/rest/swagger.json>** で入手できます。

第6章 自己完結型 RED HAT DECISION MANAGER SPRING BOOT JAR ファイルの作成

KIE Server および 1 つ以上の KJAR ファイルなど、完全なサービスを含む、自己完結型 Red Hat Decision Manager Spring Boot JAR ファイル 1 つを作成できます。Red Hat Decision Manager Spring Boot JAR ファイルは、ランタイム時に読み込まれる KJAR ファイルには依存しません。

必要に応じて、Red Hat Decision Manager Spring Boot JAR ファイルには、モジュールを含む、同じ KJAR ファイルの複数のバージョンを含めることができます。これらの KJAR ファイルは、**artifactID** 属性値と **groupId** 属性値が同じですが、**version** の値は異なります。

含める KJAR ファイルは、クラ出力ダーの衝突を避けるために **BOOT-INF/lib** ディレクトリーの JAR ファイルから分離されています。各 KJAR クラスパスコンテナファイルは、他の KJAR クラスパスコンテナファイルから分離され、Spring Boot クラ出力ダーに依存しません。

前提条件

- 既存の Red Hat Decision Manager Spring Boot プロジェクトがある。
- プロジェクトの KJAR ファイル 1 つ以上の開発を完了している。

手順

1. プロジェクトの KJAR ファイルをすべてビルドします。デフォルトのビジネスアプリケーションでは、KJAR ソースは **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-kjar** ディレクトリーに含まれます。**BUSINESS-APPLICATION** はビジネスアプリケーションの名前に置き換えます。プロジェクトには、他の KJAR ソースディレクトリーが含まれている可能性があります。

すべての KJAR ソースのディレクトリーの KJAR ファイルをビルドするには、以下の手順を実行します。

- a. KJAR ソースディレクトリーに移動します。
- b. 以下のコマンドを入力します。

```
mvn install
```

このコマンドは、KJAR ファイルをビルドし、ローカルの Maven リポジトリーに配置します。デフォルトでは、このリポジトリーは **~/.m2/repo** ディレクトリーに配置されます。

2. **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources** ディレクトリーで、Spring Boot アプリケーションの **application.properties** ファイルに以下のプロパティーを追加します。

```
kieserver.classPathContainer=true
```

このプロパティーを **true** に設定すると、KIE Server はコンテナが使用するクラ出力ダーを使用して KJAR ファイルとその依存関係を読み込みます。

3. KIE Server が必要な KJAR モジュールを読み込むには、以下のいずれかのアクションを実行します。
 - KIE Server を設定して、Spring Boot アプリケーションで利用可能なすべての KJAR モジュールをスキャンし、デプロイするには、以下のプロパティーを **application.properties** ファイルに追加します。

```
kieserver.autoScanDeployments=true
```

このプロパティを **true** に設定すると、プログラムを使用して宣言されているか、Maven プラグインを介して宣言されているかに関係なく、KIE Server はアプリケーションで利用可能なすべての KJAR モジュールをデプロイします。

このオプションは、すべての KJAR モジュールを含める最も簡単な方法です。ただし、欠点が2つあります。

- アプリケーションは、すべての KJAR モジュールのグループ、アーティファクト、バージョン (GAV) に基づいて、すべてのコンテナ ID およびエイリアスを自動的に設定します。KJAR モジュールのカスタムコンテナ ID またはエイリアスを設定できません。
- 起動時に、アプリケーションは JAR ファイルおよび KJAR モジュールのクラスパスをスキャンします。そのため、起動期間は長くなる可能性があります。

このような欠点を回避するには、以下のいずれかのオプションで説明されているように **application.properties** ファイルを使用するか、Java ソースコードを使用して、すべての KJAR モジュールを個別に設定できます。

- サービスに追加する KJAR モジュールごとに、**application.properties** ファイルを使用して、すべての KJAR モジュールを個別に設定するには、以下のプロパティを **application.properties** ファイルに追加します。

```
kieserver.deployments[<n>].containerId=<container>
kieserver.deployments[<n>].alias=<alias>
kieserver.deployments[<n>].artifactId=<artifact>
kieserver.deployments[<n>].groupId=<group>
kieserver.deployments[<n>].version=<version>
```

以下の値を置き換えます。

- **<n>**: 連番: 1 番目の KJAR モジュールは **0**、2 番目のモジュールの場合は **1** など。
- **<container>**: KJAR モジュールのコンテナ ID
- **<alias>**: KJAR モジュールのエイリアス
- **<artifact>**: KJAR モジュールのアーティファクト ID
- **<group>**: KJAR モジュールのグループ ID
- **<version>**: KJAR モジュールのバージョン ID

以下の例では、**Evaluation** 用の KJAR モジュールの 2 つのバージョンを設定します。

```
kieserver.deployments[0].alias=evaluation_v1
kieserver.deployments[0].containerId=evaluation_v1
kieserver.deployments[0].artifactId=Evaluation
kieserver.deployments[0].groupId=com.myspace
kieserver.deployments[0].version=1.0.0-SNAPSHOT

kieserver.deployments[1].alias=evaluation_v2
kieserver.deployments[1].containerId=evaluation_v2
```

```
kieserver.deployments[1].artifactId=Evaluation
kieserver.deployments[1].groupId=com.myspace
kieserver.deployments[1].version=2.0.0-SNAPSHOT
```

- Java ソースコードを使用してすべての KJAR モジュールを個別に設定するには、以下の例のように、ビジネスアプリケーションサービスにクラスを作成します。

```
@Configuration
public class KieContainerDeployer {

    @Bean
    public KieContainerResource evaluation_v1() {
        KieContainerResource container = new KieContainerResource("evaluation_v1",
            new ReleaseId("com.myspace", "Evaluation", "1.0.0-SNAPSHOT"), STARTED);
        container.setConfigItems(Arrays.asList(new
            KieServerConfigItem(KieServerConstants.PCFG_RUNTIME_STRATEGY,
            "PER_PROCESS_INSTANCE", "String")));
        return container;
    }

    @Bean
    public KieContainerResource evaluation_v2() {
        KieContainerResource container = new KieContainerResource("evaluation_v2",
            new ReleaseId("com.myspace", "Evaluation", "2.0.0-SNAPSHOT"), STARTED);
        container.setConfigItems(Arrays.asList(new
            KieServerConfigItem(KieServerConstants.PCFG_RUNTIME_STRATEGY,
            "PER_PROCESS_INSTANCE", "String")));
        return container;
    }
}
```

追加するすべての KJAR モジュールについて、このクラスに **KieContainerResource** Bean を作成します。Bean の名前はコンテナ名であり、**KieContainerResource()** の最初のパラメーターはエイリアス名で、**ReleaseId()** のパラメーターは、KJAR モジュールのグループ ID、アーティファクト ID、およびバージョン ID です。

4. 必要に応じて、ビジネスアプリケーションを Red Hat OpenShift Container Platform Pod で実行するか、現在のディレクトリーが書き込み可能ではない他の環境で実行する場合は、**spring.jta.log-dir** プロパティを **application.properties** ファイルに追加し、書き込み可能な場所に設定します。以下に例を示します。

```
spring.jta.log-dir=/tmp
```

このパラメーターは、トランザクションログの場所を設定します。

5. **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service** ディレクトリーの Spring Boot **pom.xml** ファイルに以下の Maven プラグインを追加します。**<GROUP_ID>**、**<ARTIFACT_ID>** および **<VERSION>** は、プロジェクトが使用する KJAR アーティファクトのグループ、アーティファクト、バージョン (GAV) に置き換えます。これらの値は、KJAR ソースディレクトリーにある **pom.xml** ファイルにあります。



注記

アーティファクトのバージョンを複数追加できます。


```

<build>
  <plugins>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>${version.org.kie}</version>
      <executions>
        <execution>
          <id>copy</id>
          <phase>prepare-package</phase>
          <goals>
            <goal>package-dependencies-kjar</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <artifactItems>
          <artifactItem>
            <groupId><GROUP_ID></groupId>
            <artifactId><ARTIFACT_ID></artifactId>
            <version><VERSION></version>
          </artifactItem>
        </artifactItems>
      </configuration>
    </plugin>
  </plugins>
</build>

```

KJAR の実行に必要なアーティファクトは、ビルド時に解決されます。

以下の例では、**Evaluation** アーティファクトのバージョンを 2 つ追加します。

```

<build>
  <plugins>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>${version.org.kie}</version>
      <executions>
        <execution>
          <id>copy</id>
          <phase>prepare-package</phase>
          <goals>
            <goal>package-dependencies-kjar</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <artifactItems>
          <artifactItem>
            <groupId>com.myspace</groupId>
            <artifactId>Evaluation</artifactId>
            <version>1.0.0-SNAPSHOT</version>
          </artifactItem>
          <artifactItem>
            <groupId>com.myspace</groupId>

```

```
<artifactId>Evaluation</artifactId>
<version>2.0.0-SNAPSHOT</version>
</artifactItem>
</artifactItems>
</configuration>
</plugin>
</plugins>
</build>
```

6. 自己完結型 Spring Boot イメージをビルドするには、**<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service** ディレクトリーで以下のコマンドを入力します。

```
mvn install
```

7. オプション: 自己完結型 Spring Boot イメージを実行するには、**target** サブディレクトリーで JAR ファイルを見つけ、以下のコマンドを入力します。

```
java -jar <FILENAME>.jar
```

このコマンドで **<FILENAME>** は JAR ファイルの名前に置き換えます。

第7章 ビジネスアプリケーションの実行

デフォルトでは、ビジネスアプリケーションには、実行プロジェクト、つまりサービスプロジェクトが1つ含まれています。サービスプロジェクトは、Windows または Linux のスタンドアロン (管理対象外) または開発 (管理対象) モードで実行できます。スタンドアロンモードを使用すると、追加の要件なしでアプリケーションを起動できます。開発モードで開始されたアプリケーションは、Business Central を Decision Manager コントローラーとして使用する必要があります。

7.1. スタンドアロンモードでのビジネスアプリケーションの実行

スタンドアロン (管理対象外) モードでは、追加要件なしにビジネスアプリケーションを起動できます。

前提条件

- [business applications](#) の Web サイトを使用して作成した **<BUSINESS-APPLICATION>.zip** ファイルがある。ここで、**<BUSINESS-APPLICATION>** は Spring Boot プロジェクトの名前に置き換えます。
- ビジネスアプリケーションが設定されている。

