Red Hat Decision Manager 7.13

Integrating Red Hat Decision Manager with other products and components

Last Updated: 2024-03-14

Red Hat Decision Manager 7.13 Integrating Red Hat Decision Manager with other products and components

Legal Notice

Abstract

This document describes how to integrate Red Hat Decision Manager with other products and components, such as Spring Boot, Red Hat Single Sign-On, and other supported products.

Table of Contents

PREFACE

As a developer or system administrator, You can integrate Red Hat Decision Manager with other products and components, such as Spring Boot, Red Hat Single Sign-On, and other supported products.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see *our CTO Chris Wright's message* .

PART I. CREATING RED HAT DECISION MANAGER BUSINESS APPLICATIONS WITH SPRING BOOT

As a developer, you can create Red Hat Decision Manager Spring Boot business applications through Maven archetype commands, configure those applications, and deploy them to an existing service or in the cloud.

CHAPTER 1. RED HAT DECISION MANAGER SPRING BOOT 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 Spring Boot project.

Red Hat Decision Manager Spring Boot 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. 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.
- Dynamic assets: Contains assets that you can use with case management.
- 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. APACHE MAVEN AND RED HAT DECISION MANAGER SPRING BOOT APPLICATIONS

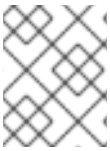
Apache Maven is a distributed build automation tool used in Java application development to create, manage, and build software projects. Maven uses standard configuration files called Project Object Model (POM) files to define projects and manage the build process. POM files describe the module and component dependencies, build order, and targets for the resulting project packaging and output using an XML file. This ensures that the project is built in a correct and uniform manner.

A Maven repository stores Java libraries, plug-ins, and other build artifacts. The default public repository is the Maven 2 Central Repository, but repositories can be private and internal within a company to share common artifacts among development teams. Repositories are also available from third parties.

You can use the online Maven repository with your Spring Boot projects or you can download the Red Hat Decision Manager Maven repository. The recommended approach is to use the online Maven repository with your Spring Boot projects. Maven settings used with a repository manager or repository on a shared server provide better control and manageability of projects.

CHAPTER 3. CONFIGURING THE MAVEN SETTINGS.XML FILE FOR THE ONLINE REPOSITORY

You can use the online Maven repository with your Maven project by configuring your user **settings.xml** file. This is the recommended approach. Maven settings used with a repository manager or repository on a shared server provide better control and manageability of projects.

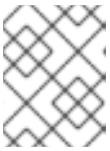


NOTE

When you configure the repository by modifying the Maven **settings.xml** file, the changes apply to all of your Maven projects.

Procedure

1. Open the Maven **~/.m2/settings.xml** file in a text editor or integrated development environment (IDE).



NOTE

If there is not a **settings.xml** file in the **~/.m2/** directory, copy the **settings.xml** file from the **$MAVEN_HOME/.m2/conf/** directory into the **~/.m2/** directory.

2. Add the following lines to the **<profiles>** element of the **settings.xml** file:

```
<!-- Configure the Maven repository -->
<profile>
 <id>red-hat-enterprise-maven-repository</id>
 <repositories>
  <repository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
  </repository>
 </repositories>
 <pluginRepositories>
  <pluginRepository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
  </pluginRepository>
 </pluginRepositories>
</profile>
```

3. Add the following lines to the **<activeProfiles>** element of the **settings.xml** file and save the file.

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

3.1. CREATING A SPRING BOOT BUSINESS APPLICATION FROM MAVEN ARCHETYPES

You can use Maven archetypes to create business applications that use the Spring Boot framework. Doing this by-passes the need to install and configure Red Hat Decision Manager. You can create a business asset project, a data model project, or a service project:

Prerequisites

- Apache Maven 3.5 or higher

Procedure

Enter one of the following commands to create your Spring Boot business application project. In these commands, replace **business-application** with the name of your business application:

- To create a business asset project that contains business processes, rules, and forms:

```
mvn archetype:generate -B -DarchetypeGroupId=org.kie -DarchetypeArtifactId=kie-kjar-archetype -DarchetypeVersion=7.67.0.Final-redhat-00024 -DgroupId=com.company -DartifactId=business-application-kjar -Dversion=1.0-SNAPSHOT -Dpackage=com.company
```

  This command creates a project which generates **business-application-kjar-1.0-SNAPSHOT.jar**.

- To create a data model asset project that provides common data structures that are shared between the service projects and business assets projects:

```
mvn archetype:generate -B -DarchetypeGroupId=org.kie -DarchetypeArtifactId=kie-model-archetype -DarchetypeVersion=7.67.0.Final-redhat-00024 -DgroupId=com.company -DartifactId=business-application-model -Dversion=1.0-SNAPSHOT -Dpackage=com.company.model
```

  This command creates a project which generates **business-application-model-1.0-SNAPSHOT.jar**.

- To create a dynamic assets project that provides case management capabilities:

```
mvn archetype:generate -B -DarchetypeGroupId=org.kie -DarchetypeArtifactId=kie-kjar-archetype -DarchetypeVersion=7.67.0.Final-redhat-00024 -DcaseProject=true -DgroupId=com.company -DartifactId=business-application-kjar -Dversion=1.0-SNAPSHOT -Dpackage=com.company
```

  This command creates a project which generates **business-application-kjar-1.0-SNAPSHOT.jar**.

- To create a service project, a deployable project that provides a service with various capabilities including the business logic that operates your business, enter one of the following commands:
  - Business automation covers features for process management, case management, decision

management and optimization. These will be by default configured in the service project of your business application but you can turn them off through configuration. To create a business application service project (the default configuration) that includes features for process management, case management, decision management, and optimization:

```
mvn archetype:generate -B -DarchetypeGroupId=org.kie -DarchetypeArtifactId=kie-service-spring-boot-archetype -DarchetypeVersion=7.67.0.Final-redhat-00024 -DgroupId=com.company -DartifactId=business-application-service -Dversion=1.0-SNAPSHOT -Dpackage=com.company.service -DappType=bpm
```

- Decision management covers mainly decision and rules related features. To create a decision management service project that includes decision and rules-related features:

```
mvn archetype:generate -B -DarchetypeGroupId=org.kie -DarchetypeArtifactId=kie-service-spring-boot-archetype -DarchetypeVersion=7.67.0.Final-redhat-00024 -DgroupId=com.company -DartifactId=business-application-service -Dversion=1.0-SNAPSHOT -Dpackage=com.company.service -DappType=brm
```

- Business optimization covers planning problems and solutions related features. To create a Red Hat build of OptaPlanner service project to help you solve planning problems and solutions related features:

```
mvn archetype:generate -B -DarchetypeGroupId=org.kie -DarchetypeArtifactId=kie-service-spring-boot-archetype -DarchetypeVersion=7.67.0.Final-redhat-00024 -DgroupId=com.company -DartifactId=business-application-service -Dversion=1.0-SNAPSHOT -Dpackage=com.company.service -DappType=planner
```

These commands create a project which generates **business-application-service-1.0-SNAPSHOT.jar**.

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.

3.2. CONFIGURING AN RED HAT DECISION MANAGER SPRING BOOT PROJECT FOR THE ONLINE MAVEN REPOSITORY

After you create your Red Hat Decision Manager Spring Boot project, configure it with the online Maven Repository to store your application data.

Prerequisites

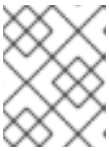
- You have a Spring Boot business application service file that you created using the Maven archetype command. For more information, see Section 3.1, “Creating a Spring Boot business application from Maven archetypes”.

Procedure

1. In the directory that contains your Red Hat Decision Manager Spring Boot application, open the **<BUSINESS-APPLICATION>-service/pom.xml** file in a text editor or IDE, where **<BUSINESS-APPLICATION>** is the name of your Spring Boot project.
2. 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>
```

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



NOTE

If your **pom.xml** file does not have the **pluginRepositories** element, add it as well.

```
<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.

3.3. DOWNLOADING AND CONFIGURING THE RED HAT PROCESS AUTOMATION MANAGER MAVEN REPOSITORY

If you do not want to use the online Maven repository, you can download and configure the Red Hat Process Automation Manager Maven repository. The Red Hat Process Automation Manager Maven repository contains many of the requirements that Java developers typically use to build their applications. This procedure describes how to edit the Maven **settings.xml** file to configure the Red Hat Process Automation Manager Maven repository.

