



Red Hat Process Automation Manager 7.4

Spring Boot での Red Hat Process Automation
Manager ビジネスアプリケーションの作成

Red Hat Process Automation Manager 7.4 Spring Boot での Red Hat Process Automation Manager ビジネスアプリケーションの作成

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

法律上の通知

Copyright © 2019 Red Hat, Inc.

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

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

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

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

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

Linux[®] 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

本ドキュメントでは、Spring Boot スターターを使用して Red Hat Process Automation Manager のビジネスアプリケーションを作成する方法について説明します。

目次

前書き	3
第1章 RED HAT PROCESS AUTOMATION MANAGER ビジネスアプリケーションの作成	4
第2章 ビジネスアプリケーションの作成	5
第3章 ビジネスアプリケーションの構成	7
3.1. ビジネスアプリケーションの認証と承認	7
3.2. APPLICATION.PROPERTIESファイルの設定	7
3.3. 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	10
3.4. QUARTZ を使用したクラスターのビジネスアプリケーションの設定	12
3.5. 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	14
3.6. MYSQL または POSTGRESQL データベースを使用したビジネスアプリケーションの設定	15
3.7. JPA 用のビジネスアプリケーションの設定	16
3.8. SWAGGER ドキュメントの有効化	16
第4章 ビジネスアプリケーションの実行	18
4.1. 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	18
4.2. 開発モードでのビジネスアプリケーションの実行	19
第5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と BUSINESS CENTRAL からのデプ ロイ	21
付録A バージョン情報	23

前書き

開発者は、[business applications](#) の Web サイトから Spring Boot スターターを使用して、Red Hat Process Automation Manager ビジネスアプリケーションをすばやく作成して構成し、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 PROCESS AUTOMATION MANAGER ビジネスアプリケーションの作成

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、Spring Boot スターターは、**pom.xml** ファイルで、このファイルにはアプリケーション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 Process Automation Manager 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 スターターに基づいており、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では、注文管理や宿泊管理など、ドメインごとに固有のビジネス目標を達成できます。

[business application](#) の Web サイトで、Process Automation Manager、Decision Manager または Business Optimizer 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 Process Automation 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. **Business Automation** 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 7.4** を選択し、**Next** をクリックします。



注記

Enterprise 7.4 を選択して、Red Hat Process Automation 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。反対に、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には、**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。
9. テキストエディターで、**<business-application>/business-application-service/src/main/docker/settings.xml** ファイルを開きます。
10. 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>
```

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

```
<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>
```

上記を行うことで、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。

第3章 ビジネスアプリケーションの構成

3.1. ビジネスアプリケーションの認証と承認

デフォルトでは、全 REST エンドポイント (**/rest/** を含む URL) を保護することで、ビジネスアプリケーションのセキュリティを確保します。さらに、ビジネスアプリケーションには認証情報のログが 2 セットあり、ユーザーが開発モードで Business Central に接続できるようになっています。ID が **user**、パスワードが **user** のユーザーと、ID が **kieserver**、パスワードが **kieserver1!** のユーザーがあります。

認証も承認も Spring セキュリティーをもとに行われます。実稼働環境で使用するビジネスアプリケーションはすべて、このセキュリティ設定を変更してください。設定は、**<business-application>/<business-application>-services/src/main/java/com/company/service/DefaultWebSecurityConfig.java** ファイルで変更できます。

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/rest/*").authenticated()
            .and()
            .httpBasic();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        PasswordEncoder encoder = PasswordEncoderFactories.createDelegatingPasswordEncoder();

        auth.inMemoryAuthentication().withUser("kieserver").password(encoder.encode("kieserver1!")).roles("kie-server")
            .and()
            .withUser("john").password(encoder.encode("john@pwd1")).roles("kie-server", "PM", "HR");
    }
}
```

3.2. APPLICATION.PROPERTIES ファイルの設定

ビジネスアプリケーションを作成したら、**application.properties** ファイルで、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。

前提条件

- [business application](#) の Web サイトを使用して作成した **<business-application>.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/src/main/resources** フォルダに移動します。
2. テキストエディターで **application.properties** ファイルを開きます。
3. たとえば、以下のように 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。

```
server.address=localhost
server.port=8090

cxf.path=/rest
```