手順

1. **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service** ディレクトリーに移動します。
2. 以下のコマンドの1つを実行します。

表7.1 スタンドアロンの起動オプション

コマンド	説明
./launch.sh clean install	(Linux または UNIX) スタンドアロンモードで起動します。
./launch.bat clean install	(Windows) スタンドアロンモードで起動します。
./launch.sh clean install -Pmysql	(Linux または UNIX) MySQL データベースでアプリケーションを設定した場合に、スタンドアロンモードで起動します。
./launch.bat clean install -Pmysql	(Windows) MySQL データベースでアプリケーションを設定した場合に、スタンドアロンモードで起動します。
./launch.sh clean install -Ppostgres	(Linux または UNIX) PostgreSQL データベースでアプリケーションを設定した場合に、スタンドアロンモードで起動します。
./launch.bat clean install -Ppostgres	(Windows) PostgreSQL データベースでアプリケーションを設定した場合に、スタンドアロンモードで起動します。

clean install の引数で、Maven に、新規インストールをするように指示を出します。プロジェクトは、以下の順番に構築されます。

- データモデル
- ビジネスアセット
- サービス
スクリプトの初回実行時には、プロジェクトの依存関係がすべてダウンロードされるため、プロジェクトのビルドに時間がかかる場合があります。ビルドの最後に、アプリケーションが起動します。

3. 以下のコマンドを入力して、ビジネスアプリケーションにアクセスします。

```
http://localhost:8090/
```

4. 認証情報 **user/user** または **kieserver/kieserver1!** を入力します。

7.2. 開発モードでのビジネスアプリケーションの実行

開発 (管理対象) モードでは、開発者は Red Hat Decision Manager ビジネスアプリケーションアセットのプロジェクトで作業し、再起動の必要なしに動的に変更をデプロイできます。さらに、開発モードでは、プロセスインスタンス、タスク、ジョブなど、ビジネス自動化機能が完全に監視されている環境が提供されます。

前提条件

- [business applications](#) の Web サイトを使用して作成した Spring Boot アプリケーションの ZIP ファイルがある。
- ビジネスアプリケーションを設定している。
- Business Central をインストールし、実行している。

手順

1. **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service** ディレクトリーに移動します。**<BUSINESS-APPLICATION>** は Spring Boot プロジェクトの名前に置き換えます。
2. 以下のコマンドの1つを実行します。

表7.2 管理起動オプション

コマンド	説明
./launch-dev.sh clean install	(Linux または UNIX) 開発モードで起動します。
./launch-dev.bat clean install	(Windows) 開発モードで起動します。
./launch-dev.sh clean install -Pmysql	(Linux または UNIX) MySQL データベースでアプリケーションを設定した場合に、開発モードで起動します。

<code>./launch-dev.bat clean install -Pmysql</code>	(Windows) MySQL データベースでアプリケーションを設定した場合に、開発モードで起動します。
<code>./launch-dev.sh clean install -Ppostgres</code>	(Linux または UNIX) PostgreSQL データベースでアプリケーションを設定した場合に、開発モードで起動します。
<code>./launch-dev.bat clean install -Ppostgres</code>	(Windows) PostgreSQL データベースでアプリケーションを設定した場合に、開発モードで起動します。

clean install の引数で、Maven に、新規インストールをするように指示を出します。プロジェクトは、以下の順番に構築されます。

- データモデル
- ビジネスアセット
- サービス
スクリプトの初回実行時には、プロジェクトの依存関係がすべてダウンロードされるため、プロジェクトのビルドに時間がかかる場合があります。ビルドの最後に、アプリケーションが起動します。

3. 以下のコマンドを入力して、ビジネスアプリケーションにアクセスします。

`http://localhost:8090/`

4. 認証情報 **user/user** または **kieserver/kieserver1!** を入力します。ビジネスアプリケーションを起動した後に、Decision Manager コントローラーと接続すると、Business Central の **Menu** → **Deploy** → **Execution Servers** に表示されます。

第8章 RED HAT OPENSIFT CONTAINER PLATFORM での SPRINGBOOT ビジネスアプリケーションの実行

Red Hat OpenShift Container Platform で Red Hat Decision Manager SpringBoot ビジネスアプリケーションを実行するには、イミュータブルイメージを作成し、このイメージを Red Hat OpenShift Container Platform 環境にプッシュします。

前提条件

- Red Hat Decision Manager Spring Boot ビジネスアプリケーションを開発済みである。アプリケーションの作成方法は、[2章 ビジネスアプリケーションの作成](#)を参照してください。
- 必要に応じて、アプリケーションに Spring セキュリティーを設定している。Spring セキュリティーの設定方法は、[4章 Red Hat Decision Manager での Spring Security](#) を参照してください。
- ビジネスアプリケーションに必要な追加の Spring 設定を完了している。ビジネスアプリケーションの Spring 設定に関する方法については、[5章 Red Hat Decision Manager Spring Boot の設定](#)を参照してください。
- ビジネスアプリケーションに、JAR ファイルを1つ作成している。SpringBoot ビジネスアプリケーションに単一の JAR ファイルを作成する方法は、[6章 自己完結型 Red Hat Decision Manager Spring Boot JAR ファイルの作成](#)を参照してください。
- oc** コマンドを使用して Red Hat OpenShift Container Platform 環境にログインしており、必要なプロジェクトがアクティブである。

手順

- ビジネスアプリケーションプロジェクトディレクトリー外部で、以下のサブディレクトリーを含めて **ocp-image** ディレクトリーを作成します。

```
ocp-image
|--/root
|--/opt
|-- /spring-service
```

- ビジネスアプリケーションの単一の JAR ファイルを **root/opt/spring-service** サブディレクトリーにコピーします。以下に例を示します。

```
cd ../business-application-service
cp target/business-application-service-1.0-SNAPSHOT.jar ../ocp-image/root/opt/spring-service/
```

- ocp-image** ディレクトリーで、以下の内容を含む **Dockerfile** ファイルを作成します。

```
FROM registry.access.redhat.com/ubi8/openjdk-11:latest
COPY root /
EXPOSE 8090
WORKDIR /opt/spring-service/
CMD ["sh", "-c", "java ${JAVA_OPTIONS} -Dorg.kie.server.mode=PRODUCTION -jar /opt/spring-service/<FILENAME>.jar"]
```

<FILENAME>.jar は、ビジネスアプリケーションの単一の JAR ファイルの名前に置き換えます。

4. 初期イメージをビルドし、Red Hat OpenShift Container Platform 環境にデプロイするには、以下の手順を実行します。

- a. イメージをビルドするには、**ocp-image** ディレクトリーで以下のコマンドを実行します。

```
oc new-build --binary --strategy=docker --name openshift-kie-springboot
oc start-build openshift-kie-springboot --from-dir=. --follow
```

オプション L: **openshift-kie-springboot** は、これらのコマンドと後続の全コマンドのカスタムアプリケーションの名前に置き換えます。

- b. Red Hat OpenShift Container Platform 環境でイメージをデプロイするには、以下のコマンドを実行します。

```
oc new-app openshift-kie-springboot
```

- c. オプション: イメージのルートを公開するには、以下のコマンドを実行します。

```
oc expose service/openshift-kie-springboot --port=8090
```

5. 新しいバージョンの Red Hat Decision Manager または Spring Boot の JAR ファイルをビルドした場合など、すでにイメージをビルドし、更新する必要がある場合には、**ocp-image** ディレクトリーで以下のコマンドを実行します。

```
oc start-build openshift-kie-springboot --from-dir=. --follow
```

第9章 BUSINESS CENTRAL へのビジネスアセットプロジェクトのインポートおよびデプロイ

Red Hat Decision Manager ビジネスアプリケーションの一部であるビジネスアセットプロジェクトを Business Central にインポートしてから、そのプロジェクトをビジネスアプリケーションにデプロイできます。

前提条件

- 開発モードで実行されているビジネスアプリケーションプロジェクトがある。
- Red Hat Decision Manager の Business Central がインストールされている。

手順

1. **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-kjar** ディレクトリーに移動します。**<BUSINESS-APPLICATION>** は Spring Boot プロジェクトの名前に置き換えます。
2. 以下のコマンドを実行して、プロジェクトの Git リポジトリを初期化します。

```
$ git init
$ git add -A
$ git commit -m "Initial project structure"
```

3. Business Central にログインし、**Menu → Design → Projects** の順に移動します。
4. **Import Project** を選択して、以下の URL を入力します。

```
file:///<business-application-path>/<business-application-name>-kjar
```

5. **Import** をクリックして、インポートするプロジェクトを確定します。
6. ビジネスアセットプロジェクトを Business Central にインポートしてから、**Add Assets** をクリックして、ルールやデシジョンテーブルなどのアセットを使用しているビジネスアセットプロジェクトに追加します。
7. プロジェクトページで **Deploy** をクリックして、実行中のビジネスアプリケーションにプロジェクトをデプロイします。



注記

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

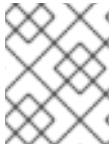
KIE Server の環境モードを設定するには、**org.kie.server.mode** システムプロパティを **org.kie.server.mode=development** または **org.kie.server.mode=production** に設定します。Business Central の対応するプロジェクトでのデプロイメント動作を設定するには、プロジェクトの **Settings → General Settings → Version** に移動し、**Development Mode** オプションを選択します。デフォルトでは、KIE Server および Business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

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

第10章 JMS メッセージブローカーで監査データの複製

KIE Server 監査データを Java Message Service (JMS) メッセージブローカー (ActiveMQ、Artemis など) に複製してから、外部データベーススキーマにデータをダンプし、アプリケーションスキーマから監査データを削除して Spring Boot アプリケーションのパフォーマンスを向上することができます。

メッセージブローカーのデータを複製するようにアプリケーションを設定すると、KIE Server でイベントが発生したときにそのイベントの記録は KIE Server データベーススキーマに保存され、メッセージブローカーに送信されます。その後、外部サービスを設定して、メッセージブローカーデータをアプリケーションのデータベーススキーマの正確なレプリカに使用できます。このデータは、イベントが KIE Server によって生成されるたびに、メッセージブローカーおよび外部データベースに追加されます。