NOTE

When you configure the repository by modifying the Maven **settings.xml** file, the changes apply to all of your Maven projects.

Prerequisites

- You have created a Red Hat Process Automation Manager Spring Boot project.

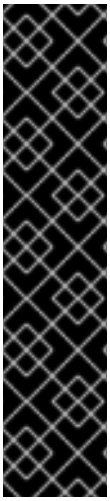
Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required) and then select the following product and version from the drop-down options:
   - **Product:** Process Automation Manager
   - **Version:** 7.13.5
2. Download **Red Hat Process Automation Manager 7.13 Maven Repository**(**rhpam-7.13.5-maven-repository.zip**).
3. Extract the downloaded archive.
4. Change to the **~/.m2/** directory and open the Maven **settings.xml** file in a text editor or integrated development environment (IDE).
5. Add the following lines to the **<profiles>** element of the Maven **settings.xml** file, where ***<MAVEN_REPOSITORY>*** is the path of the Maven repository that you downloaded. The format of ***<MAVEN_REPOSITORY>*** must be **file://$PATH**, for example **file:///home/userX/rhpam-7.13.5.GA-maven-repository/maven-repository**.

```
<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. Add the following lines to the **<activeProfiles>** element of the Maven **settings.xml** file and save the file.

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



IMPORTANT

If your Maven repository contains outdated artifacts, you might encounter one of the following Maven error messages when you build or deploy your project, where ***<ARTIFACT_NAME>*** is the name of a missing artifact and ***<PROJECT_NAME>*** is the name of the project you are trying to build:

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

To resolve the issue, delete the cached version of your local repository located in the ~/**.m2**/**repository** directory to force a download of the latest Maven artifacts.

CHAPTER 4. SPRING SECURITY WITH RED HAT DECISION MANAGER

Spring Security is provided by a collection of servlet filters that make up the Spring Security library. These filters provide authentication through user names and passwords and authorization through roles. The default Spring Security implementation generated in a Red Hat Decision Manager Spring Boot application provides authorization without authentication. This means that anyone with a user name and password valid for the application can access the application without a role.

The servlet filters protect your Spring Boot application against common exploits such as cross-site request forgery (CSRF) and cross-origin resource sharing (CORS). Spring Web relies on the **DispatcherServlet** to redirect incoming HTTP requests to your underlying java REST resources annotated with the **@Controller** annotation. The **DispatchServlet** is agnostic of elements such as security. It is good practice and more efficient to handle implementation details such a security outside of the business application logic. Therefore, Spring uses filters to intercept HTTP requests before routing them to the **DispatchServlet**.

A typical Spring Security implementation consists of the following steps that use multiple servlet filters:

1. Extract and decode or decrypt user credentials from the HTTP request.
2. Complete authentication by validating the credentials against the corporate identity provider, for example a database, a web service, or Red Hat Single Sign-On.
3. Complete authorization by determining whether the authorized user has access rights to perform the request.
4. If the user is authenticated and authorized, propagate the request to the **DispatchServlet**.

Spring breaks these steps down into individual filters and chains them together in a FilterChain. This chaining method provides the flexibility required to work with almost any identity provider and security framework. With Spring Security, you can define a FilterChain for your application programmatically. The following section is from the **business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java** file from a Spring Boot business application service file created using the Maven archetype command. For information, see Section 3.1, “Creating a Spring Boot business application from Maven archetypes” .

```
@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)** Overrides the default **configure(HttpSecurity http)** method and defines a custom FilterChain using the Spring HttpClient fluent API/DSL

- **(2)** Disables common exploit filters for CORS and CSRF tokens for local testing
- **(3)** Requires authentication for any requests made to the pattern 'rest/*' but no roles are defined
- **(4)** Allows basic authentication through the authorization header, for example header 'Authorization: Basic dGVzdF91c2VyOnBhc3N3b3Jk'
- **(5)** Removes the 'X-Frame-Options' header from request/response

This configuration allows any authenticated user to execute the KIE API.

Because the default implementation is not integrated into any external identity provider, users are defined in memory, in the same **DefaultWebSecurityConfg** class. The following section shows the users that are provided when you create a Red Hat Decision Manager Spring Boot business application:

```
@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. USING SPRING SECURITY TO AUTHENTICATE WITH AUTHORIZATION

By default, anyone with a user name and password valid for the Red Hat Decision Manager Spring Boot application can access the application without requiring a role. Spring Security authentication and authorization are derived from the **HTTPSecurity** filter chain configuration. To protect the REST API from users that do not have a specific role mapping, use the Spring Security **.authorizeRequests()** method to match the URLs that you want to authorize.

Prerequisites

- You have a Red Hat Decision Manager Spring Boot application.

Procedure

1. In the directory that contains your Red Hat Decision Manager Spring Boot application, open the **business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java** file in a text editor or IDE.
2. To authorize requests for access by an authenticated user only if they have a specific role, edit the **.antMatchers("/rest/*").authenticated().and()** line in one of the following ways:
   - To authorize for a single role, edit the **antMatchers** method as shown in the following example, where **<role>** is the role that that the user must have for access:

```
@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();
}
...
```

- To authorize a user that has one of a range of roles, edit the **antMatchers** method as shown in the following example, where **<role>** and **<role1>** are each roles the user can have for access:

```
@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();
    }
    ...
```

The **authorizeRequests** method requires authorization of requests for a specific expression. All requests must be successfully authenticated. Authentication is performed using HTTP basic authentication. If an authenticated user tries to access a resource that is protected for a role that they do not have, the user receives an **HTTP 403 (Forbidden)** error.

4.2. DISABLING SPRING SECURITY IN A RED HAT DECISION MANAGER BUSINESS APPLICATION

You can configure Spring Security in a Red Hat Decision Manager business application to provide the security context without authentication.

Prerequisites

- You have a Red Hat Decision Manager Spring Boot application.

Procedure

1. In the directory that contains your Red Hat Decision Manager Spring Boot application, open the **business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java** file in a text editor or integrated development environment (IDE).

2. Edit the **.antMatchers** method as shown in the following example:

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

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

}
```

The **PermitAll** method allows any and all requests for the specified URL pattern.

NOTE

Because no security context is passed in the **HttpServletRequest**, Spring creates an **AnonymousAuthenticationToken** and populates the **SecurityContext** with the **anonymousUser** user with no designated roles other than the **ROLE_ANONYMOUS** role. The user will not have access to many of the features of the application, for example they will be unable to assign actions to group assigned tasks.

4.3. USING SPRING SECURITY WITH PREAUTHENICATION

If you disable Spring Security authentication by using the **PermitAll** method, any user can log in to the application, but users will have limited access and functionality. However, you can preauthenticate a user, for example a designated service account, so a group of users can use the same login but have all of the permissions that they require. That way, you do not need to create credentials for each user.

The easiest way to implement preauthentication is to create a custom filter servlet and add it before the security FilterChain in the **DefaultWebSecurityConfig** class. This way, you can inject a customized, profile-based security context, control its contents, and keep it simple.

Prerequisites

- You have a Red Hat Decision Manager Spring Boot application and you have disabled Spring Security as Section 4.2, “Disabling Spring Security in a Red Hat Decision Manager business application”.