4. 以下のように、簡単に識別できるように Process Server (**kieserver**) 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/jbpm-console/rest/controller
```

以下の表で、ビジネスアプリケーションに設定可能な Process Server のパラメーターを一覧で紹介します。

表3.1 kieserver パラメーター

パラメーター	値	説明
kieserver.serverId	string	Process Automation 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	Process Automation 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のに使用する名前。 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. 非同期実行を有効にするには、以下のように、**jbpm.executor.enabled** パラメーターの値を **true** に設定し、他の **jbpm.executor** パラメーターをコメントし、必要に応じて値を変更します。

```
jbpm.executor.enabled=true
jbpm.executor.retries=5
jbpm.executor.interval=0
jbpm.executor.threadPoolSize=1
jbpm.executor.timeUnit=SECONDS
```

以下の表で、ビジネスアプリケーションに設定可能なエグゼキューターのパラメーターを一覧で紹介します。

表3.2 エグゼキューターのパラメーター

パラメーター	値	説明
jbpm.executor.enabled	true, false	エグゼキューターコンポーネントを無効または有効にします。
jbpm.executor.retries	整数	ジョブの実行中にエラーが発生した場合の再試行回数を指定します。
jbpm.executor.interval	整数	エグゼキューターがデータベースの同期にかける時間を指定します。時間の単位は jbpm.executor.timeUnit パラメーターで指定します。デフォルトは無効 (値 0) です。
jbpm.executor.threadPoolSize	整数	スレッドプールのサイズを指定します。
jbpm.executor.timeUnit	string	エグゼキューターがデータベースの同期に費やす間隔を計算するときに使用する時間の単位を指定します。値は、 java.util.concurrent.TimeUnit の有効な定数を指定する必要があります。デフォルト値は、 SECONDS です。

6. ビジネスアプリケーションの作成時に **Business Automation** を選択した場合には、以下のコンポーネントの内、ランタイムに起動するコンポーネントを指定します。

表3.3 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。
kieserver.jbpm.enabled	true, false	Red Hat Process Automation Manager コンポーネントを有効または無効にします。

パラメーター	値	説明
--------	---	----

kieserver.jbpmui.enabled	true, false	Red Hat Process Automation Manager UI コンポーネントを有効または無効にします。
kieserver.casemgmt.enabled	true, false	ケース管理のコンポーネントを有効または無効にします。

3.3. 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

Red Hat Single Sign-On (RH SSO) を使用して、サービス間のシングルサインオンを有効化し、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 **<business-application>.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、インストールしている。『[Red Hat Single Sign-On Getting Started Guide](#)』を参照してください。
2. RH SSO 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。
 - 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. 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>

```

4. **application.properties** ファイルを更新します。

```

# 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

```

5. **DefaultWebSecurityConfig.java** ファイルを変更して、Spring Security が RH SSO で正しく動作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.AuthenticationManagerBuilder;
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.RegisterSessionAuthenticationStrategy;
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());
    }
}

```

3.4. QUARTZ を使用したクラスターのビジネスアプリケーションの設定

クラスターでアプリケーションを実行する予定の場合には、Quartz タイマーサービスを設定する必要があります。

前提条件

- [business application](#) の Web サイトを使用して作成した **<business-application>.zip** ファイルがあり、クラスターで使用する予定である。

手順

1. **quartz.properties** ファイルを作成して、以下のコンテンツを追加します。

```

#=====
==
# Configure Main Scheduler Properties
#=====
==
org.quartz.scheduler.instanceName = SpringBootScheduler
org.quartz.scheduler.instanceId = AUTO
org.quartz.scheduler.skipUpdateCheck=true
org.quartz.scheduler.idleWaitTime=1000
#=====
==
# Configure ThreadPool
#=====
==
org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool

```