注記

監査データのみがメッセージブローカーに保存されます。他のデータはレプリケートされません。

前提条件

- 既存の Red Hat Decision Manager Spring Boot プロジェクトがある。

手順

1. テキストエディターで Spring Boot アプリケーションの **pom.xml** ファイルを開きます。
2. KIE Server Spring Boot 監査依存関係を **pom.xml** ファイルに追加します。

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-server-spring-boot-autoconfiguration-audit-replication</artifactId>
  <version>${version.org.kie}</version>
</dependency>
```

3. JMS クライアントの依存関係を追加します。以下の例では、Advanced Message Queuing Protocol (AMQP) 依存関係を追加します。

```
<dependency>
  <groupId>org.amqphub.spring</groupId>
  <artifactId>amqp-10-jms-spring-boot-starter</artifactId>
  <version>2.2.6</version>
</dependency>
```

4. JMS プールの依存関係を追加します。

```
<dependency>
  <groupId>org.messaginghub</groupId>
  <artifactId>pooled-jms</artifactId>
</dependency>
```

5. KIE Server 監査レプリケーションがキューを使用するように設定するには、以下のタスクを完了します。
 - a. 以下の行を、Spring Boot アプリケーションの **application.properties** ファイルに追加します。

```
kieserver.audit-replication.producer=true
kieserver.audit-replication.queue=audit-queue
```

- b. メッセージブローカークライアントに必要なプロパティを追加します。以下の例は、AMQP に KIE Server を設定する方法を示しています。ここで、**<JMS_HOST_PORT>** はブローカーがリッスンするポートで、**<USERNAME>** および **<PASSWORD>** はブローカーのログイン認証情報になります。

```
amqphub.amqp10jms.remote-url=amqp://<JMS_HOST_PORT>
amqphub.amqp10jms.username=<USERNAME>
amqphub.amqp10jms.password=<PASSWORD>
amqphub.amqp10jms.pool.enabled=true
```

- c. 以下の行を、メッセージブローカーデータを使用するサービスの **application.properties** ファイルに追加します。

```
kieserver.audit-replication.consumer=true
kieserver.audit-replication.queue=audit-queue
```

- d. メッセージブローカークライアントに必要なプロパティを、メッセージブローカーデータを使用するサービスの **application.properties** ファイルに追加します。以下の例は、AMQP に KIE Server を設定する方法を示しています。ここで、**<JMS_HOST_PORT>** はメッセージブローカーがリッスンするポートで、**<USERNAME>** および **<PASSWORD>** はメッセージブローカーのログイン認証情報になります。

```
amqphub.amqp10jms.remote-url=amqp://<JMS_HOST_PORT>
amqphub.amqp10jms.username=<USERNAME>
amqphub.amqp10jms.password=<PASSWORD>
amqphub.amqp10jms.pool.enabled=true
```

6. KIE Server 監査レプリケーションがトピックを使用するように設定するには、以下のタスクを実行します。

- a. 以下の行を、Spring Boot アプリケーションの **application.properties** ファイルに追加します。

```
kieserver.audit-replication.producer=true
kieserver.audit-replication.topic=audit-topic
```

- b. メッセージブローカークライアントに必要なプロパティを、メッセージブローカーデータを使用するサービスの **application.properties** ファイルに追加します。以下の例は、AMQP に KIE Server を設定する方法を示しています。ここで、**<JMS_HOST_PORT>** はメッセージブローカーがリッスンするポートで、**<USERNAME>** および **<PASSWORD>** はメッセージブローカーのログイン認証情報になります。

```
spring.jms.pub-sub-domain=true
amqphub.amqp10jms.remote-url=amqp://<JMS_HOST_PORT>
amqphub.amqp10jms.username=<USERNAME>
amqphub.amqp10jms.password=<PASSWORD>
amqphub.amqp10jms.pool.enabled=true
```

- c. 以下の行を、メッセージブローカーデータを使用するサービスの **application.properties** ファイルに追加します。

■

```
kieserver.audit-replication.consumer=true
kieserver.audit-replication.topic=audit-topic::jbpm
kieserver.audit-replication.topic.subscriber=jbpm
spring.jms.pub-sub-domain=true
```

- d. メッセージブローカークライアントに必要なプロパティを、メッセージブローカーデータを使用するサービスの **application.properties** ファイルに追加します。以下の例は、AMQP に KIE Server を設定する方法を示しています。ここで、**<JMS_HOST_PORT>** はメッセージブローカーがリスンするポートで、**<USERNAME>** および **<PASSWORD>** はメッセージブローカーのログイン認証情報になります。

```
amqphub.amqp10jms.remote-url=amqp://<JMS_HOST_PORT>
amqphub.amqp10jms.username=<USERNAME>
amqphub.amqp10jms.password=<PASSWORD>
amqphub.amqp10jms.pool.enabled=true
amqphub.amqp10jms.clientId=jbpm
```

7. オプション: レプリケートされたデータが含まれる KIE Server を読み取り専用に設定するには、**application.properties** ファイルの **org.kie.server.rest.mode.readonly** プロパティを **true** に設定します。

```
org.kie.server.rest.mode.readonly=true
```

関連情報

- 「[Spring Boot JMS 監査レプリケーションパラメーター](#)」

10.1. SPRING BOOT JMS 監査レプリケーションパラメーター

以下の表は、Spring Boot で Red Hat Decision Manager アプリケーションの JMS 監査レプリケーションを設定するために使用するパラメーターについて説明しています。

表10.1 Spring Boot JMS 監査レプリケーションパラメーター

パラメーター	値	説明
kieserver.audit-replication.producer	true, false	ビジネスアプリケーションが、JMS メッセージを複製してキューまたはトピックのいずれかに送信するプロデューサーとして動作するかどうかを指定します。
kieserver.audit-replication.consumer	true, false	ビジネスアプリケーションが、キューまたはトピックのいずれかから JMS メッセージを受信するコンシューマーとして動作するかどうかを指定します。
kieserver.audit-replication.queue	string	メッセージを送信または消費する JMS キューの名前。
kieserver.audit-replication.topic	string	メッセージを送信または消費する JMS トピックの名前。

パラメーター	値	説明
kieserver.audit-replication.topic.subscriber	string	トピックサブスクライバーの名前。
org.kie.server.rest.mode.readonly	true, false	ビジネスアプリケーションの読み取り専用モードを指定します。

パート II. RED HAT DECISION MANAGER と RED HAT FUSE の 統合

システム管理者は、Red Hat JBoss Enterprise Application Platform で、Red Hat Decision Manager と Red Hat Fuse を統合して、統合サービス間の通信を容易化します。

第11章 RED HAT FUSE および RED HAT DECISION MANAGER

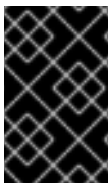
Red Hat Fuse は、アジャイル統合ソリューションの一部である、分散型のクラウドネイティブ統合プラットフォームです。チームはこの分散アプローチを使用することで、必要に応じて統合サービスをデプロイできます。Fuse には、統合エキスパート、アプリケーション開発者、ビジネスユーザーなど、さまざまなユーザーに柔軟にサービスを提供でき、各自がデプロイメント、アーキテクチャー、ツールを選択できます。API 中心のコンテナベースのアーキテクチャーは、各サービスを切り離して、個別に作成、展開、デプロイできるようにします。その結果、企業全体のコラボレーションをサポートする統合ソリューションが実現できます。

Red Hat Decision Manager は、ビジネスルール管理、複合イベント処理、Decision Model & Notation (DMN) 実行、およびプランニングの問題を解決するための Red Hat ビルドの OptaPlanner を組み合わせた、オープンソースの意思決定管理プラットフォームです。これにより、ビジネス上の意思決定を自動化し、そのロジックをビジネス全体で利用できるようにします。

ルール、デシジョンテーブル、DMN モデルなどのビジネスアセットはプロジェクト内で組織化され、Business Central リポジトリに保存されます。これにより、ビジネス全体で一貫性や透明性を維持し、監査を行えます。ビジネスユーザーは、IT 担当者からのサポートなしでビジネスロジックを編集できます。

Apache Karaf コンテナプラットフォームに Red Hat Fuse をインストールしてから、このコンテナに Red Hat Decision Manager をインストールし、設定してください。

Red Hat JBoss Enterprise Application Platform の別のインスタンスに Red Hat Fuse をインストールし、Red Hat Decision Manager と統合することもできます。**kie-camel** モジュールは、Red Hat Fuse と Red Hat Decision Manager 間の統合を提供します。



重要

Red Hat Decision Manager 7.12 がサポートする Red Hat Fuse のバージョンについては、[Red Hat Process Automation Manager 7 でサポートされる設定](#) を参照してください。



注記

Red Hat Fuse on Spring Boot をインストールできます。Red Hat Decision Manager では、このシナリオ向けの特別な統合はありません。

Red Hat Fuse on Spring Boot で実行しているアプリケーションの **kie-server-client** ライブラリーを使用して、KIE Server で実行している Red Hat Decision Manager サービスと通信できるようにします。

kie-server-client ライブラリーの使用に関する詳細は、[KIE API を使った Red Hat Decision Manager の操作](#) を参照してください。

第12章 APACHE KARAF 上に FUSE を統合した RED HAT DECISION MANAGER のデシジョンエンジン

Apache Karaf は、スタンドアロンで、オープンソースのランタイム環境です。OSGi Alliance の OSGi 標準に基づいています。Karaf は、高度なクラ出力デバッグサポートを備えた OSGi バンドルを通じてモジュール化のサポートを提供します。Karaf コンテナでは、依存関係の複数のバージョンを並行してデプロイできます。ホットコードスワップを使用すると、コンテナをシャットダウンせずにモジュールをアップグレードまたは置き換えることができます。

Red Hat Decision Manager は、Karaf 機能を使用して、Karaf で FUSE と統合されます。これらの機能を使用して、Karaf 上の FUSE 向けに Red Hat Decision Manager の個別のコンポーネントをインストールできます。

機能ファイルは XML ファイルで、このファイルを使用して特定の機能向けにどの DOSGI バンドルをインストールするかを指定します。以下の機能 XML ファイルにより、Red Hat Decision Manager と Fuse on Karaf の統合を容易にします。

- **rhba-features-<FUSE-VERSION>-features.xml**
このファイルは、Karaf にインストールした Fuse に含まれており、**<FUSE-VERSION>** は Fuse のバージョンに置き換えます。このファイルは、**system/org/jboss/fuse/features/rhba-features** ディレクトリーの Karaf システムリポジトリに保存されます。このファイルには、Red Hat Decision Manager 機能をインストールするための前提条件が含まれています。
- **kie-karaf-features-7.59.0.Final-redhat-00006-features-fuse.xml**
このファイルは Red Hat Decision Manager に含まれており、Red Hat Decision Manager 機能を提供します。この機能により、Red Hat Fuse にデプロイできる OSGi 機能が決まります。OSGi ユーザーは、このファイルから機能をインストールして、Red Hat Decision Manager を Fuse にインストールし、それぞれのアプリケーションで使用できます。この機能ファイルは、Red Hat Process Automation Manager で配布されるオンラインおよびオフラインの Maven リポジトリに配置されています。このファイルのグループ ID、アーティファクト ID、およびバージョン (GAV) 識別子は、**org.kie:kie-karaf-features:7.59.0.Final-redhat-00006** です。

12.1. KARAF 上の古くなった RED HAT DECISION MANAGER 機能 XML ファイルの削除

お使いの環境に、以前の Red Hat Decision Manager 機能 XML ファイル (例: **kie-karaf-features-<VERSION>-features.xml**) が含まれている場合には、このファイルと関連するファイルすべてを削除してから、最新の機能 XML ファイルをインストールする必要があります。

前提条件

- 以前の機能 XML ファイルが Apache Karaf の環境に存在する。

手順

1. 以下のコマンドを実行して、使用環境に以前の Red Hat Decision Manager 機能 XML ファイルが含まれているかどうかを確認します。