Procedure

1. Create the following class that extends the **AnonymousAuthenticationFilter** class:

```
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.ServletRequest;
```

```
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((HttpServletReques
t) 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. Replace the following variables:

   - Replace **<CLASS_NAME>** with a name for this class, for example **AnonymousAuthFilter**.
   - Replace **<ANONYMOUS_USER>** with a user ID, for example **Service_Group**.
   - Replace **<ROLE>** with the role that has the privileges that you want to give to **<ANONYMOUS_USER>**.

3. If you want to give **<ANONYMOUS_USER>** more than one role, add additional roles as shown in the following example:

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

4. Add **.anonymous().authenticationFilter(new <CLASS_NAME>()).and()** to the **business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java** file, where **<CLASS_NAME>** is the name of the class that you created:

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

Most organizations provide user and group details through single sign-on (SSO) tokens. You can use Red Hat Single Sign-On (RHSSO) 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 Spring Boot business application.

Procedure

1. Download and install RHSSO. For instructions, see the *Red Hat Single Sign-On Getting Started Guide*.

2. Configure RHSSO:

   a. Either use the default master realm or create a new realm.
   A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

   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, as shown in the following 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 element and property to the Spring Boot project **pom.xml** file, where **<KEYCLOAK_VERSION>** is the version of Keycloak that you are using:

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

4. Add the following dependencies to the Spring Boot 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>
```

5. In your Spring Boot project directory, open the **business-application-service/src/main/resources/application.properties** file and add the following lines:

```
# 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. Modify the **business-application-service/src/main/java/com/company/service/DefaultWebSecurityConfig.java** file to ensure that Spring Security works correctly with RHSSO:

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

CHAPTER 5. RED HAT DECISION MANAGER SPRING BOOT CONFIGURATION

After you create your Spring Boot project, you can configure several components to customize your application.

5.1. CONFIGURING REST ENDPOINTS FOR SPRING BOOT APPLICATIONS

After you create your Spring Boot project, you can configure the host, port, and path for the REST endpoint for your Spring Boot application.

Prerequisites

- You have a Spring Boot business application service file that you created using the Maven archetype command. For more information, see Section 3.1, "Creating a Spring Boot business application from Maven archetypes".

Procedure

1. Navigate to the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources** folder, where **<BUSINESS-APPLICATION>** is the name of your Spring Boot project.
2. Open the **application.properties** file in a text editor.
3. Configure the host, port, and path for the REST endpoints, where **<ADDRESS>** is the server address and **<PORT>** is the server port:

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

The following example adds the REST endpoint to the address **localhost** on port **8090**.

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

5.2. CONFIGURING THE KIE SERVER IDENTITY

After you create your Spring Boot project, you can configure KIE Server so that it can be easily identified.

Prerequisites

- You have a Spring Boot business application service file that you created using the Maven archetype command. For more information, see Section 3.1, "Creating a Spring Boot business application from Maven archetypes".

Procedure

1. Navigate to the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources** folder, where **<BUSINESS-APPLICATION>** is the name of your Spring Boot project.
2. Open the **application.properties** file in a text editor.
3. Configure the KIE Server parameters as shown in the following 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 describes the KIE Server parameters that you can configure in your business project:

Table 5.1. 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 it connects 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.

5.3. CONFIGURING KIE SERVER COMPONENTS TO START AT RUNTIME

If you selected **Business Automation** when you created your Spring Boot business application, you can specify which KIE Server components must start at runtime.

Prerequisites

- You have a Spring Boot business application service file that you created using the Maven archetype command. For more information, see Section 3.1, “Creating a Spring Boot business application from Maven archetypes”.

Procedure

1. Navigate to the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources** folder, where **<BUSINESS-APPLICATION>** is the name of your Spring Boot project.
2. Open the **application.properties** file in a text editor.
3. To set a component to start at runtime, set the value of the component to **true.** The following table lists the components that you can set to start at runtime:

Table 5.2. 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.

5.4. CONFIGURING BUSINESS APPLICATION USER GROUP PROVIDERS

With Red Hat Decision 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 Spring Boot business application.

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();
}
```

5.5. ENABLING SWAGGER DOCUMENTATION

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

Prerequisites

- You have a Spring Boot business application.

Procedure

1. Navigate to the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service** folder, where **<BUSINESS-APPLICATION>** is the name of your Spring Boot project.
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 6. CREATING A SELF-CONTAINED RED HAT DECISION MANAGER SPRING BOOT JAR FILE

You can create a single self-contained Red Hat Decision Manager Spring Boot JAR file that contains a complete service, including KIE Server and one or more KJAR files. The Red Hat Decision Manager Spring Boot JAR file does not depend on any KJAR files loading at runtime.

If necessary, the Red Hat Decision Manager Spring Boot JAR file can contain multiple versions of the same KJAR file, including modules. These KJAR files can have the same **artifactID** and **groupID** attribute values, but have different **version** values.

The included KJAR files are separated from any JAR files in the **BOOT-INF/lib** directory to avoid class loader collisions. Each KJAR classpath container file is isolated from other KJAR classpath container files and does not rely on the Spring Boot class loader.

Prerequisites

- You have an existing Red Hat Decision Manager Spring Boot project.
- You have completed development of one or more KJAR files for the project.

Procedure

1. Build all KJAR files for the project. In the default business application, the KJAR source is contained in the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-kjar** directory, where **BUSINESS-APPLICATION** is the name of the business application. Your project might include other KJAR source directories.
   To build the KJAR files, for every KJAR source directory, complete the following steps:
   a. Change to the KJAR source directory.
   b. Enter the following command:

   ```
   mvn install
   ```

   This command builds the KJAR file and places it into the local Maven repository. By default, this repository is located in the **~/.m2/repo** directory.
2. In the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service/src/main/resources** directory, add the following property to your Spring Boot application **application.properties** file:

   ```
   kieserver.classPathContainer=true
   ```

   When this property is set to **true**, KIE Server uses the class loader used by the container to load KJAR files and their dependencies.
3. Complete one of the following actions to ensure that KIE Server loads the necessary KJAR modules:
   - To configure KIE Server to scans and deploy all KJAR modules available in the Spring Boot application, add the following property to the **application.properties** file:

     ```
     kieserver.autoScanDeployments=true
     ```

When this property is set to **true**, KIE Server deploys all KJAR modules available in the application, whether they are declared programmatically or through the Maven plug-in.

This option is the simplest method to include all KJAR modules. However, it has two drawbacks:

- The application sets all container IDs and aliases automatically, based on the group, artifact, and version (GAV) of every KJAR module. You cannot set a custom container ID or alias for a KJAR module.
- At startup time, the application scans the JAR file and the class path for KJAR modules. Therefore, the duration of startup might be increased.

To avoid these drawbacks, you can configure every KJAR module individually using the **application.properties** file or using Java source code, as described in one of the following options.

- To configure every KJAR module individually using the **application.properties** file, for each of the KJAR modules that you want to include in the service, add the following properties to the **application.properties** file:

```
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>
```

Replace the following values:

- **<n>**: A sequential number: **0** for the first KJAR module, **1** for the second module, and so on
- **<container>**: The container ID for the KJAR module
- **<alias>**: The alias for the KJAR module
- **<artifact>**: The artifact ID for the KJAR module
- **<group>**: The group ID for the KJAR module
- **<version>**: The version ID for the KJAR module

The following example configures two versions of the **Evaluation** KJAR module:

```
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
```

- To configure every KJAR module individually using Java source code, create a class in your business application service, similar to the following example:

```
@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;
    }
}
```

For every KJAR module that you want to include, create a **KieContainerResource** bean in this class. The name of the bean is the container name, the first parameter of **KieContainerResource()** is the alias name, and the parameters of **ReleaseId()** are the group ID, artifact ID, and version ID of the KJAR module.

4. Optional: If your business application will run in an Red Hat OpenShift Container Platform pod or in any other environment where the current directory is not writable, add the **spring.jta.log-dir** property to the **application.properties** file and set it to a writable location. For example:

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

This parameter sets the location for the transaction log.

5. In the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service** directory, add the following Maven plug-in in the Spring Boot **pom.xml** file where **<GROUP_ID>**, **<ARTIFACT_ID>**, and **<VERSION>** are the group, artifact, and version (GAV) of a KJAR artifact that your project uses. You can find these values in the **pom.xml** file that is located in the KJAR source directory.

NOTE

You can add more than one version of an artifact.

```
<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>
```

The artifacts required to run the KJAR will be resolved at build time.

