Red Hat Process Automation Manager 7.5

Creating Red Hat Process Automation Manager business applications with Spring Boot

Last Updated: 2020-06-05

Red Hat Process Automation Manager 7.5 Creating Red Hat Process Automation Manager business applications with Spring Boot

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

Legal Notice

Abstract

This document describes how to create Red Hat Process Automation Manager business applications using Spring Boot starters.

Table of Contents

PREFACE

As a developer, you can use Spring Boot starters through the business applications website to quickly create Red Hat Process Automation Manager business applications, configure those applications, and deploy them to an existing service or in the cloud.

CHAPTER 1. RED HAT PROCESS AUTOMATION MANAGER BUSINESS APPLICATIONS

Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring Boot is a lightweight framework based on Spring Boot starters. Spring Boot starters are **pom.xml** files that contain a set of dependency descriptors that you can include in your application.

Red Hat Process Automation Manager business applications are flexible, UI-agnostic logical groupings of individual services that provide certain business capabilities. Business applications are based on Spring Boot starters. They are usually deployed separately and can be versioned individually. A complete business application enables a domain to achieve specific business goals, for example order management or accommodation management.

On the business application website you can create a Process Automation Manager, Decision Manager, or Business Optimizer business application. After you create and configure your business application, you can deploy it to an existing service or to the cloud, through OpenShift.

Business applications can contain one or more of the following projects and more than one project of the same type:

- Business assets (KJAR): Contains business processes, rules, and forms and are easily imported into Business Central.
- Data model: Data model projects provide common data structures that are shared between the service projects and business assets projects. This enables proper encapsulation, promotes reuse, and reduces shortcuts. Each service project can expose its own public data model.
- Service: A deployable project that provides the actual service with various capabilities. It includes the business logic that operates your business. In most cases, a service project includes business assets and data model projects. A business application can split services into smaller component service projects for better manageability.

CHAPTER 2. CREATING A BUSINESS APPLICATION

You can use the business application website to quickly and easily create business applications using the Spring Boot framework. Doing this by-passes the need to install and configure Red Hat Process Automation Manager.

Procedure

1. Enter the following URL in a web browser:

```
https://start.jbpm.org
```

2. Click **Configure your business application**
3. Click **Business Automation** and click **Next**.
4. Enter a package and application name.
5. Select **Enterprise 7.5** from the **Version** menu and click **Next**.

NOTE

You must select **Enterprise 7.5** to create a Red Hat Process Automation Manager business application.

6. Select the project types that you want to include in your project. You can include more than one project type.
   - **Business Assets**: Contains business processes, rules, and forms and are easily imported into Business Central. Select **Dynamic Assets** instead if you want to add adaptive and dynamic assets such as cases.
   - **Data Model**: Provides common data structures that are shared between the service projects and business assets projects. This enables proper encapsulation, promotes reuse, and reduces shortcuts. Each service project can expose its own public data model.
   - **Service**: Includes business logic that operates your business.
7. Click **Generate business application**.
   The **<business-application>.zip** file downloads, where **<business-application>** is the name that you entered in the **Application Name** box.
8. Unzip the **<business-application>.zip** file.
9. Open the **<business-application>/business-application-service/src/main/docker/settings.xml** file in a text editor.
10. Add the following repository to the **repositories** element:

```
<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. Add the following plug-in repository to the **pluginRepositories** element:

```
<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>
```

Doing this adds the productized Maven repository to your business application.

CHAPTER 3. BUSINESS APPLICATION CONFIGURATION

3.1. BUSINESS APPLICATION AUTHENTICATION AND AUTHORIZATION

By default, business applications are secured by protecting all REST endpoints (URLs that contain /**rest**/). In addition, business applications have two sets of log in credentials that allow users to connect to Business Central in development mode: the user with the ID **user** and password **user** and the user with the ID **kieserver** and password **kieserver1!**.

Both authentication and authorization is based on Spring security. Alter this security configuration for all business applications used in production environments. You can make configuration changes in the **<business-application>/<business-application>-services/src/main/java/com/company/service/DefaultWebSecurityConfig.java** file:

```
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("ki
e-server")
      .and()
      .withUser("john").password(encoder.encode("john@pwd1")).roles("kie-server", "PM", "HR");
  }
}
```

3.2. CONFIGURING THE APPLICATION.PROPERTIES FILE

After you create your business application, you can configure several components through the **application.properties** file to customize your application.

Prerequisites

- You have a **<business-application>.zip** file that you created using the business application website.

Procedure

1. Unzip the **<business-application>.zip** file and navigate to the **<business-application>/<business-application>-service/src/main/resources** folder.
2. Open the **application.properties** file in a text editor.
3. Configure the host, port, and path for the REST endpoints, for example:

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

cxf.path=/rest
```

4. Configure the Process Server (**kieserver**) so that it can be easily identified, for example:

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

The following table lists the Process Server parameters that you can configure in your business application:

Table 3.1. kieserver parameters

Parameter	Values	Description
kieserver.serverId	string	The ID used to identify the business application when connecting to the Process Automation Manager controller.
kieserver.serverName	string	The name used to identify the business application when connecting to the Process Automation Manager controller. Can be the same string used for the **kieserver.serverId** parameter.
kieserver.location	URL	Used by other components that use the REST API to identify the location of this server. Do not use the location as defined by **server.address** and **server.port**.
kieserver.controllers	URLs	A comma-separated list of controller URLs.

5. To enable asynchronous execution, set the value of the **jbpm.executor.enabled** parameter to **true**, uncomment the other **jbpm.executor** parameters, and change the values as required, for example:

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

The following table lists the executor parameters that you can configure in your business application:

Table 3.2. Executor parameters

Parameter	Values	Description
jbpm.executor.enabled	**true, false**	Disables or enables the executor component.
jbpm.executor.retries	integer	Specifies the number of retries if errors occur while a job is running.
jbpm.executor.interval	integer	Specifies the length of time that the executor uses to synchronize with the database. The unit of time is specified by the **jbpm.executor.timeUnit** parameter. Disabled by default (value **0**).
jbpm.executor.threadPoolSize	integer	Specifies the thread pool size.
jbpm.executor.timeUnit	string	Specifies the time unit used to calculate the interval that the executor uses to synchronize with the database. The value must be a valid constant of **java.util.concurrent.TimeUnit**. The default value is **SECONDS**.

6. If you selected **Business Automation** when you created your business application, specify which of the following components that you want to start at runtime:

Table 3.3. kieserver capabilities parameters

Parameter	Values	Description
kieserver.drools.enabled	**true, false**	Enables or disables the Decision Manager component.
kieserver.dmn.enabled	**true, false**	Enables or disables the Decision Model and Notation (DMN) component.
kieserver.jbpm.enabled	**true, false**	Enables or disables the Red Hat Process Automation Manager component.

Parameter	Values	Description
kieserver.jbpmui.enabled	**true, false**	Enables or disables the Red Hat Process Automation Manager UI component.
kieserver.casemgmt.enabled	**true, false**	Enables or disables the case management component.

3.3. CONFIGURING THE BUSINESS APPLICATION WITH RED HAT SINGLE SIGN-ON

You can use Red Hat Single Sign-On (RH SSO) to enable single sign-on between your services and to have a central place to configure and manage your users and roles.

Prerequisites

- You have a **<business-application>.zip** file that you created using the business applications website.

Procedure

1. Download and install RH SSO. For instructions, see the *Red Hat Single Sign-On Getting Started Guide*.
2. Configure RH SSO:
   a. Either use the default master realm or create a new realm.
   b. Create the **springboot-app** client and set the **AccessType** to public.
   c. Set a valid redirect URI and web origin according to your local setup, for example:
      - Valid redirect URIs: **http://localhost:8090/***
      - Web origin: **http://localhost:8090**
   d. Create realm roles that are used in the application.
   e. Create users that are used in the application and assign roles to them.
3. Add the following dependencies to the service project **pom.xml** file:

```
<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. Update the **application.properties** file:

```
# 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. Modify the **DefaultWebSecurityConfig.java** file to ensure that Spring Security works correctly with RH SSO:

```
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());
    }
}
```

3.4. CONFIGURING THE BUSINESS APPLICATION FOR A CLUSTER USING QUARTZ

If you plan to run your application in a cluster you must configure the Quartz timer service.