```
$ JBossFuse:karaf@root> feature:repo-list
$ JBossFuse:karaf@root> feature:list
```

2. 以下のコマンドを入力します。**<FUSE_HOME>** は Fuse のインストールディレクトリーに置き換えて、Red Hat Fuse コンソールを起動します。

■


```
$ ./<FUSE_HOME>/bin/fuse
```

- 以下のコマンドを入力します。**<FEATURE_NAME>** は、アンインストールする機能の名前に置き換えて、以前の機能 XML ファイルを使用するアプリケーションまたは機能をアンインストールします。

```
JBossFuse:karaf@root> features:uninstall <FEATURE_NAME>
```

以下の例では、機能の削除方法を紹介します。

```
JBossFuse:karaf@root> features:uninstall drools-module
JBossFuse:karaf@root> features:uninstall jbpm
JBossFuse:karaf@root> features:uninstall kie-ci
```

- Karaf の home で、**drools**、**kie**、または **jbpm** を使用するバンドルへの参照を検索します。以下の例では、**grep** を使用してこれらのコンポーネントを検索する方法を示しています。

```
karaf@root> list -t 0 -s | grep drools
karaf@root> list -t 0 -s | grep kie
karaf@root> list -t 0 -s | grep jbpm
```

この例は、上記のコマンドからの出力です。

250	Active	80	7.19.0.201902201522	org.drools.canonical-model
251	Active	80	7.19.0.201902201522	org.drools.cdi
252	Active	80	7.19.0.201902201522	org.drools.compiler

- 以下のコマンドを入力します。**BUNDLE_ID** は、検索で返されたバンドル ID に置き換えて、以前の手順で検出されたバンドルを削除します。

```
karaf@root> osgi:uninstall BUNDLE_ID
```

- 次のコマンドを入力して、古くなった **drools-karaf-features** の URL を削除します。

```
karaf@root> features:removeurl
mvn:org.kie/kie-karaf-features/VERSION.Final-redhat-VERSION/xml/features
```

- Fuse を再起動します。

12.2. XML ファイルを使用した KARAF への RED HAT DECISION MANAGER 機能のインストール

Karaf に Red Hat Decision Manager 機能をインストールして、Red Hat Decision Manager プロセス向けに動的ランタイム環境を作成できます。

前提条件

- Apache Karaf コンテナで Red Hat Fuse が利用できるようになっている。Apache Karaf への Fuse のインストール手順については、[Installing Red Hat Fuse on the Apache Karaf container](#) を参照してください。

- 「[Karaf 上の古くなった Red Hat Decision Manager 機能 XML ファイルの削除](#)」に記載されているように、以前の Red Hat Decision Manager 機能 XML ファイルが削除されている。

手順

Red Hat Decision Manager の機能をインストールするには、次のコマンドを入力します。

```
$ JBossFuse:karaf@root> feature:install <FEATURE_NAME>
```

注記

org.drools.osgi.spring.OsgiKModuleBeanFactoryPostProcessor の代わりに **org.kie.spring.KModuleBeanFactoryPostProcessor** を使用して、OSGi 環境の KIE 要素を後処理します。

kie-spring 機能をインストールする前に **drools-module** 機能をインストールしないようにしてください。先にインストールしてしまうと、**drools-compiler** バンドルにより、**kie-spring** がエクスポートしたパッケージが検出されなくなります。

これらの機能を間違った順番でインストールした場合は、**osgi:refresh drools-compiler_bundle_ID** を実行して、**drools-compiler** が強制的に **Import-Package** メタデータをリビルドするようにします。

このコマンドでは、**<FEATURE_NAME>** は、「[Red Hat Decision Manager の Karaf 機能](#)」に記載の機能の1つに置き換えます。

12.3. MAVEN を使用した KARAF への RED HAT DECISION MANAGER 機能のインストール

必要に応じて、Apache Karaf 上にある Fuse で Red Hat Decision Manager をインストールして、統合サービスをデプロイします。

前提条件

- Apache Karaf インストールに Red Hat Fuse 7.9 が存在する。インストール手順については、[Installing Red Hat Fuse on the Apache Karaf container](#)を参照してください。
- 「[Karaf 上の古くなった Red Hat Decision Manager 機能 XML ファイルの削除](#)」の説明のように、以前の機能 XML ファイルが削除されている。

手順

1. Maven リポジトリを設定するには、テキストエディターで **FUSE_HOME/etc/org.ops4j.pax.url.mvn.cfg** ファイルを開きます。
2. **https://maven.repository.redhat.com/ga/** リポジトリが **org.ops4j.pax.url.mvn.repositories** 変数に存在することを確認します。必要に応じて、追加してください。

注記

org.ops4j.pax.url.mvn.repositories 変数内のエントリーをコンマ、スペース、およびバックスラッシュ (,) で区切ります。バックスラッシュを追加すると、強制的に改行されます。

3. Fuse を起動するには以下のコマンドを入力します。<FUSE_HOME> は Fuse のインストールディレクトリーに置き換えます。

```
$ ./FUSE_HOME/bin/fuse
```

4. インストールの前提条件が含まれる機能ファイルに参照を追加するには、以下のコマンドを入力します。<FUSE_VERSION> は、インストールする Fuse のバージョンに置き換えます。

```
$ feature:repo-add mvn:org.jboss.fuse.features/rhba-features/<FUSE-VERSION>/xml/features
```

5. 以下のコマンドを入力して、Red Hat Decision Manager 機能の XML ファイルへの参照を追加します。

```
$ JBossFuse:karaf@root> features:addurl mvn:org.kie/kie-karaf-features/VERSION/xml/features-fuse
```

現在の **drools-karaf-features** バージョンを確認するには、[Red Hat Decision Manager 7 Supported Configurations](#) ページを参照してください。

6. 以下のコマンドを入力して、Red Hat Decision Manager 機能の XML ファイルで提供される機能をインストールします。このコマンドでは、<FEATURE_NAME> は、「[Red Hat Decision Manager の Karaf 機能](#)」に記載の機能の1つに置き換えます。

```
JBossFuse:karaf@root> features:install <FEATURE_NAME>
```

7. 次のコマンドを入力して、インストールを確認します。

```
$ JBossFuse:karaf@root> feature:list
```

機能が正常にインストールされると、ステータスは **started** になります。

12.4. RED HAT DECISION MANAGER の KARAF 機能

以下の表では、Red Hat Decision Manager の Karaf 機能を紹介します。

機能	説明
drools-module	Drools のコアとコンパイラーが含まれており、プレーン DRL から KIE ベースと KIE セッションを作成するのに使用します。また、実行可能モデルの実装も含まれています。永続性、プロセス、またはデシジョンテーブルを必要とせずに Drools を使用してルール評価ができます。
drools-template	Drools テンプレートが含まれています。

機能	説明
drools-jpa	プロセスまたはデシジョンテーブルを必要とせず に、Drools を使用して永続性 (Persistence) とトラン ザクション (Transaction) でルール評価を行いま す。 drools-jpa 機能には drools-module が含まれ ます。 droolsjbpm-hibernate 機能をインストール するか、互換性のあるハイバネートバンドルがイン ストールされていることを確認しないといけない場 合があります。
drools-decisiontable	デシジョンテーブルと合わせて、Drools を使用しま す。
コアエンジン JAR および kie-ci	Red Hat Decision Manager と KIE スキャナー (kie- ci) を使用して、Maven リポジトリから kJAR をダ ウンロードします。
kie-camel	Fuse と Red Hat Decision Manager を統合する Apache Camel エンドポイントである kie-camel コ ンポーネントを提供します。
kie-spring	kie-spring コンポーネントをインストールし、XML タグを使用して KIE セッションのリスナーを設定可 能にします。

第13章 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM への FUSE のインストール

Red Hat JBoss EAP 7.4 に Red Hat Fuse 7.9 をインストールして、Red Hat Decision Manager と統合します。

前提条件

- Red Hat JBoss Enterprise Application Platform 7.4 に Red Hat Decision Manager がインストールされている。インストールの説明は、[Red Hat JBoss EAP 7.4 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。
- Red Hat JBoss Enterprise Application Platform 7.4 の別のインスタンスを利用できる。

手順

- Red Hat JBoss Enterprise Application Platform 7.4 に Red Hat Fuse 7.9 をインストールします。インストールの説明は、Red Hat Fuse ドキュメントの [JBoss EAP のインストール](#) セクション参照してください。
- テキストエディターで、Fuse のホームディレクトリーにある **pom.xml** を開きます。
- 以下の例のように **pom.xml** ファイルを編集して、**kie-camel** コンポーネントの依存関係を含めて、統合プロジェクトを作成します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-core</artifactId>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
</dependency>
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <exclusions>
    <exclusion>
      <groupId>aopalliance</groupId>
      <artifactId>aopalliance</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-api</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.jboss.spec.javax.xml.bind</groupId>
      <artifactId>jboss-jaxb-api_2.3_spec</artifactId>
    </exclusion>
    <exclusion>
      <groupId>javax.activation</groupId>
      <artifactId>activation</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