The following example adds two version of the **Evaluation** artifact:

```
<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. To build the self-contained Spring Boot image, enter the following command in the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-service** directory:

```
mvn install
```

7. Optional: to run the self-contained Spring Boot image, locate the JAR file in the **target** subdirectory and enter the following command:

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

In this command, replace **<FILENAME>** with the name of the JAR file.

CHAPTER 7. 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.

7.1. RUNNING BUSINESS APPLICATIONS IN STANDALONE MODE

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

Prerequisites

- You have a Spring Boot business application.
- The business application is configured.

Procedure

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

Table 7.1. 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!**.

7.2. RUNNING BUSINESS APPLICATIONS IN DEVELOPMENT MODE

Development (managed) mode enables developers to work on a Red Hat Decision 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 Spring Boot business application.
- You configured the business application.
- Business Central is installed and running.

Procedure

1. Navigate to the **<BUSINESS-APPLICATION>**/**<BUSINESS-APPLICATION>-service** folder, where **<BUSINESS-APPLICATION>** is the name of your Spring Boot project.
2. Enter one of the following commands:

Table 7.2. 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 8. RUNNING A SPRINGBOOT BUSINESS APPLICATION ON RED HAT OPENSHIFT CONTAINER PLATFORM

To run your Red Hat Decision Manager SpringBoot business application on Red Hat OpenShift Container Platform, create an immutable image and push this image to your Red Hat OpenShift Container Platform environment.

Prerequisites

- You have developed a Red Hat Decision Manager SpringBoot business application. For instructions about creating the application, see Section 3.1, "Creating a Spring Boot business application from Maven archetypes".
- If necessary, you have configured Spring security for the application. For instructions about configuring Spring security, see Chapter 4, *Spring Security with Red Hat Decision Manager* .
- You have completed any necessary additional Spring configuration for the business application. For instructions about Spring configuration for your business application, see Chapter 5, *Red Hat Decision Manager Spring Boot configuration*.
- You created a single JAR file for the business application. For instructions about creating a single JAR file for your SpringBoot business application, see Chapter 6, *Creating a self-contained Red Hat Decision Manager Spring Boot JAR file*.
- You are logged on to your Red Hat OpenShift Container Platform environment using the **oc** command and the required project is active.

Procedure

1. Outside the business application project directories, create an **ocp-image** directory with the following subdirectories:

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

2. Copy the single JAR file for your business application into the **root/opt/spring-service** subdirectory. For example:

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

3. In the **ocp-image** directory, create a **Dockerfile** file with the following content:

```
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"]
```

Replace **<FILENAME>.jar** with the name of the single JAR file for your business application.

4. To build the initial image and deploy it in your Red Hat OpenShift Container Platform environment, complete the following steps:

   a. To build the image, run the following commands in the **ocp-image** directory:

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

   Optional: replace **openshift-kie-springboot** with a custom application name in these commands and all subsequent commands.

   b. To deploy the image in the Red Hat OpenShift Container Platform environment, run the following command:

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

   c. Optional: To expose the route for the image, run the following command:

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

5. If you already built and deployed the image and need to update it, for example if you built the JAR file for a new version of Red Hat Decision Manager or of Spring Boot, run the following command in the **ocp-image** directory:

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

CHAPTER 9. IMPORTING AND DEPLOYING BUSINESS ASSETS PROJECTS IN BUSINESS CENTRAL

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

Prerequisites

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

Procedure

1. Navigate to the **<BUSINESS-APPLICATION>/<BUSINESS-APPLICATION>-kjar** folder, where **<BUSINESS-APPLICATION>** is the name of your Spring Boot project.
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 the 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 rules and decision tables 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 KIE Server. In a development environment, you can click **Deploy** to deploy the built KJAR file to a KIE 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 KIE 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 KIE Server.

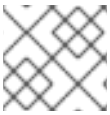
To configure the KIE Server environment mode, set the **org.kie.server.mode** system property to **org.kie.server.mode=development** or **org.kie.server.mode=production**. To configure the deployment behavior for a corresponding project in Business Central, go to project **Settings → General Settings → Version** and toggle the **Development Mode** option. By default, KIE 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 KIE Server that is in production mode.

8. To review project deployment details, click **View deployment details** in the deployment banner at the top of the screen or in the **Deploy** drop-down menu. This option directs you to the **Menu → Deploy → Execution Servers** page.

CHAPTER 10. REPLICATING AUDIT DATA IN A JMS MESSAGE BROKER

You can replicate KIE Server audit data to a Java Message Service (JMS) message broker, for example ActiveMQ or Artemis, and then dump the data in an external database schema so that you can improve the performance of your Spring Boot application by deleting the audit data from your application schema.

If you configure your application to replicate data in a message broker, when an event occurs in KIE Server the record of that event is stored in the KIE Server database schema and it is sent to the message broker. You can then configure an external service to consume the message broker data into an exact replica of the application's database schema. The data is appended in the message broker and the external database every time an event is produce by KIE Server.



NOTE

Only audit data is stored in the message broker. No other data is replicated.

Prerequisites

- You have an existing Red Hat Decision Manager Spring Boot project.

Procedure

1. Open the Spring Boot application's **pom.xml** file in a text editor.
2. Add the KIE Server Spring Boot audit dependency to the **pom.xml** file:

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

3. Add the dependency for your JMS client. The following example adds the Advanced Message Queuing Protocol (AMQP) dependency:

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

4. Add the JMS pool dependency:

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

5. To configure KIE Server audit replication to use queues, complete the following tasks:
   a. Add the following lines to your Spring Boot application's **application.properties** file:

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

b. Add the properties required for your message broker client. The following example shows how to configure KIE Server for AMPQ, where **<JMS_HOST_PORT>** is the port that the broker listens on and **<USERNAME>** and **<PASSWORD** are the login credentials for the broker:

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

c. Add the following lines to the **application.properties** file of the service that will consume the message broker data:

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

d. Add the properties required for your message broker client to the **application.properties** file of the service that will consume the message broker data. The following example shows how to configure KIE Server for AMPQ, where **<JMS_HOST_PORT>** is the port that your message broker listens on and **<USERNAME>** and **<PASSWORD>** are the login credentials for the message broker:

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

6. To configure KIE Server audit replication to use topics, complete the following tasks:

a. Add the following lines to your Spring Boot application's **application.properties** file:

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

b. Add the properties required for your message broker client to the **application.properties** file of the service that will consume the message broker data. The following example shows how to configure KIE Server for AMPQ, where **<JMS_HOST_PORT>** is the port that your message broker listens on and **<USERNAME>** and **<PASSWORD** are the login credentials for the message broker:

```
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. Add the following lines to the **application.properties** file of the service that will consume the message broker data:

```
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. Add the properties required for your message broker client to the **application.properties** file of the service that will consume the message broker data. The following example shows how to configure KIE Server for AMPQ, where **<JMS_HOST_PORT>** is the port that your message broker listens on and **<USERNAME>** and **<PASSWORD>** are the login credentials for the message broker:

```
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. Optional: To configure the KIE Server that contains the replicated data to be read only, set the **org.kie.server.rest.mode.readonly** property in the **application.properties** file to **true**:

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

Additional resources

- Section 10.1, "Spring Boot JMS audit replication parameters"

10.1. SPRING BOOT JMS AUDIT REPLICATION PARAMETERS

The following table describes the parameters used to configure JMS audit replication for Red Hat Decision Manager applications on Spring Boot.

Table 10.1. Spring Boot JMS audit replication parameters

Parameter	Values	Description
kieserver.audit-replication.producer	**true, false**	Specifies whether the business application will act as a producer to replicate and send the JMS messages to either a queue or a topic.
kieserver.audit-replication.consumer	**true, false**	Specifies whether the business application will act as a consumer to receive the JMS messages from either a queue or a topic.
kieserver.audit-replication.queue	**string**	The name of the JMS queue to either send or consume messages.
kieserver.audit-replication.topic	**string**	The name of the JMS topic to either send or consume messages.
kieserver.audit-replication.topic.subscriber	**string**	The name of the topic subscriber.

Parameter	Values	Description
org.kie.server.rest.mode.readonly	**true, false**	Specifies read only mode for the business application.

PART II. INTEGRATING RED HAT FUSE WITH RED HAT DECISION MANAGER

As a system administrator, you can integrate Red Hat Decision Manager with Red Hat Fuse on Red Hat JBoss Enterprise Application Platform to facilitate communication between integrated services.

CHAPTER 11. RED HAT FUSE AND RED HAT DECISION MANAGER

Red Hat Fuse is a distributed, cloud-native integration platform that is part of an agile integration solution. Its distributed approach enables teams to deploy integrated services where required. Fuse has the flexibility to service diverse users, including integration experts, application developers, and business users, each with their own choice of deployment, architecture, and tooling. The API-centric, container-based architecture decouples services so they can be created, extended, and deployed independently. The result is an integration solution that supports collaboration across the enterprise.