Prerequisites

- You have a **<business-application>.zip** file that you created using the business application website, that you want to use in a cluster.

Procedure

1. Create the **quartz.properties** file and add the following content:

```
#============================================================================
==
# 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.DeploymentsAwareStdJDBCDelegate
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.SpringConnectionProvider
org.quartz.dataSource.myDS.dataSourceName=quartzDataSource
org.quartz.dataSource.notManagedDS.connectionProvider.class=org.jbpm.springboot.quartz.SpringConnectionProvider
org.quartz.dataSource.notManagedDS.dataSourceName=quartzNotManagedDataSource
```

NOTE

Data source names in the Quartz configuration file refer to Spring beans. The connection provider must be set to **org.jbpm.springboot.quartz.SpringConnectionProvider** to enable integration with Spring-based data sources.

2. Include the following properties in the **<business-application>/<business-application>-service/src/main/resourcesapplication.properties** file to enable the Quartz clustered timers and set the path of the **quartz.properties** file that you created in the previous step:

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

3. Create a managed and an unmanaged data source by adding the following content to the **<business-application>/<business-application>-service/src/main/resources/application.properties** file:

```
# 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
```

In the preceding example, replace **<QUARTZ_TABLES_H2>** with the name of a Quartz H2 database schema script. The last three lines of the preceding configuration initialize the database schema.

By default, Quartz requires two data sources:

- Managed data source to participate in the transaction of the decision engine or process engine
- Unmanaged data source to look up timers to trigger without any transaction handling

Red Hat Process Automation Manager business applications assume that the Quartz database (schema) will be co-located with Red Hat Process Automation Manager tables and therefore produce data sources used for transactional operations for Quartz.

The other (non transactional) data source must be configured but it should point to the same database as the main data source.

3.5. CONFIGURING BUSINESS APPLICATION USER GROUP PROVIDERS

With Red Hat Process Automation Manager, you can manage human-centric activities. To provide integration with user and group repositories, you can use two KIE API entry points:

- **UserGroupCallback**: Responsible for verifying whether a user or group exists and for collecting groups for a specific user
- **UserInfo**: Responsible for collecting additional information about users and groups, for example email addresses and preferred language

You can configure both of these components by providing alternative code, either code provided out of the box or custom developed code.

For the **UserGroupCallback** component, retain the default implementation because it is based on the security context of the application. For this reason, it does not matter which backend store is used for authentication and authorisation (for example, RH-SSO). It will be automatically used as a source of information for collecting user and group information.

The **UserInfo** component is a separate component because it collects more advanced information.

Prerequisites

- You have a **<business-application>.zip** file that you created using the business application website and that contains a business automation project.

Procedure

1. To provide an alternative implementation of **UserGroupCallback**, add the following code to the Application class or a separate class annotated with **@Configuration**:

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

2. To provide an alternative implementation of **UserInfo**, add the following code to the Application class or a separate class annotated with **@Configuration**:

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

3.6. CONFIGURING A BUSINESS APPLICATION WITH A MYSQL OR POSTGRESQL DATABASE

Red Hat Process Automation Manager business applications are generated with the default H2 database. You can change the database type to MySQL or PostgreSQL.

Prerequisites

- You have a **<business-application>.zip** file that you created using the business applications website.

Procedure

1. Unzip the **<business-application>.zip** file and navigate to the **<business-application>/business-application-service/src/main/resources** folder.
2. Open the **application.properties** file in a text editor.
3. To configure your business application to use a MySQL database, find the following parameters in the **application.properties** file and change the values as shown:

```
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. To configure your business application to use a PostgreSQL database, find the following parameters in the **application.properties** file and change the values as shown:

```
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. Save the **application.properties** file.