```
</exclusion>
</exclusions>
</dependency>
<dependency>
  <groupId>org.jbpm</groupId>
  <artifactId>jbpm-bpmn2</artifactId>
</dependency>
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-camel</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-cxf</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-cxf-transport</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.thoughtworks.xstream</groupId>
      <artifactId>xstream</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.jboss.spec.javax.ws.rs</groupId>
      <artifactId>jboss-jaxrs-api_2.0_spec</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

第14章 KIE-CAMEL コンポーネント

kie-camel コンポーネントは、Fuse と Red Hat Decision Manager が統合された Red Hat Fuse が提供する Apache Camel エンドポイントです。このコンポーネントを使用して、ルートにするプルして実行できる Maven グループ ID、アーティファクト ID、バージョン (GAV) の識別子を使用して、Red Hat Decision Manager モジュールを指定できます。また、ファクトとしてメッセージボディーの一部を指定することもできます。埋め込みエンジンまたは KIE Server で、**kie-camel** コンポーネントを使用できます。

埋め込みエンジン

このシナリオでは、KIE エンジンが Fuse 統合プロジェクトと同じコンテナで実行されます。エンジン間の通信には、KIE コマンドを使用できます。Camel プロデューサーを作成するには、以下の URI を使用します。

```
kie-local:kie-session-name?action=execute
```

たとえば、次のコマンドを入力して、Spring の Camel ルートを初期化します。

```
<from uri="direct:runCommand" />
<to uri="kie-local:kie-session1?action=execute"/>
```

KIE Server

このシナリオでは、**kie-camel** コンポーネントは KIE Server REST API を使用して KIE Server に接続します。こうすることでユーザーは、KIE Server API を使用して KIE Server と通信できます。プロデューサーを作成するには、以下の URI を使用します。

```
kie:http://username:password@kie-server-url`
```

たとえば、次のコマンドを入力して、Spring の Camel ルートを初期化します。

```
<from uri="direct:runCommand" />
<to uri="kie:http://user:psswd@localhost:8080/kie-server-services/services/rest/server"/>
```

メッセージには以下のヘッダーが含まれます。

表14.1 メッセージのヘッダーと説明

ヘッダー	説明
CamelKieClient	KIE Server クライアント (必須)
CamelKieOperation	KIE Server クライアント (必須)
CamelKieParameterName	クライアントメソッドパラメーターの値 (任意)
CamelKieBodyParam	メッセージボディーを保存するメソッドパラメーター (任意)

パート III. RED HAT DECISION MANAGER と RED HAT SINGLE SIGN-ON の統合

システム管理者は、Red Hat シングルサインオンを Red Hat Decision Manager に統合し、単一の認証メソッドを使用することで Red Hat Decision Manager ブラウザーアプリケーションを保護できます。

前提条件

- Red Hat JBoss EAP 7.4 に Red Hat Decision Manager がインストールされている。詳細は、[Red Hat JBoss EAP 7.4 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15章 統合オプション

Red Hat シングルサインオン (RH-SSO) は、ブラウザーアプリケーションと REST Web サービス、および Git へのアクセスのセキュリティを確保するために使用できるシングルサインオンソリューションです。

Red Hat Decision Manager と RH-SSO を統合する際に、Red Hat Decision Manager 向けに SSO と IDM (アイデンティティ管理) を作成します。RH-SSO のセッション管理機能により、一度認証するだけで、Web 上でさまざまな Red Hat Decision Manager 環境を使用できます。

以下の章では、Red Hat Decision Manager と RH-SSO を統合する方法を説明します。

- **18章RH-SSO を使用した Business Central の認証**

RH-SSO サーバーを使用して Red Hat Decision Manager を認証するには、Red Hat Decision Manager Web クライアント (Business Central) とリモートサービスの両方を RH-SSO で保護する必要があります。この統合により、Business Central またはリモートサービスコンシューマーのいずれかから RH-SSO を介して Red Hat Decision Manager に接続できます。

- **19章RH-SSO を使用した KIE Server の認証**

RH-SSO サーバーを使用して KIE Server を認証するには、KIE Server が提供するリモートサービスのセキュリティを確保する必要があります。これを行うことで、リモートの Red Hat Decision Manager サービスコンシューマー (ユーザーまたはサービス) を有効にし、RH-SSO を経由して認証します。KIE Server には Web インターフェイスがありません。

- **20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**

Business Central または KIE Server が RH-SSO を使用している場合、サードパーティークライアントは RH-SSO を使用して自己認証する必要があります。認証後は、Business Central および KIE Server が提供するリモートサービスのエンドポイント (REST API、リモートファイルシステムサービスなど) を使用できます。

Red Hat Decision Manager との LDAP 統合を容易にするには、LDAP での RH-SSO を使用することを検討してください。詳細は、[Red Hat Single Sign-On Server Administration Guide](#)の LDAP and Active Directory セクションを参照してください。

第16章 RH-SSO のインストールおよび設定

レルムは、Web またはアプリケーションサーバーに定義するセキュリティポリシードメインです。セキュリティレルムは、異なるアプリケーションリソースのアクセスを制限するのに使用します。RH-SSO インスタンスが非公開か他の製品と共有されているにかかわらず、新規レルムを作成する必要があります。マスターレルムを、スーパー管理者がシステムのレルムを作成して管理する場所として維持できます。他の製品システムと共有している RH-SSO インスタンスと統合して、これらのアプリケーションでシングルサインオンを行うためには、これらのアプリケーションですべて同じレルムが使用される必要があります。RH-SSO レルムを作成するには、RH-SSO 7.5 をダウンロード、インストール、および設定します。

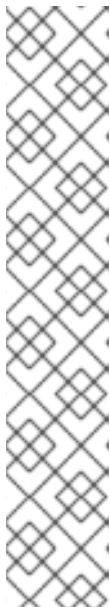


注記

Business Central および KIE Server が異なるサーバーにインストールされている場合は、両サーバーでこの手順を行ってください。

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。
 - **製品:** Red Hat Single Sign-On
 - **バージョン:** 7.5
2. **Red Hat Single Sign-On 7.5.0 Server(rh-sso-7.5.0.zip)** と最新のサーバーパッチをダウンロードします。
3. 基本的な RH-SSO スタンドアロンサーバーをインストールして設定するには、[Red Hat Single Sign On Getting Started Guide](#) の手順に従います。実稼働環境の高度な設定については、[Red Hat Single Sign On Server Administration Guide](#) を参照してください。



注記

同じシステムで RH-SSO と Red Hat Decision Manager サーバーの両方を実行する場合には、以下のアクションのいずれかによりポートの競合を避けてください。

- **RHSSO_HOME/standalone/configuration/standalone-full.xml** ファイルを更新して、ポートのオフセットを 100 に設定してください。以下に例を示します。

```
<socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:100}">
```

- 環境変数を使用して、サーバーの実行時にポートオフセットを設定します。

```
bin/standalone.sh -Djboss.socket.binding.port-offset=100
```

第17章 RED HAT DECISION MANAGER ロールおよびユーザー

Business Central または KIE Server にアクセスするには、サーバーを起動する前にユーザーを作成して適切なロールを割り当てます。Business Central または KIE Server のインストール時に、ユーザーとロールを作成できます。

Business Central と KIE Server の両方が単一のインスタンスで動作している場合、Business Central で認証されたユーザーは KIE Server にもアクセスできます。

ただし、Business Central と KIE Server が別のインスタンスで動作している場合、Business Central で認証されたユーザーが KIE Server にアクセスするには、別途認証が必要です。たとえば、Business Central で認証されているものの、KIE Server で認証されていないユーザーが Business Central でプロセ定義を表示または管理しようとすると、401 エラーがログファイルに記録され、**Invalid credentials to load data from remote server.Contact your system administrator.** メッセージが Business Central に表示されます。

本セクションでは、Red Hat Decision Manager のユーザーロールを説明します。



注記

admin、**analyst**、および **rest-all** のロールは Business Central 用に予約されています。**kie-server** ロールは KIE Server 用に予約されています。このため、Business Central または KIE Server のいずれか、またはそれら両方がインストールされているかどうかによって、利用可能なロールは異なります。

- **admin:** **admin** ロールを持つユーザーは Business Central 管理者です。管理者は、ユーザーの管理や、リポジトリの作成、クローン作成、および管理ができます。アプリケーションで必要な変更をすべて利用できます。**admin** ロールを持つユーザーは、Red Hat Decision Manager の全領域にアクセスできます。
- **analyst:** **analyst** ロールを持つユーザーには、すべてのハイレベル機能へのアクセスがあります。プロジェクトのモデル化が可能です。ただし、このユーザーは、**Design → Projects** ビューでスペースに貢献者を追加したり、スペースを削除したりできません。**analyst** ロールを持つユーザーは、管理者向けの **Deploy → Execution Servers** ビューにアクセスできません。ただし、これらのユーザーは、ライブラリーパースペクティブにアクセスするときに **Deploy** ボタンを使用できます。
- **rest-all:** **rest-all** ロールを持つユーザーは、Business Central REST 機能にアクセスできます。
- **kie-server:** **kie-server** ロールを持つユーザーは、KIE Server REST 機能にアクセスできます。

17.1. RED HAT DECISION MANAGER ユーザーの追加

Business Central または KIE Server の認証に RH-SSO を使用する前に、作成したレルムにユーザーを追加する必要があります。新しいユーザーを追加して、Red Hat Decision Manager にアクセスするためのロールを追加するには、以下の手順を行います。