```

org.quartz.threadPool.threadCount = 5
org.quartz.threadPool.threadPriority = 5
#=====
==
# Configure JobStore
#=====
==
org.quartz.jobStore.misfireThreshold = 60000
org.quartz.jobStore.class=org.quartz.impl.jdbcjobstore.JobStoreCMT
org.quartz.jobStore.driverDelegateClass=org.jbpm.process.core.timer.impl.quartz.Deployments
AwareStdJDBCDelegate
org.quartz.jobStore.useProperties=false
org.quartz.jobStore.dataSource=myDS
org.quartz.jobStore.nonManagedTXDataSource=notManagedDS
org.quartz.jobStore.tablePrefix=QRTZ_
org.quartz.jobStore.isClustered=true
org.quartz.jobStore.clusterCheckinInterval = 5000
#=====
==
# Configure Datasources
#=====
==
org.quartz.dataSource.myDS.connectionProvider.class=org.jbpm.springboot.quartz.SpringConn
ectionProvider
org.quartz.dataSource.myDS.dataSourceName=quartzDataSource
org.quartz.dataSource.notManagedDS.connectionProvider.class=org.jbpm.springboot.quartz.S
pringConnectionProvider
org.quartz.dataSource.notManagedDS.dataSourceName=quartzNotManagedDataSource

```



注記

Quartz 設定ファイルのデータソース名は、Spring bean を参照します。接続プロバイダーは、**org.jbpm.springboot.quartz.SpringConnectionProvider** に設定して、Spring ベースのソースとの統合を有効化する必要があります。

2. **<business-application>/<business-application>-service/src/main/resources/application.properties** ファイルに以下のプロパティを追加して、クラスター化された Quartz タイマーを有効にして、1つ前の手順で作成した **quartz.properties** ファイルのパスを設定します。

```

jbpm.quartz.enabled=true
jbpm.quartz.configuration=quartz.properties

```

3. 以下の内容を **<business-application>/<business-application>-service/src/main/resources/application.properties** ファイルに追加して、管理対象のデータソースと、管理対象外のデータソースを作成します。

```

# enable to use data base as storage
jbpm.quartz.db=true

quartz.datasource.name=quartz
quartz.datasource.username=sa
quartz.datasource.password=sa
quartz.datasource.url=jdbc:h2:./target/spring-boot-jbpm;MVCC=true

```

```
quartz.datasource.driver-class-name=org.h2.Driver

# used to configure connection pool
quartz.datasource.dbcp2.maxTotal=15

# used to initialize quartz schema
quartz.datasource.initialization=true
spring.datasource.schema=classpath*:<QUARTZ_TABLES_H2>.sql
spring.datasource.initialization-mode=always
```

上記の例では、**<QUARTZ_TABLES_H2>** は Quartz H2 データベースのスキーマスクリプトの名前に置き換えます。上記の設定の最後の 3 行で、データベーススキーマを初期化します。

デフォルトでは、Quartz には 2 つのデータソースが必要です。

- デシジョンエンジンまたはプロセスエンジンのトランザクションに参加する管理対象データソース。
- タイマーを検索してトランザクション処理を行わずにトリガーする管理対象外のデータソース。

Red Hat Process Automation Manager ビジネスアプリケーションでは、Quartz データベース (スキーマ) が Red Hat Process Automation Manager テーブルと共存することを想定しているので、Quartz のトランザクション操作に使用するデータソースを生成します。

他の (トランザクション以外) データソースを設定する必要がありますが、主なデータソースと同じデータベースを参照する必要があります。

3.5. 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 Process Automation 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ので、別のコンポーネントとなっています。

前提条件

- [business applications](#) の Web サイトを使用して作成し、ビジネス自動化プロジェクトが含まれる **<business-application>.zip** ファイルがある。

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();
}
```

3.6. MYSQL または POSTGRESQL データベースを使用したビジネスアプリケーションの設定

Red Hat Process Automation Manager ビジネスアプリケーションは、デフォルトの H2 データベースで生成されます。データベースのタイプは、MySQL または PostgreSQL に変更できます。

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。

手順

1. **<business-application>.zip** ファイルを展開して、**<business-application>/business-application-service/src/main/resources** フォルダに移動します。
2. テキストエディターで **application.properties** ファイルを開きます。
3. ビジネスアプリケーションで MySQL データベースを使用するように設定するには、**application.properties** ファイルで以下のパラメーターを検索して、以下の値に変更します。

```
spring.datasource.username=jbpm
spring.datasource.password=jbpm
spring.datasource.url=jdbc:mysql://localhost:3306/jbpm
spring.datasource.driver-class-name=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

4. ビジネスアプリケーションで PostgreSQL データベースを使用するように設定するには、**application.properties** ファイルで以下のパラメーターを検索して、以下の値に変更します。

```
spring.datasource.username=jbpm
spring.datasource.password=jbpm
spring.datasource.url=jdbc:postgresql://localhost:5432/jbpm
```