Red Hat Decision Manager is an open source decision management platform that combines business rules management, complex event processing, Decision Model & Notation (DMN) execution, and Red Hat build of OptaPlanner for solving planning problems. It automates business decisions and makes that logic available to the entire business.

Business assets such as rules, decision tables, and DMN models are organized in projects and stored in the Business Central repository. This ensures consistency, transparency, and the ability to audit across the business. Business users can modify business logic without requiring assistance from IT personnel.

You can install Red Hat Fuse on the Apache Karaf container platform and then install and configure Red Hat Process Automation Manager in that container.

You can also install Red Hat Fuse on a separate instance of Red Hat JBoss Enterprise Application Platform and integrate it with Red Hat Process Automation Manager. The **kie-camel** module provides integration between Red Hat Fuse and Red Hat Process Automation Manager.

IMPORTANT

For the version of Red Hat Fuse that Red Hat Decision Manager 7.13 supports, see Red Hat Decision Manager 7 Supported Configurations.

NOTE

You can install Red Hat Fuse on Spring Boot. Red Hat Decision Manager provides no special integration for this scenario.

You can use the **kie-server-client** library in an application running on Red Hat Fuse on Spring Boot to enable communication with Red Hat Decision Manager services running on a KIE Server.

For instructions about using the **kie-server-client** library, see *Interacting with Red Hat Decision Manager using KIE APIs*.

CHAPTER 12. RED HAT DECISION MANAGER DECISION ENGINE WITH FUSE ON APACHE KARAF

Apache Karaf is a standalone open-source runtime environment. It is based on the OSGi standard from the OSGi Alliance. Karaf provides support for modularisation through OSGi bundles with sophisticated class-loading support. You can deploy multiple versions of a dependency side by side in a Karaf container. You can use hot code swapping to upgrade or replace a module without shutting down the container.

Red Hat Decision Manager integration with Fuse on Karaf is provided through Karaf features. You can install individual components of Red Hat Decision Manager for Fuse on Karaf using these features.

Features files are XML files that specify which OSGI bundles are installed for a particular feature. The following features XML files facilitate Red Hat Decision Manager and Fuse on Karaf integration:

- **rhba-features-<FUSE-VERSION>-features.xml**
  This file is a part of Fuse installed in Karaf where **<FUSE-VERSION>** is the version of Fuse. This file is stored in the Karaf system repository, in the **system/org/jboss/fuse/features/rhba-features** directory. This file contains prerequisites for installing Red Hat Decision Manager features.

- **kie-karaf-features-7.67.0.Final-redhat-00024-features-fuse.xml**
  This file is a part of Red Hat Decision Manager and provides Red Hat Decision Manager features, which define the OSGi features that can be deployed into Red Hat Fuse. OSGi users can install features from this file to install Red Hat Decision Manager into Fuse and use it in their applications. You can find this features file in the online and offline Maven repository that is distributed with Red Hat Decision Manager. The group ID, artifact ID, and version (GAV) identifier of this file is **org.kie:kie-karaf-features:7.67.0.Final-redhat-00024**.

12.1. UNINSTALLING OBSOLETE RED HAT DECISION MANAGER FEATURES XML FILES ON KARAF

If your installation contains older versions of the Red Hat Decision Manager features XML files (for example, **kie-karaf-features-<VERSION>-features.xml**), you must remove these files and all associated files before installing the most recent features XML files.

Prerequisites

- Obsolete features XML files exist in your Apache Karaf installation.

Procedure

1. Enter the following commands to determine whether your installation contains obsolete Red Hat Decision Manager features XML files:

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

2. Enter the following command, where **<FUSE_HOME>** is the Fuse installation directory, to start the Red Hat Fuse console:

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

3. Enter the following command, where **<FEATURE_NAME>** is the name of the feature that you want to uninstall, to uninstall features or applications that use obsolete features XML files:

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

The following example shows how to remove features:

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

4. Search Karaf home for references to bundles that use **drools**, **kie**, or **jbpm**. The following example shows how to use **grep** to search for these components:

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

The example shows the output from these commands:

```
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
```

5. Enter the following command, where **BUNDLE_ID** is a bundle ID returned in the search, to remove the bundles found in the previous step:

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

6. Enter the following command to remove the obsolete **drools-karaf-features** URL:

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

7. Restart Fuse.

12.2. INSTALLING RED HAT DECISION MANAGER FEATURES ON KARAF USING XML FILES

You can install Red Hat Decision Manager features on Karaf to create a dynamic runtime environment for your Red Hat Decision Manager processes.

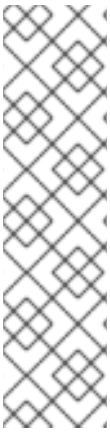
Prerequisites

- A Red Hat Fuse installation in an Apache Karaf container is available. For information about installing Fuse in Apache Karaf, see *Installing Red Hat Fuse on the Apache Karaf container* .
- You have removed any obsolete Red Hat Decision Manager features XML files as described in Section 12.1, “Uninstalling obsolete Red Hat Decision Manager features XML files on Karaf” .

Procedure

To install Red Hat Decision Manager features, enter the following command:

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



NOTE

Use **org.drools.osgi.spring.OsgiKModuleBeanFactoryPostProcessor** instead of **org.kie.spring.KModuleBeanFactoryPostProcessor** to postprocess KIE elements in an OSGi environment.

Do not install the **drools-module** feature before the **kie-spring** feature. If you do, the **drools-compiler** bundle will not detect packages exported by **kie-spring**.

If you install the features in the incorrect order, run **osgi:refresh drools-compiler_bundle_ID** to force the **drools-compiler** to rebuild its **Import-Package** metadata.

In this command, **<FEATURE_NAME>** is one of the features listed in Section 12.4, "Red Hat Decision Manager Karaf features".

12.3. INSTALLING RED HAT DECISION MANAGER FEATURES ON KARAF THROUGH MAVEN

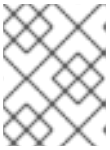
Install Red Hat Decision Manager with Fuse on Apache Karaf to deploy integrated services where required.

Prerequisites

- A Red Hat Fuse 7.12 on Apache Karaf installation exists. For installation instructions, see *Installing Red Hat Fuse on the Apache Karaf container* .
- Any obsolete features XML files have been removed, as described in Section 12.1, "Uninstalling obsolete Red Hat Decision Manager features XML files on Karaf".

Procedure

1. To configure the Maven repository, open the **FUSE_HOME/etc/org.ops4j.pax.url.mvn.cfg** file in a text editor.
2. Make sure that the **https://maven.repository.redhat.com/ga/** repository is present in the **org.ops4j.pax.url.mvn.repositories** variable and add it if necessary.



NOTE

Separate entries in the **org.ops4j.pax.url.mvn.repositories** variable with a comma, space, and backslash (**,** \). The backslash forces a new line.

3. To start Fuse, enter the following command, where **FUSE_HOME** is the Fuse installation directory:

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

4. To add a reference to the features file that contains installation prerequisites, enter the following command, where **<FUSE_VERSION** is the version of Fuse that you are installing:

-

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

5. Enter the following command to add a reference to the Red Hat Decision Manager features XML file:

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

To see the current **drools-karaf-features** version, see the Red Hat Decision Manager 7 Supported Configurations page.

6. Enter the following command to install a feature provided by Red Hat Decision Manager features XML file. In this command, **<FEATURE_NAME>** is one of the features listed in Section 12.4, "Red Hat Decision Manager Karaf features".

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

7. Enter the following command to verify the installation:

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

Successfully installed features have the status **started**.

12.4. RED HAT DECISION MANAGER KARAF FEATURES

The following table lists Red Hat Decision Manager Karaf features.

Feature	Description
drools-module	Contains the core and compiler of Drools, used to create KIE bases and KIE sessions from plain DRL. It also contains the implementation of the executable model. Uses Drools for rules evaluation, without requiring persistence, processes, or decision tables.
drools-template	Contains the Drools templates.
drools-jpa	Uses Drools for rules evaluation with persistence and transactions, but without requiring processes or decision tables. The **drools-jpa** feature includes the **drools-module**. However, you might also need to install the **droolsjbpm-hibernate** feature or ensure that a compatible hibernate bundle is installed.
drools-decisiontable	Uses Drools with decision tables.
Core engine JARs and **kie-ci**	Uses Red Hat Decision Manager with the KIE scanner (**kie-ci**) to download kJARs from a Maven repository.