1. RH-SSO 管理コンソールにログインして、ユーザーを追加するレルムを開きます。
2. **Manage** セクションで **Users** メニューアイテムをクリックします。
Users ページに空のユーザー一覧が表示されます。
3. 空のユーザー一覧で **Add User** ボタンをクリックして、新規ユーザーの作成を開始します。
Add User ページが開きます。

4. **Add User** ページで、ユーザー情報を入力して **Save** をクリックします。
5. **Credentials** タブをクリックして、パスワードを作成します。
6. Red Hat Decision Manager へのアクセスを許可するロールの新規ユーザーを割り当てます。たとえば、Business Central にアクセスするには **admin** ロールを割り当てるか、KIE Server にアクセスするには **kie-server** ロールを割り当てます。



注記

Business Central から OpenShift にデプロイするプロジェクトの場合は、ロールを割り当てずに **mavenuser** という RH-SSO ユーザーを作成し、OpenShift テンプレートの **BUSINESS_CENTRAL_MAVEN_USERNAME** および **BUSINESS_CENTRAL_MAVEN_PASSWORD** にこのユーザーを追加します。

7. **Roles** セクションの **Realm Roles** タブで、このロールをレルムロールとして定義します。
Business Central で使用するロールの場合は、ロールを **kie** クライアントのクライアントロールとして定義できます。**kie** クライアントの設定手順は、[「RH-SSO への Business Central クライアントの作成」](#) を参照してください。クライアントロールを使用するには、[「Business Central への RH-SSO クライアントアダプターのインストール」](#) の説明に従って、Business Central の追加設定も設定する必要があります。

レルムロールとして KIE Server で使用するロールを定義する必要があります。

8. **Users** ページの **Role Mappings** タブをクリックして、ロールを割り当てます。

第18章 RH-SSO を使用した BUSINESS CENTRAL の認証

本章では、RH-SSO を介して Business Central を認証する方法を説明します。この章には以下のセクションが含まれます。

- [「RH-SSO への Business Central クライアントの作成」](#)
- [「Business Central への RH-SSO クライアントアダプターのインストール」](#)
- [「RH-SSO による Business Central の外部ファイルシステムおよび Git リポジトリサービスへのアクセスを可能にする」](#)

前提条件

- [Red Hat JBoss EAP 7.4 への Red Hat Decision Manager のインストールおよび設定](#)の記載通りに、Business Central が Red Hat JBoss EAP 7.4 サーバーにインストールされている。
- [16章RH-SSO のインストールおよび設定](#) の記載通りに、RH-SSO がインストールされている。
- [「Red Hat Decision Manager ユーザーの追加」](#) の記載通りに、Business Central ユーザーが RH-SSO に追加されている。
- オプション: Business Central から RH-SSO ユーザーを管理する場合には RH-SSO の全 realm-management クライアントロールが Business Central の管理者ユーザーに追加されている。



注記

このセクションは、[「RH-SSO への Business Central クライアントの作成」](#) を除き、スタンドアロンのインストールが対象です。Red Hat OpenShift Container Platform で RH-SSO と Red Hat Decision Manager を統合する場合には、[「RH-SSO への Business Central クライアントの作成」](#) の手順のみを実行して、Red Hat OpenShift Container Platform に Red Hat Decision Manager 環境をデプロイしてください。Red Hat OpenShift Container Platform に Red Hat Decision Manager をデプロイする手順は、[Red Hat OpenShift Container Platform への Red Hat Decision Manager のデプロイメント](#) を参照してください。

18.1. RH-SSO への BUSINESS CENTRAL クライアントの作成

RH-SSO サーバーの起動後、RH-SSO 管理コンソールを使用して RH-SSO 向けに Business Central クライアントを作成します。

手順

1. Web ブラウザーに **<http://localhost:8180/auth/admin>** と入力して、RH-SSO 管理コンソールを開き、RH-SSO のインストール時に作成した管理者の認証情報を使用してログインします。



注記

Red Hat OpenShift Container Platform で RH-SSO を設定している場合は、RH-SSO ルートに公開されている URL を入力します。OpenShift 管理者は、必要に応じてこの URL を提供してください。

初回のログイン時に、新規ユーザー登録フォームで初期ユーザーを設定できます。

2. RH-SSO 管理コンソールで、**Realm Settings** メニューアイテムをクリックします。
3. **Realm Settings** ページで **Add Realm** をクリックします。
Add realm ページが表示されます。
4. **Add realm** ページで、レルムの名前を指定して **Create** をクリックします。
5. **Clients** メニューアイテムをクリックし、**Create** をクリックします。
Add Client ページが表示されます。
6. **Add Client** ページで、レルムにクライアントを新規作成するのに必要な情報を指定します。以下に例を示します。
 - **Client ID:** kie
 - **Client protocol:** openid-connect
 - **Root URL:** **http://localhost:8080/decision-central**



注記

Red Hat OpenShift Container Platform で RH-SSO を設定している場合は、KIE Server ルートに公開されている URL を入力します。OpenShift 管理者は、必要に応じてこの URL を提供してください。

7. **Save** をクリックして変更を保存します。
作成した新規クライアントの **Access Type** は、デフォルトでは **public** に設定されています。この設定を **confidential** に変更します。

これで、Business Central アプリケーションのクライアントが含まれるレルムに RH-SSO サーバーが設定され、**localhost:8180** で HTTP 接続をリスンした状態で実行しています。このレルムは、Business Central アプリケーションに異なるユーザー、ロール、セッションを提供します。



注記

RH-SSO サーバークライアントは単一の decision-central デプロイメントに対して URL を 1 つ使用します。配置設定が 2 つ以上ある場合、以下のようなエラーメッセージが表示されることがあります。

ご迷惑をおかけしております... 無効なパラメーター: **redirect_uri**

このエラーを解決するには、クライアント設定の **Valid Redirect URIs** フィールドに ***** を追加します。

Configure ページで、**Clients > kie > Settings** と進み、**Valid Redirect URIs** フィールドに、たとえば、***** を追加してください。

http://localhost:8080/business-central/*

18.2. BUSINESS CENTRAL への RH-SSO クライアントアダプターのインストール

RH-SSO をインストールしたら、Red Hat JBoss EAP に RH-SSO クライアントアダプターをインストールして、Business Central に対して設定する必要があります。

前提条件

- [Red Hat JBoss EAP 7.4 への Red Hat Decision Manager のインストールおよび設定](#)の記載通りに、Business Central が Red Hat JBoss EAP 7.4 インスタンスにインストールされている。
- [16章RH-SSO のインストールおよび設定](#) の記載通りに、RH-SSO がインストールされている。
- 「[Red Hat Decision Manager ユーザーの追加](#)」 の記載通りに、**admin** ロールが割り当てられたユーザーが RH-SSO に追加されている。

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。
 - **製品:** Red Hat Single Sign-On
 - **バージョン:** 7.5
2. **Patches** タブを選択します。
3. **Red Hat Single Sign-On 7.5 Client Adapter for EAP** (**rh-sso-7.5.0-eap7-adapter.zip** または最新のバージョン) をダウンロードします。
4. adapter zip を展開してインストールします。インストール手順については、[Red Hat Single Sign On Securing Applications and Services Guide](#) の JBoss EAP Adapter セクションを参照してください。



注記

-Dserver.config=standalone-full.xml プロパティーでアダプターをインストールします。

5. Red Hat JBoss EAP インストールの **EAP_HOME/standalone/configuration** ディレクトリーに移動し、テキストエディターで **standalone-full.xml** ファイルを開きます。
6. 以下の例に表示されているシステムプロパティーを **<system-properties>** に追加します。

```
<system-properties>
  <property name="org.jbpm.workbench.kie_server.keycloak" value="true"/>
  <property name="org.uberfire.ext.security.management.api.userManagementServices"
value="KCAdapterUserManagementService"/>
  <property name="org.uberfire.ext.security.management.keycloak.authServer"
value="http://localhost:8180/auth"/>
</system-properties>
```

7. オプション: クライアントロールを使用する場合は、以下のシステムプロパティーを追加します。

```
<property name="org.uberfire.ext.security.management.keycloak.use-resource-role-
mappings" value="true"/>
```


デフォルトでは、クライアントのリソース名は **kie** です。クライアントリソース名は、RH-SSO でクライアントの設定に使用したクライアント名と同じである必要があります。カスタムのクライアントリソースを使用する場合は、以下のシステムプロパティーも追加します。

```
<property name="org.uberfire.ext.security.management.keycloak.resource"
value="customClient"/>
```

customClient はクライアントリソース名に置き換えます。

8. RH-SSO サブシステム設定を追加します。以下に例を示します。

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
  <secure-deployment name="decision-central.war">
    <realm>demo</realm>
    <realm-public-
key>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCrVrCuTtArbgaZzL1hvh0xtL5mc
7o0NqPVnYXkLvgcwic3BjLGw1tGEGoJaXDuSaRillobm53JBhJx33UNv+5z/UMG4kytBWxheNV
KnL6GgqINabMaFiPLPCF8kAgKnsi79NMo+n6KnSY8YeUmec/p2vjO2NjsSAVcWEQMvHJ31L
wIDAQAB</realm-public-key>
    <auth-server-url>http://localhost:8180/auth</auth-server-url>
    <ssl-required>external</ssl-required>
    <enable-basic-auth>true</enable-basic-auth>
    <resource>kie</resource>
    <credential name="secret">759514d0-dbb1-46ba-b7e7-ff76e63c6891</credential>
    <principal-attribute>preferred_username</principal-attribute>
  </secure-deployment>
</subsystem>
```

この例で、

- **secure-deployment name** は、アプリケーションの WAR ファイルの名前です。
- **realm** は、使用するアプリケーション用に作成したレルムの名前です。
- **realm-public-key** は、作成したレルムの公開鍵です。この鍵は、RH-SSO 管理コンソールで作成したレルムの **Realm settings** ページの **Keys** タブで確認できます。**realm-public-key** の値を指定しない場合は、サーバーが自動的に取得します。
- **auth-server-url** は、RH-SSO 認証サーバーの URL です。
- **enable-basic-auth** は、クライアントがトークンベースと Basic 認証の両方のアプローチを使用して要求を実行できるように、Basic 認証メカニズムを有効にする設定です。
- **resource** は、作成したクライアントの名前です。クライアントロールを使用するには、RH-SSO でクライアントの設定時に使用したクライアントリソース名を設定します。
- **credential name** は、作成したクライアントの秘密鍵です。この鍵は、RH-SSO 管理コンソールの **Clients** ページの **Credentials** タブで確認できます。
- **principal-attribute** は、アプリケーションでユーザー名を表示するための属性です。この値を指定しないと、アプリケーションに、ユーザー名ではなくユーザー ID が表示されます。



注記

RH-SSO サーバーは、ユーザー名を小文字に変換します。したがって、RH-SSO と統合すると、Red Hat Decision Manager ではユーザー名が小文字で表示されます。ユーザー名が、ビジネスプロセスに大文字でハードコードされている場合は、アプリケーションが大文字のユーザー名を識別できない場合があります。

クライアントロールを使用する場合は、**<secure-deployment>** の下に以下の設定も追加します。