```
spring.datasource.driver-class-name=org.postgresql.xa.PGXADatasource
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

5. **application.properties** ファイルを保存します。

3.7. JPA 用のビジネスアプリケーションの設定

Java Persistence API (JPA) は、オブジェクトをリレーショナルデータベースにマッピング可能にする標準技術です。Red Hat Process Automation Manager のビジネスアプリケーションには、JPA を設定する必要があります。

前提条件

- [business applications](#) ファイルを使用して作成した Red Hat Process Automation Manager **<business-application>.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/src/main/resources** フォルダに移動します。
2. テキストエディターで **application.properties** ファイルを開きます。
3. **application.properties** ファイルで以下のパラメーターを検索して、以下の値が含まれていることを確認します。

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.properties.hibernate.show_sql=false
spring.jpa.properties.hibernate.hbm2ddl.auto=update
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

4. ビジネスアプリケーションにビジネス自動化機能がある場合は、コンマ区切りの一覧として、パッケージを追加し、エンティティマネージャーファクトリーにエンティティを追加してください。

```
spring.jpa.properties.entity-scan-packages=org.jbpm.springboot.samples.entities
```

ビジネス自動化機能の付いたビジネスアプリケーションは、Red Hat Process Automation Manager に同梱の **persistence.xml** ファイルをもとに、エンティティマネージャーファクトリーを作成します。**org.jbpm.springboot.samples.entities** パッケージにあるエンティティはすべて、エンティティマネージャーファクトリーに自動的に追加され、アプリケーションの他の JPA エンティティと同じものが使用されます。

関連資料

JPA の設定に関する情報は、[Spring Boot Reference Guide](#) を参照してください。

3.8. SWAGGER ドキュメントの有効化

Red Hat Process Automation 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 **<business-application>.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** フォルダーに移動します。
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>3.1.11</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 (オプション)、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4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 上で、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、を実行できます。開発モードで起動したアプリケーションの場合には、Business Central を Process Automation Manager コントローラーとして利用できる必要があります。

4.1. 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。
- 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** フォルダーに移動します。
2. 以下のコマンドの1つを実行します。

表4.1 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。

The **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、プロジェクトの全依存関係がダウンロードされるので、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。

4.2. 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 Process Automation 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

- ビジネスアセットのプロジェクトが含まれる [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。
- 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** フォルダに移動します。
2. 以下のコマンドの1つを実行します。

表4.2 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。

The **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、プロジェクトの全依存関係がダウンロードされるので、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、Process Automation 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5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と BUSINESS CENTRAL からのデプロイ

Red Hat Process Automation 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 Process Automation Manager の Business Central がインストールされている。

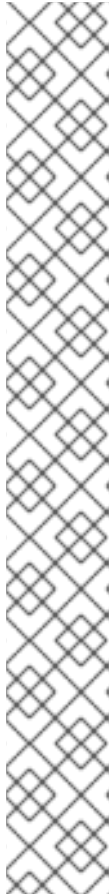
手順

1. **<business-application>/<business-application>-kjar** フォルダーに移動します。
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** を開いて、ビジネスプロセスなどのアセットをご自身の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 Process Server にデプロイせずに設定済みの Maven リポジトリに公開することもできます。開発環境では、**Deploy** をクリックすると、(該当する場合) 実行中のインスタンスを中止することなくビルドされた KJAR ファイルを Process Server にデプロイすることができます。または **Redeploy** をクリックして、ビルドされた KJAR ファイルをデプロイして実行中のインスタンスを中止することもできます。ビルドされた KJAR ファイルを次回にデプロイまたは再デプロイすると、以前のデプロイメントユニット (KIE コンテナ) が同じターゲット Process Server で自動的に更新されます。実稼働環境では **Redeploy** オプションは無効になっており、**Deploy** をクリックして KJAR ファイルを Process Server 上の新規デプロイメントユニット (KIE コンテナ) にデプロイすることのみが可能です。

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

8. デプロイメントを検証するには、**Menu → Deploy → Execution Servers** に移動します。
9. 新たにデプロイしたビジネスアセットと対話するには、**Menu → Manage → Process Definitions**、**Process Instances** の順に移動します。

付録A バージョン情報

本書の最終更新日: 2019 年 8 月 12 日 (月)