Feature	Description
kie-camel	Provides the **kie-camel** component, an Apache Camel endpoint that integrates Fuse with Red Hat Decision Manager.
kie-spring	Installs the **kie-spring** component that enables you to configure listeners to KIE sessions using XML tags.

CHAPTER 13. INSTALLING FUSE ON RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

Install Red Hat Fuse 7.12 on Red Hat JBoss EAP 7.4 to integrate it with Red Hat Decision Manager.

Prerequisites

- A Red Hat Decision Manager installation on Red Hat JBoss Enterprise Application Platform 7.4 is available. For installation instructions, see *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.4*.
- A separate instance of Red Hat JBoss Enterprise Application Platform 7.4 is available.

Procedure

1. Install Red Hat Fuse 7.12 on Red Hat JBoss Enterprise Application Platform 7.4. For installation instructions, see the *Installing on JBoss EAP* section in Red Hat Fuse documentation.
2. Open the **pom.xml** file in the Fuse home directory in a text editor.
3. Create the integration project with a dependency on the **kie-camel** component by editing the **pom.xml** file as shown in the following example:

```
<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>
<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>
```

CHAPTER 14. THE KIE-CAMEL COMPONENT

The **kie-camel** component is an Apache Camel endpoint provided by Red Hat Fuse that integrates Fuse with Red Hat Decision Manager. It enables you to specify a Red Hat Decision Manager module by using a Maven group ID, artifact ID, and version (GAV) identifier which you can pull into the route and execute. It also enables you to specify portions of the message body as facts. You can use the **kie-camel** component with embedded engines or with KIE Server.

Embedded engines

In this scenario, KIE engines run in the same container as the Fuse integration project and you can communicate with engines using KIE commands. To create the Camel producer, use the following URI:

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

For example, enter the following command to initialize a Camel route in Spring:

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

KIE Server

In this scenario, the **kie-camel** component connects to KIE Server using the KIE Server REST API. This enables users to communicate with KIE Server using the KIE Server API. To create a producer, use the following URI:

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

For example, enter the following command to initialize a Camel route in Spring:

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

The message has the following headers:

Table 14.1. Message headers and descriptions

Header	Description
CamelKieClient	KIE Server client (mandatory)
CamelKieOperation	KIE Server client (mandatory)
CamelKieParameterName	The value of the client method parameter (optional)
CamelKieBodyParam	The method parameter where the message body is stored (optional)

PART III. INTEGRATING RED HAT DECISION MANAGER WITH RED HAT SINGLE SIGN-ON

As a system administrator, you can integrate Red Hat Single Sign-On with Red Hat Decision Manager to secure your Red Hat Decision Manager browser applications with a single authentication method.

Prerequisites

- Red Hat Decision Manager is installed on Red Hat JBoss EAP 7.4. For information, see *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.4*.

CHAPTER 15. INTEGRATION OPTIONS

Red Hat Single Sign-On (RH-SSO) is a single sign-on solution that you can use to secure your browser applications with your REST web services and Git access.

When you integrate Red Hat Decision Manager with RH-SSO, you create an SSO and identity management (IDM) environment for Red Hat Decision Manager. The session management feature of RH-SSO enables you to use a single authentication for different Red Hat Decision Manager environments on the internet.

The following chapters describe how you can integrate RH-SSO with Red Hat Decision Manager:

- **Chapter 18, *Authenticating Business Central through RH-SSO***
  To authenticate Red Hat Decision Manager through an RH-SSO server, you must secure both the Red Hat Decision Manager web client (Business Central) and remote services through RH-SSO.This integration enables you to connect to Red Hat Decision Manager through RH-SSO using either Business Central or a remote service consumer.

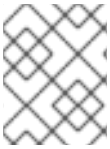
- **Chapter 19, *Authenticating KIE Server through RH-SSO***
  To authenticate KIE Server through an RH-SSO server, you must secure the remote services provided by KIE Server. Doing this enables any remote Red Hat Decision Manager service consumer (user or a service) to authenticate through RH-SSO. Note that KIE Server does not have a web interface.

- **Chapter 20, *Authenticating third-party clients through RH-SSO***
  If Business Central or KIE Server are using RH-SSO, third-party clients must authenticate themselves using RH-SSO. After authentication, they can consume the remote service endpoints provided by Business Central and KIE Server, such as the REST API or remote file system services.

To facilitate LDAP integration with Red Hat Decision Manager, consider using RH-SSO with LDAP. For information, see the "LDAP and Active Directory" section of the *Red Hat Single Sign-On Server Administration Guide*.

CHAPTER 16. INSTALLING AND CONFIGURING RH-SSO

A realm is a security policy domain defined for a web or application server. Security realms are used to restrict access for different application resources. You should create a new realm whether your RH-SSO instance is private or shared with other products. You can keep the master realm as a place for super administrators to create and manage the realms in your system. If you are integrating with an RH-SSO instance that is shared with other product installations to achieve single sign-on with those applications, all of those applications must use the same realm. To create an RH-SSO realm, download, install, and configure RH-SSO 7.5.



NOTE

If Business Central and KIE Server are installed on different servers, complete this procedure on both servers.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required) and then select the product and version from the drop-down options:
   - **Product:** Red Hat Single Sign-On
   - **Version:** 7.5
2. Download **Red Hat Single Sign-On 7.5.0 Server**(**rh-sso-7.5.0.zip**) and the latest server patch.
3. To install and configure a basic RH-SSO standalone server, follow the instructions in the *Red Hat Single Sign On Getting Started Guide*. For advanced settings for production environments, see the *Red Hat Single Sign On Server Administration Guide* .

NOTE

If you want to run both RH-SSO and Red Hat Decision Manager servers on the same system, ensure that you avoid port conflicts by taking one of the following actions:

- Update the ***RHSSO_HOME*/standalone/configuration/standalone-full.xml** file and set the port offset to 100. For example:

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

- Use an environment variable to set the port offset when running the server:

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

CHAPTER 17. RED HAT DECISION MANAGER ROLES AND USERS

To access Business Central or KIE Server, you must create users and assign them appropriate roles before the servers are started. You can create users and roles when you install Business Central or KIE Server.

If both Business Central and KIE Server are running on a single instance, a user who is authenticated for Business Central can also access KIE Server.

However, if Business Central and KIE Server are running on different instances, a user who is authenticated for Business Central must be authenticated separately to access KIE Server. For example, if a user who is authenticated on Business Central but not authenticated on KIE Server tries to view or manage process definitions in Business Central, a 401 error is logged in the log file and the **Invalid credentials to load data from remote server. Contact your system administrator.** message appears in Business Central.

This section describes Red Hat Decision Manager user roles.

NOTE

The **admin**, **analyst**, and **rest-all** roles are reserved for Business Central. The **kie-server** role is reserved for KIE Server. For this reason, the available roles can differ depending on whether Business Central, KIE Server, or both are installed.

- **admin**: Users with the **admin** role are the Business Central administrators. They can manage users and create, clone, and manage repositories. They have full access to make required changes in the application. Users with the **admin** role have access to all areas within Red Hat Decision Manager.
- **analyst**: Users with the **analyst** role have access to all high-level features. They can model projects. However, these users cannot add contributors to spaces or delete spaces in the **Design → Projects** view. Access to the **Deploy → Execution Servers** view, which is intended for administrators, is not available to users with the **analyst** role. However, the **Deploy** button is available to these users when they access the Library perspective.
- **rest-all**: Users with the **rest-all** role can access Business Central REST capabilities.
- **kie-server**: Users with the **kie-server** role can access KIE Server REST capabilities.

17.1. ADDING RED HAT DECISION MANAGER USERS

Before you can use RH-SSO to authenticate Business Central or KIE Server, you must add users to the realm that you created. To add new users and assign them a role to access Red Hat Decision Manager, complete the following steps:

1. Log in to the RH-SSO Admin Console and open the realm that you want to add a user to.
2. Click the **Users** menu item under the **Manage** section.
   An empty user list appears on the **Users** page.
3. Click the **Add User** button on the empty user list to start creating your new user.
   The **Add User** page opens.

4. On the **Add User** page, enter the user information and click **Save**.
5. Click the **Credentials** tab and create a password.
6. Assign the new user one of the roles that allows access to Red Hat Decision Manager. For example, assign the **admin** role to access Business Central or assign the **kie-server** role to access KIE Server.