```
<use-resource-role-mappings>true</use-resource-role-mappings>
```

9. Elytron サブシステムには、JACC 仕様に基づいた組み込み型ポリシープロバイダーがあります。**standalone.xml**、または Elytron がインストールされているファイルで手動で JACC を有効にするには、以下のタスクのいずれかを実行します。

- ポリシープロバイダーを作成するには、Red Hat JBoss EAP の管理コマンドラインインターフェイス (CLI) で以下のコマンドを入力します。

```
/subsystem=undertow/application-security-domain=other:remove()
/subsystem=undertow/application-security-domain=other:add(http-authentication-
factory="keycloak-http-authentication")
/subsystem=ejb3/application-security-domain=other:write-attribute(name=security-
domain, value=KeycloakDomain)
```

Red Hat JBoss EAP 管理 CLI に関する詳細は、Red Hat JBoss EAP の [Management CLI Guide](#) を参照してください。

- Red Hat JBoss EAP インストールの **EAP_HOME/standalone/configuration** ディレクトリーに移動します。**standalone.xml** ファイルおよび **standalone-full.xml** ファイルで Elytron と undertow サブシステム設定の場所を特定して JACC を有効にします。以下に例を示します。

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0" ... >
...
<application-security-domains>
  <application-security-domain name="other" http-authentication-factory="keycloak-http-
authentication"/>
</application-security-domains>

<subsystem xmlns="urn:jboss:domain:ejb3:9.0">
...
<application-security-domains>
  <application-security-domain name="other" security-domain="KeycloakDomain"/>
</application-security-domains>
```

10. **EAP_HOME/bin/** に移動し、以下のコマンドを実行して Red Hat JBoss EAP サーバーを起動します。

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



注記

RH-SSO セキュリティーサブシステムを使用するようにアプリケーションの WAR ファイルを更新して、Business Central の RH-SSO アダプターを設定することもできます。ただし Red Hat では、RH-SSO サブシステムからアダプターを設定することを推奨します。つまり、設定を各 WAR ファイルに適用するのではなく、Red Hat JBoss EAP の設定を更新します。

18.3. RH-SSO による BUSINESS CENTRAL の外部ファイルシステムおよび GIT リポジトリサービスへのアクセスを可能にする

Business Central が RH-SSO 認証を使用してファイルシステムや Git リポジトリなどの他のリモートサービスを消費できるようにするには、設定ファイルを作成する必要があります。

手順

1. JSON 設定ファイルを生成します。
 - a. **RH-SSO 管理コンソール** (<http://localhost:8180/auth/admin>) に移動します。
 - b. **Clients** をクリックします。
 - c. 以下の設定で新規クライアントを作成します。
 - **Client ID** は **kie-git** に設定します。
 - **Access Type** は **confidential** に設定します。
 - **Standard Flow Enabled** オプションを無効にします。
 - **Direct Access Grants Enabled** オプションを有効にします。

[Clients](#) » [kie-git](#)

Kie-git

Settings	Credentials	Roles	Mappers	Scope	Revocation	Sessions	Offline Access	Clustering	Installation
Client ID	<input type="text" value="kie-git"/>								
Name	<input type="text"/>								
Description	<input type="text"/>								
Enabled	<input checked="" type="checkbox"/> ON <input type="checkbox"/> OFF								
Consent Required	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF								
Client Protocol	<input type="text" value="openid-connect"/>								
Client Template	<input type="text"/>								
Access Type	<input type="text" value="confidential"/>								
Standard Flow Enabled	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF								
Direct Access Grants Enabled	<input checked="" type="checkbox"/> ON <input type="checkbox"/> OFF								
Service Accounts Enabled	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF								
Root URL	<input type="text"/>								
Base URL	<input type="text"/>								
Admin URL	<input type="text"/>								
<input type="button" value="Save"/> <input type="button" value="Cancel"/>									

- d. **Save** をクリックします。
- e. クライアント設定画面の上部にある **Installation** タブをクリックして、**Format Option** に **Keycloak OIDC JSON** を選択します。
- f. **Download** をクリックします。
2. ダウンロードした JSON ファイルを、サーバーのファイルシステム内でアクセス可能なディレクトリに移動するか、アプリケーションクラスパスに追加します。このファイルのデフォルトの名前と場所は **\$EAP_HOME/kie-git.json** です。
3. オプション: **EAP_HOME/standalone/configuration/standalone-full.xml** ファイルの **<system-properties>** タグの下に、以下のシステムプロパティを追加してください。

```
<property name="org.uberfire.ext.security.keycloak.keycloak-config-file"
value="$EAP_HOME/kie-git.json"/>
```

プロパティの **\$EAP_HOME/kie-git.json** 値を、新しい JSON 設定ファイルへの絶対パスまたはクラスパス (**classpath:/EXAMPLE_PATH/kie-git.json**) に置き換えてください。



注記

org.uberfire.ext.security.keycloak.keycloak-config-file プロパティを設定しない場合、Red Hat Decision Manager は **\$EAP_HOME/kie-git.json** ファイルを読み取ります。

結果

RH-SSO サーバーで認証されたすべてのユーザーは、内部の GIT リポジトリをクローンすることができます。以下のコマンドで、**USER_NAME** を RH-SSO ユーザー (**admin** など) に置き換えます。

```
git clone ssh://USER_NAME@localhost:8001/system
```

+



注記

RH-SSO サーバークライアントは、単一のリモートサービスデプロイメントに対して URL を 1 つ使用します。配置設定が 2 つ以上ある場合、以下のようなエラーメッセージが表示されることがあります。

ご迷惑をおかけしております... **無効なパラメーター: redirect_uri**

このエラーを解決するには、クライアント設定の **Valid Redirect URIs** フィールドに **/*** を追加します。

Configure ページで、**Clients > kie-git > Settings** と進み、**Valid Redirect URIs** フィールドに、たとえば、**/*** を追加してください。

```
http://localhost:8080/remote-system/*
```

第19章 RH-SSO を使用した KIE SERVER の認証

KIE Server は、サードパーティークライアントの REST API を提供します。KIE Server と RH-SSO を統合した場合は、サードパーティークライアントのアイデンティティ管理を RH-SSO サーバーに委譲できます。

Red Hat Decision Manager のレルムクライアントを作成して、Red Hat JBoss EAP に RH-SSO クライアントアダプターを設定したら、KIE Server に RH-SSO 認証を設定できます。

前提条件

- [16章RH-SSO のインストールおよび設定](#) の記載通りに、RH-SSO がインストールされている。
- 「[Red Hat Decision Manager ユーザーの追加](#)」 の記載通りに、**kie-server** ロールが割り当てられたユーザーが1つ以上 RH-SSO に追加されている。
- [Red Hat JBoss EAP 7.4 への Red Hat Decision Manager のインストールおよび設定](#) の記載通りに、KIE Server が Red Hat JBoss EAP 7.4 インスタンスにインストールされている。

本章は以下のセクションで設定されます。

- 「[RH-SSO で KIE Server クライアントの作成](#)」
- 「[クライアントアダプターを使用する KIE Server のインストールおよび設定](#)」
- 「[KIE Server のトークンベースの認証](#)」



注記

このセクションは、「[RH-SSO で KIE Server クライアントの作成](#)」を除き、スタンドアロンのインストールが対象です。Red Hat OpenShift Container Platform で RH-SSO と Red Hat Decision Manager を統合する場合には、「[RH-SSO で KIE Server クライアントの作成](#)」の手順を実行して、Red Hat OpenShift Container Platform に Red Hat Decision Manager 環境をデプロイしてください。Red Hat OpenShift Container Platform に Red Hat Decision Manager をデプロイする手順は、[Red Hat OpenShift Container Platform への Red Hat Decision Manager のデプロイメント](#) を参照してください。

19.1. RH-SSO で KIE SERVER クライアントの作成

RH-SSO 管理コンソールを使用して、既存のレルムに KIE Server クライアントを作成します。

前提条件

- [Red Hat JBoss EAP 7.4 への Red Hat Decision Manager のインストールおよび設定](#) の記載通りに、KIE Server が Red Hat JBoss EAP 7.4 サーバーにインストールされている。
- [16章RH-SSO のインストールおよび設定](#) の記載通りに、RH-SSO がインストールされている。
- 「[Red Hat Decision Manager ユーザーの追加](#)」 の記載通りに、**kie-server** ロールが割り当てられたユーザーが1つ以上 RH-SSO に追加されている。

手順

1. RH-SSO 管理コンソールで、[16章RH-SSO のインストールおよび設定](#)で作成したセキュリティーレルムを開きます。

2. **Clients** をクリックし、**Create** をクリックします。
Add Client ページが表示されます。
3. **Add Client** ページで、レلمに KIE Server クライアントを作成するのに必要な情報を入力し、**Save** をクリックします。以下に例を示します。

- クライアント ID: **kie-execution-server**
- Root URL: **http://localhost:8080/kie-server**
- クライアントのプロトコル: **openid-connect**



注記

Red Hat OpenShift Container Platform で RH-SSO を設定している場合は、KIE Server ルートに公開されている URL を入力します。OpenShift 管理者は、必要に応じてこの URL を提供してください。

4. 新規クライアントの **Access Type** は、デフォルトでは **public** に設定されています。この設定を **confidential** に変更して、もう一度 **Save** をクリックします。
5. **Credentials** タブに移動して秘密鍵をコピーします。秘密鍵は、**kie-execution-server** クライアントを設定するのに必要になります。