3.7. CONFIGURING BUSINESS APPLICATIONS FOR JPA

The Java Persistence API (JPA) is a standard technology that enables you to map objects to relational databases. You must configure JPA for your Red Hat Process Automation Manager business application.

Prerequisites

- You have a Red Hat Process Automation Manager **<business-application>.zip** file that you created using the business applications website.

Procedure

1. Unzip the **<business-application>.zip** file and navigate to the **<business-application>/<business-application>-service/src/main/resources** folder.
2. Open the **application.properties** file in a text editor.
3. Find the following parameters in the **application.properties** file and verify that they have the values shown:

```
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. If your business application has business automation capabilities, you can add entities to the entity manager factory by adding a comma-separated listed of packages:

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

   Business applications with business automation capabilities create an entity manager factory based on the **persistence.xml** file that comes with Red Hat Process Automation Manager. All entities found in the **org.jbpm.springboot.samples.entities** package are automatically added to the entity manager factory and used the same as any other JPA entity in the application.

Additional resources

For more information about configuring JPA, see the *Spring Boot Reference Guide*.

3.8. ENABLING SWAGGER DOCUMENTATION

You can enable Swagger-based documentation for all endpoints available in the service project of your Red Hat Process Automation Manager business application.

Prerequisites

- You have a **<business-application>.zip** file that you created using the business applications website.

Procedure

1. Unzip the **<business-application>.zip** file and navigate to the **<business-application>/<business-application>-service** folder.
2. Open the service project **pom.xml** file in a text editor.
3. Add the following dependencies to the service project **pom.xml** file and save the file.

```
<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. To enable the Swagger UI (optional), add the following dependency to the **pom.xml** file and save the file.

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

5. Open the **<business-application>/<business-application>-service/src/main/resources/application.properties** file in a text editor.
6. Add the following line to the **application.properties** file to enable Swagger support:

```
kieserver.swagger.enabled=true
```

After you start the business application, you can view the Swagger document at **http://localhost:8090/rest/swagger.json**. The complete set of endpoints is available at **http://localhost:8090/rest/api-docs?url=http://localhost:8090/rest/swagger.json**.

CHAPTER 4. BUSINESS APPLICATION EXECUTION

By default, business applications contain a single executable project, the service project. You can execute the service project on Windows or Linux, in standalone (unmanaged) or development (managed) mode. Standalone mode enables you to start your application without additional requirements. Applications started in development mode require Business Central to be available as the Process Automation Manager controller.

4.1. RUNNING BUSINESS APPLICATIONS IN STANDALONE MODE

Standalone (unmanaged) mode enables you to start your business application without additional requirements.

Prerequisites

- You have a **<business-application>.zip** file that you created using the business applications website.
- The business application is configured.

Procedure

1. Navigate to the **<business-application>/<business-application>-service** folder.
2. Enter one of the following commands:

Table 4.1. Standalone launch options

Command	Description
./launch.sh clean install	Launches in standalone mode on Linux or UNIX.
./launch.bat clean install	Launches in standalone mode on Windows.
./launch.sh clean install -Pmysql	Launches in standalone mode on Linux or UNIX if you have configured the application with a MySQL database.
./launch.bat clean install -Pmysql	Launches in standalone mode on Windows if you have configured the application with a MySQL database.
./launch.sh clean install -Ppostgres	Launches in standalone mode on Linux or UNIX if you have configured the application with a postgreSQL database.
./launch.bat clean install -Ppostgres	Launches in standalone mode on Windows if you have configured the application with a PostgreSQL database.

The **clean install** argument directs Maven to build a fresh installation. The projects are then built in the following order:

- Data model
- Business assets
- Service
  The first time that you run the script, it might take a while to build the project because all dependencies of the project are downloaded. At the end of the build, the application starts.