NOTE

For projects that deploy from Business Central on OpenShift, create an RH-SSO user called **mavenuser** without any role assigned, then add this user to the **BUSINESS_CENTRAL_MAVEN_USERNAME** and **BUSINESS_CENTRAL_MAVEN_PASSWORD** in your OpenShift template.

7. Define the roles as realm roles in the **Realm Roles** tab under the **Roles** section. Alternatively, for roles used in Business Central, you can define the roles as client roles for the **kie** client. For instructions about configuring the **kie** client, see Section 18.1, "Creating the Business Central client for RH-SSO". To use client roles, you must also configure additional settings for Business Central, as described in Section 18.2, "Installing the RH-SSO client adapter for Business Central".

   You must define roles used in KIE Server as realm roles.
8. Click the **Role Mappings** tab on the **Users** page to assign roles.

CHAPTER 18. AUTHENTICATING BUSINESS CENTRAL THROUGH RH-SSO

This chapter describes how to authenticate Business Central through RH-SSO. It includes the following sections:

- Section 18.1, "Creating the Business Central client for RH-SSO"
- Section 18.2, "Installing the RH-SSO client adapter for Business Central"
- Section 18.3, "Enabling access to external file systems and Git repository services for Business Central using RH-SSO"

Prerequisites

- Business Central is installed in a Red Hat JBoss EAP 7.4 server, as described in *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.4*.
- RH-SSO is installed as described in Chapter 16, *Installing and configuring RH-SSO* .
- You added Business Central users to RH-SSO as described in Section 17.1, "Adding Red Hat Decision Manager users".
- Optional: To manage RH-SSO users from Business Central, you added all realm-management client roles in RH-SSO to the Business Central administrator user.

NOTE

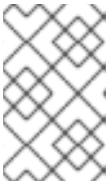
Except for Section 18.1, "Creating the Business Central client for RH-SSO" , this section is intended for standalone installations. If you are integrating RH-SSO and Red Hat Decision Manager on Red Hat OpenShift Container Platform, complete only the steps in Section 18.1, "Creating the Business Central client for RH-SSO" and then deploy the Red Hat Decision Manager environment on Red Hat OpenShift Container Platform. For information about deploying Red Hat Decision Manager on Red Hat OpenShift Container Platform, see Deploying Red Hat Decision Manager on Red Hat OpenShift Container Platform.

18.1. CREATING THE BUSINESS CENTRAL CLIENT FOR RH-SSO

After the RH-SSO server starts, use the RH-SSO Admin Console to create the Business Central client for RH-SSO.

Procedure

1. Enter **http://localhost:8180/auth/admin** in a web browser to open the RH-SSO Admin Console and log in using the admin credentials that you created while installing RH-SSO.



NOTE

If you are configuring RH-SSO with Red Hat OpenShift Container Platform, enter the URL that is exposed by the RH-SSO routes. Your OpenShift administrator can provide this URL if necessary.

When you login for the first time, you can set up the initial user on the new user registration form.

2. In the RH-SSO Admin Console, click the **Realm Settings** menu item.
3. On the **Realm Settings** page, click **Add Realm**.
   The **Add realm** page opens.
4. On the **Add realm** page, provide a name for the realm and click **Create**.
5. Click the **Clients** menu item and click **Create**.
   The **Add Client** page opens.
6. On the **Add Client** page, provide the required information to create a new client for your realm. For example:
   - **Client ID**: kie
   - **Client protocol**: openid-connect
   - Root URL: **http://*localhost*:8080/business-central**

NOTE

If you are configuring RH-SSO with Red Hat OpenShift Container Platform, enter the URL that is exposed by the KIE Server routes. Your OpenShift administrator can provide this URL if necessary.

7. Click **Save** to save your changes.
   After you create a new client, its **Access Type** is set to **public** by default. Change it to **confidential**.

   The RH-SSO server is now configured with a realm with a client for Business Central applications and running and listening for HTTP connections at **localhost:8180**. This realm provides different users, roles, and sessions for Business Central applications.

NOTE

The RH-SSO server client uses one URL to a single business-central deployment. The following error message might be displayed if there are two or more deployment configurations:

We are sorry... **Invalid parameter: redirect_uri**

To resolve this error, append **/*** to the **Valid Redirect URIs** field in the client configuration.

On the **Configure** page, go to **Clients** > **kie** > **Settings**, and append the **Valid Redirect URIs** field with **/***, for example:

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

18.2. INSTALLING THE RH-SSO CLIENT ADAPTER FOR BUSINESS CENTRAL

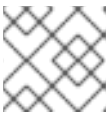
After you install RH-SSO, you must install the RH-SSO client adapter for Red Hat JBoss EAP and configure it for Business Central.

Prerequisites

- Business Central is installed in a Red Hat JBoss EAP 7.4 instance, as described in *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.4*.
- RH-SSO is installed as described in Chapter 16, *Installing and configuring RH-SSO* .
- A user with the **admin** role has been added to RH-SSO as described in Section 17.1, "Adding Red Hat Decision Manager users".

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required) and then select the product and version from the drop-down options:
   - **Product:** Red Hat Single Sign-On
   - **Version:** 7.5
2. Select the **Patches** tab.
3. Download **Red Hat Single Sign-On 7.5 Client Adapter for EAP 7** (**rh-sso-7.5.0-eap7-adapter.zip** or the latest version).
4. Extract and install the adapter zip file. For installation instructions, see the "JBoss EAP Adapter" section of the *Red Hat Single Sign On Securing Applications and Services Guide* .



NOTE

Install the adapter with the **-Dserver.config=standalone-full.xml** property.

5. Navigate to the ***EAP_HOME*/standalone/configuration** directory in your Red Hat JBoss EAP installation and open the **standalone-full.xml** file in a text editor.
6. Add the system properties listed in the following example to **<system-properties>**:

```
<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. Optional: If you want to use client roles, add the following system property:

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

   By default, the client resource name is **kie**. The client resource name must be the same as the client name that you used to configure the client in RH-SSO. If you want to use a custom client resource name, add the following system property:

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

Replace **customClient** with the client resource name.

8. Add the RH-SSO subsystem configuration. For example:

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
 <secure-deployment name="business-central.war">
  <realm>demo</realm>
  <realm-public-
key>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCrVrCuTtArbgaZzL1hvh0xtL5mc
7o0NqPVnYXkLvgcwiC3BjLGw1tGEGoJaXDuSaRllobm53JBhjx33UNv+5z/UMG4kytBWxheNV
KnL6GgqlNabMaFfPLPCF8kAgKnsi79NMo+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>
```

In this example:

- **secure-deployment name** is the name of your application's WAR file.
- **realm** is the name of the realm that you created for the applications to use.
- **realm-public-key** is the public key of the realm you created. You can find the key in the **Keys** tab in the **Realm settings** page of the realm you created in the RH-SSO Admin Console. If you do not provide a value for **realm-public-key**, the server retrieves it automatically.
- **auth-server-url** is the URL for the RH-SSO authentication server.
- **enable-basic-auth** is the setting to enable basic authentication mechanism, so that the clients can use both token-based and basic authentication approaches to perform the requests.
- **resource** is the name for the client that you created. To use client roles, set the client resource name that you used when configuring the client in RH-SSO.
- **credential name** is the secret key for the client you created. You can find the key in the **Credentials** tab on the **Clients** page of the RH-SSO Admin Console.
- **principal-attribute** is the attribute for displaying the user name in the application. If you do not provide this value, your User Id is displayed in the application instead of your user name.

NOTE

The RH-SSO server converts the user names to lower case. Therefore, after integration with RH-SSO, your user name will appear in lower case in Red Hat Decision Manager. If you have user names in upper case hard coded in business processes, the application might not be able to identify the upper case user.

If you want to use client roles, also add the following setting under **<secure-deployment>**:

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

9. The Elytron subsystem provides a built-in policy provider based on JACC specification. To enable the JACC manually in the **standalone.xml** or in the file where Elytron is installed, do any of the following tasks:

   - To create the policy provider, enter the following commands in the management command-line interface (CLI) of Red Hat JBoss EAP:

     ```
     /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)
     ```

     For more information about the Red Hat JBoss EAP management CLI, see the *Management CLI Guide* for Red Hat JBoss EAP.