注記

RH-SSO サーバークライアントは単一の KIE Server デプロイメントに対して URL を 1 つ使用します。配置設定が 2 つ以上ある場合、以下のようなエラーメッセージが表示されることがあります。

ご迷惑をおかけしております... 無効なパラメーター: **redirect_uri**

このエラーを解決するには、クライアント設定の **Valid Redirect URIs** フィールドに ***** を追加します。

Configure ページで、**Clients > kie-execution-server > Settings** と進み、**Valid Redirect URIs** フィールドに、たとえば、***** を追加してください。

http://localhost:8080/kie-server/*

19.2. クライアントアダプターを使用する KIE SERVER のインストールおよび設定

RH-SSO をインストールしたら、Red Hat JBoss EAP に RH-SSO クライアントアダプターをインストールして、KIE Server に対して設定する必要があります。

前提条件

- [Red Hat JBoss EAP 7.4 への Red Hat Decision Manager のインストールおよび設定](#)の記載通りに、KIE Server が Red Hat JBoss EAP 7.4 サーバーにインストールされている。
- [16章RH-SSO のインストールおよび設定](#)の記載通りに、RH-SSO がインストールされている。

- 「[Red Hat Decision Manager ユーザーの追加](#)」の記載通りに、**kie-server** ロールが割り当てられたユーザーが1つ以上 RH-SSO に追加されている。



注記

KIE Server を Business Central 以外のアプリケーションにデプロイする場合には、2 番目のサーバーに RH-SSO をインストールして設定します。

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。
 - **製品:** Red Hat Single Sign-On
 - **バージョン:** 7.5
2. **Red Hat Single Sign-On 7.5 Client Adapter for JBoss EAP** (**rh-sso-7.5.0-eap7-adapter.zip** または最新のバージョン) をダウンロードします。
3. adapter zip を展開してインストールします。インストール手順については、[Red Hat Single Sign On Securing Applications and Services Guide](#) の JBoss EAP Adapter セクションを参照してください。
4. **EAP_HOME/standalone/configuration** に移動して、**standalone-full.xml** ファイルを開きます。
5. 両方のファイルから、**<single-sign-on/>** 要素を削除します。
6. Red Hat JBoss EAP システムの **EAP_HOME/standalone/configuration** ディレクトリーに移動し、**standalone-full.xml** ファイルを編集して RH-SSO サブシステム設定を追加します。以下に例を示します。
7. Red Hat JBoss EAP システムの **EAP_HOME/standalone/configuration** に移動し、**standalone-full.xml** ファイルを編集して RH-SSO サブシステム設定を追加します。以下に例を示します。

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
  <secure-deployment name="kie-server.war">
    <realm>demo</realm>
    <realm-public-
key>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCrVrCuTtArbgaZzL1hvh0xtL5mc
7o0NqPVnYXkLvGcwic3BjLGw1tGEGoJaXDuSaRllobm53JBhx33UNv+5z/UMG4kytBWxheNV
KnL6GgqINabMaFfPLPCF8kAgKnsi79NMo+n6KnSY8YeUmec/p2vjO2NjsSAVcWEQMVhJ31L
wIDAQAB</realm-public-key>
    <auth-server-url>http://localhost:8180/auth</auth-server-url>
    <ssl-required>external</ssl-required>
    <resource>kie-execution-server</resource>
    <enable-basic-auth>true</enable-basic-auth>
    <credential name="secret">03c2b267-7f64-4647-8566-572be673f5fa</credential>
    <principal-attribute>preferred_username</principal-attribute>
  </secure-deployment>
</subsystem>
```



```
<system-properties>
  <property name="org.kie.server.sync.deploy" value="false"/>
</system-properties>
```

この例で、

- **secure-deployment name** は、アプリケーションの WAR ファイルの名前です。
- **realm** は、使用するアプリケーション用に作成したレルムの名前です。
- **realm-public-key** は、作成したレルムの公開鍵です。この鍵は、RH-SSO 管理コンソールで作成したレルムの **Realm settings** ページの **Keys** タブで確認できます。この公開鍵の値を指定しない場合は、サーバーが自動的に取得します。
- **auth-server-url** は、RH-SSO 認証サーバーの URL です。
- **resource** は、作成したサーバークライアントの名前です。
- **enable-basic-auth** は、クライアントがトークンベースと Basic 認証の両方のアプローチを使用して要求を実行できるように、Basic 認証メカニズムを有効にする設定です。
- **credential name** は、作成したサーバークライアントの秘密鍵です。この鍵は、RH-SSO 管理コンソールの **Clients** ページの **Credentials** タブで確認できます。
- **principal-attribute** は、アプリケーションでユーザー名を表示するための属性です。この値を指定しないと、アプリケーションに、ユーザー名ではなくユーザー ID が表示されます。

8. 設定変更を保存します。

9. 以下のコマンドを使用し、Red Hat JBoss EAP サーバーを再起動して KIE Server を実行します。

```
EXEC_SERVER_HOME/bin/standalone.sh -c standalone-full.xml -Dorg.kie.server.id=<ID> -
Dorg.kie.server.user=<USER> -Dorg.kie.server.pwd=<PWD> -Dorg.kie.server.location=
<LOCATION_URL> -Dorg.kie.server.controller=<CONTROLLER_URL> -
Dorg.kie.server.controller.user=<CONTROLLER_USER> -Dorg.kie.server.controller.pwd=
<CONTROLLER_PASSWORD>
```

以下に例を示します。

```
EXEC_SERVER_HOME/bin/standalone.sh -c standalone-full.xml -
Dorg.kie.server.id=kieserver1 -Dorg.kie.server.user=kieserver -
Dorg.kie.server.pwd=password -Dorg.kie.server.location=http://localhost:8080/kie-
server/services/rest/server -Dorg.kie.server.controller=http://localhost:8080/decision-
central/rest/controller -Dorg.kie.server.controller.user=kiecontroller -
Dorg.kie.server.controller.pwd=password
```

10. KIE Server の実行中に、以下のコマンドを実行してサーバーの状態を確認します。<KIE_SERVER_USER> は **kie-server** ロールが割り当てられているユーザーで、そのパスワードは <PASSWORD> です。

```
curl http://<KIE_SERVER_USER>:<PASSWORD>@localhost:8080/kie-
server/services/rest/server/
```

19.3. KIE SERVER のトークンベースの認証

Red Hat Decision Manager と KIE Server 間の通信に、トークンベースの認証を使用することもできます。アプリケーションにおいて、ユーザー名とパスワードの代わりに、完全なトークンをアプリケーションサーバーのシステムプロパティーとして使用できます。ただし、トークンは自動的に更新されないため、アプリケーションの通信が行われている間にトークンが失効しないようにする必要があります。トークンを取得する方法は「[トークンベースの認証](#)」を参照してください。

手順

1. トークンを使用して KIE Server を管理するように Business Central を設定するには、以下を実行します。
 - a. **org.kie.server.token** プロパティーを設定します。
 - b. **org.kie.server.user** プロパティーと **org.kie.server.pwd** プロパティーは設定しないでください。
これで、Red Hat Decision Manager は **Authorization: Bearer \$TOKEN** 認証メソッドを使用します。
2. トークンベースの認証を使用して REST API を使用する場合は、以下を行います。
 - a. **org.kie.server.controller.token** プロパティーを設定します。
 - b. **org.kie.server.controller.user** プロパティーおよび **org.kie.server.controller.pwd** プロパティーは設定しないでください。



注記

KIE Server はトークンを更新できないため、有効期限の長いトークンを使用してください。トークンの有効期限は、2038 年 1 月 19 日以前に設定してください。セキュリティのベストプラクティスで、お使いの環境に適したソリューションかどうかを確認してください。

第20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

Business Central または KIE 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**: Business Central リモートサービスを使用する場合
- **kie-server**: KIE 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

20.1. BASIC 認証

Business Central と KIE 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 ベースのリモトリポジトリエンドポイントの場合:

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

- KIE Server の場合:

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

20.2. トークンベースの認証

よりセキュアな認証オプションを希望される場合は、RH-SSO から付与されたトークンを使用すると、Business Central および KIE Server の両方からリモートサービスを使用できます。

手順

1. RH-SSO 管理コンソールで **Clients** メニューアイテムをクリックし、**Create** をクリックして新規クライアントを作成します。
Add Client ページが表示されます。
2. **Add Client** ページで、レلمにクライアントを新規作成するのに必要な情報を指定します。以下に例を示します。
 - **Client ID**: **kie-remote**
 - **クライアントのプロトコル**: **openid-connect**
3. **Save** をクリックして変更を保存します。

4. Realm Settings でトークンの設定を変更します。

- a. RH-SSO 管理コンソールで、**Realm Settings** メニューアイテムをクリックします。
- b. **Tokens** タブをクリックします。
- c. **Access Token Lifespan** の値を **15** 分に変更します。
これにより、有効期限が切れる前にトークンを取得してサービス呼び出すための十分な時間が得られます。
- d. **Save** をクリックして変更を保存します。

5. リモートクライアントの公開クライアントを作成したら、以下のコマンドを使用して、RH-SSO サーバーのトークンエンドポイントに HTTP 要求を行ってトークンを取得できます。

```
RESULT=`curl --data "grant_type=password&client_id=kie-remote&username=admin&password=password" http://localhost:8180/auth/realms/demo/protocol/openid-connect/token`
```

このコマンドのユーザーは Business Central RH-SSO ユーザーです。詳細は、[「Red Hat Decision Manager ユーザーの追加」](#) を参照してください。

6. RH-SSO サーバーから取得したトークンを表示するには、以下のコマンドを使用します。

```
TOKEN=`echo $RESULT | sed 's/.*access_token":.*/g' | sed 's/".*//g`
```

このトークンを使用してリモートの呼び出しを認証できるようになります。たとえば、Red Hat Decision Manager の内部リポジトリを確認するには、以下のようにトークンを使用します。

```
curl -H "Authorization: bearer $TOKEN" http://localhost:8080/decision-central/rest/repositories
```

付録A バージョン情報

本ドキュメントの最終更新日: 2023 年 2 月 1 日

付録B お問い合わせ先

Red Hat Decision Manager ドキュメントチーム: brms-docs@redhat.com