3. Enter the following command to access your business application:

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

4. Enter the credentials **user**/**user** or **kieserver**/**kieserver1!**.

4.2. RUNNING BUSINESS APPLICATIONS IN DEVELOPMENT MODE

Development (managed) mode enables developers to work on a Red Hat Process Automation Manager business application business assets project and dynamically deploy changes to the business application without the need to restart it. In addition, development mode provides a complete monitoring environment for business automation capabilities, for example process instances, tasks, and jobs.

Prerequisites

- You have a **<business-application>.zip** file that contains a business assets project, that you created using the business applications website.
- You configured the business application.
- Business Central is installed and running.

Procedure

1. Navigate to the **<business-application>/<business-application>-service** folder.
2. Enter one of the following commands:

Table 4.2. Managed launch options

Command	Description
./launch-dev.sh clean install	Launches in development mode on Linux or UNIX.
./launch-dev.bat clean install	Launches in development mode on Windows.
./launch-dev.sh clean install -Pmysql	Launches in development mode on Linux or UNIX if you have configured the application with a MySQL database.
./launch-dev.bat clean install -Pmysql	Launches in development mode on Windows if you have configured the application with a MySQL database.

./launch-dev.sh clean install -Ppostgres	Launches in development mode on Linux or UNIX if you have configured the application with a postgreSQL database.
./launch-dev.bat clean install -Ppostgres	Launches in development mode on Windows if you have configured the application with a PostgreSQL database.

The **clean install** argument directs Maven to build a fresh installation. The projects are then built in the following order:

- Data model
- Business assets
- Service
  The first time that you run the script, it might take a while to build the project because all dependencies of the project are downloaded. At the end of the build, the application starts.

3. Enter the following command to access your business application:

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

4. Enter the credentials **user**/**user** or **kieserver**/**kieserver1!**. After the business application starts, it connects to the Process Automation Manager controller and is visible in **Menu → Deploy → Execution Servers** in Business Central.

CHAPTER 5. IMPORTING BUSINESS ASSETS PROJECTS INTO AND DEPLOYING FROM BUSINESS CENTRAL

You can import a business assets project that is part of a Red Hat Process Automation Manager business application into Business Central and then deploy that project to a business application.

Prerequistes

- You have a business application project running in development mode.
- Red Hat Process Automation Manager Business Central is installed.

Procedure

1. Navigate to the **<business-application>/<business-application>-kjar** folder.
2. Execute the following following commands to initialize the Git repository for your project:

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

3. Log in to Business Central and go to **Menu → Design → Projects**.
4. Select **Import Project** and enter following URL:

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

5. Click **Import** and confirm the project to be imported.
6. After the business assets project is imported into Business Central, open the project and click **Add Assets** to add assets such as business processes to your business assets project.
7. Click **Deploy** on your project page to deploy your project to a running business application.

NOTE

You can also select the **Build & Install** option to build the project and publish the KJAR file to the configured Maven repository without deploying to a Process Server. In a development environment, you can click **Deploy** to deploy the built KJAR file to a Process Server without stopping any running instances (if applicable), or click **Redeploy** to deploy the built KJAR file and replace all instances. The next time you deploy or redeploy the built KJAR, the previous deployment unit (KIE container) is automatically updated in the same target Process Server. In a production environment, the **Redeploy** option is disabled and you can click **Deploy** only to deploy the built KJAR file to a new deployment unit (KIE container) on a Process Server.

To configure the Process Server environment mode, set the **org.kie.server.mode** system property to **org.kie.server.mode=development** or **org.kie.server.mode=production**. To configure the deployment behavior for a corresponding project in Business Central, go to project **Settings → General Settings → Version** and toggle the **Development Mode** option. By default, Process Server and all new projects in Business Central are in development mode. You cannot deploy a project with **Development Mode** turned on or with a manually added **SNAPSHOT** version suffix to a Process Server that is in production mode.

8. To verify the deployment, go to **Menu → Deploy → Execution Servers**.
9. To interact with your newly deployed business assets, go to **Menu → Manage → Process Definitions** and **Process Instances**.

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Friday, June 05, 2020.