   - Navigate to the ***EAP_HOME*/standalone/configuration** directory in your Red Hat JBoss EAP installation. Locate the Elytron and undertow subsystem configurations in the **standalone.xml** and **standalone-full.xml** files and enable JACC. For example:

     ```
     <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. Navigate to ***EAP_HOME*/bin/** and enter the following command to start the Red Hat JBoss EAP server:

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

NOTE

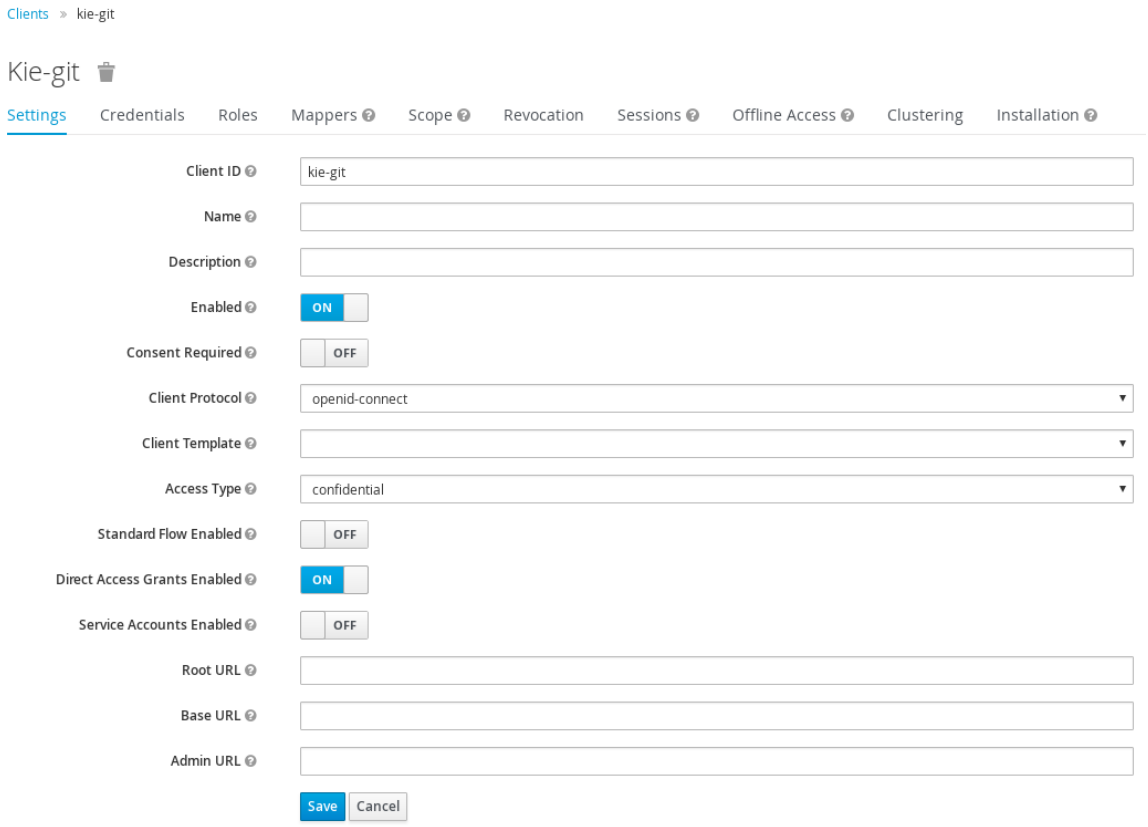
You can also configure the RH-SSO adapter for Business Central by updating your application's WAR file to use the RH-SSO security subsystem. However, Red Hat recommends that you configure the adapter through the RH-SSO subsystem. Doing this updates the Red Hat JBoss EAP configuration instead of applying the configuration on each WAR file.

18.3. ENABLING ACCESS TO EXTERNAL FILE SYSTEMS AND GIT REPOSITORY SERVICES FOR BUSINESS CENTRAL USING RH-SSO

To enable Business Central to consume other remote services, such as file systems and Git repositories, using RH-SSO authentication, you must create a configuration file.

Procedure

1. Generate a JSON configuration file:
   a. Navigate to the **RH-SSO Admin Console** located at http://localhost:8180/auth/admin.
   b. Click **Clients**.
   c. Create a new client with the following settings:
      - Set **Client ID** as **kie-git**.
      - Set **Access Type** as **confidential**.
      - Disable the **Standard Flow Enabled** option.
      - Enable the **Direct Access Grants Enabled** option.



d. Click **Save**.

e. Click the **Installation** tab at the top of the client configuration screen and choose **Keycloak OIDC JSON** as a **Format Option**.

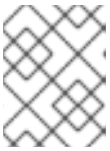
f. Click **Download**.

2. Move the downloaded JSON file to an accessible directory in the server's file system or add it to the application class path. The default name and location for this file is **$EAP_HOME/kie-git.json**.

3. Optional: In the ***EAP_HOME*/standalone/configuration/standalone-full.xml** file, under the **<system-properties>** tag, add the following system property:

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

Replace the **$*EAP_HOME*/kie-git.json** value of the property with the absolute path or the class path (**classpath:/*EXAMPLE_PATH*/kie-git.json**) to the new JSON configuration file.



NOTE

If you do not set the **org.uberfire.ext.security.keycloak.keycloak-config-file** property, Red Hat Decision Manager reads the **$EAP_HOME/kie-git.json** file.

Result

All users authenticated through the RH-SSO server can clone internal GIT repositories. In the following command, replace **USER_NAME** with a RH-SSO user, for example **admin**:

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

+

NOTE

The RH-SSO server client uses one URL to a single remote service deployment. The following error message might be displayed if there are two or more deployment configurations:

We are sorry... **Invalid parameter: redirect_uri**

To resolve this error, append **/*** to the **Valid Redirect URIs** field in the client configuration.

On the **Configure** page, go to **Clients** > **kie-git** > **Settings**, and append the **Valid Redirect URIs** field with **/***, for example:

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

CHAPTER 19. AUTHENTICATING KIE SERVER THROUGH RH-SSO

KIE Server provides a REST API for third-party clients. If you integrate KIE Server with RH-SSO, you can delegate third-party client identity management to the RH-SSO server.

After you create a realm client for Red Hat Decision Manager and set up the RH-SSO client adapter for Red Hat JBoss EAP, you can set up RH-SSO authentication for KIE Server.

Prerequisites

- RH-SSO is installed as described in Chapter 16, *Installing and configuring RH-SSO* .
- At least one user with the **kie-server** role has been added to RH-SSO as described in Section 17.1, "Adding Red Hat Decision Manager users" .
- KIE Server is installed in a Red Hat JBoss EAP 7.4 instance, as described in *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.4*.

This chapter contains the following sections:

- Section 19.1, "Creating the KIE Server client on RH-SSO"
- Section 19.2, "Installing and configuring KIE Server with the client adapter"
- Section 19.3, "KIE Server token-based authentication"

NOTE

Except for Section 19.1, "Creating the KIE Server client on RH-SSO" , this section is intended for standalone installations. If you are integrating RH-SSO and Red Hat Decision Manager on Red Hat OpenShift Container Platform, complete the steps in Section 19.1, "Creating the KIE Server client on RH-SSO" and then deploy the Red Hat Decision Manager environment on Red Hat OpenShift Container Platform. For information about deploying Red Hat Decision Manager on Red Hat OpenShift Container Platform, see Deploying Red Hat Decision Manager on Red Hat OpenShift Container Platform.

19.1. CREATING THE KIE SERVER CLIENT ON RH-SSO

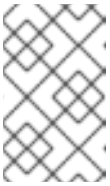
Use the RH-SSO Admin Console to create a KIE Server client in an existing realm.

Prerequisites

- KIE Server is installed in a Red Hat JBoss EAP 7.4 server, as described in *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.4*.
- RH-SSO is installed as described in Chapter 16, *Installing and configuring RH-SSO* .
- At least one user with the **kie-server** role has been added to RH-SSO as described in Section 17.1, "Adding Red Hat Decision Manager users" .

Procedure

1. In the RH-SSO Admin Console, open the security realm that you created in Chapter 16, *Installing and configuring RH-SSO* .
2. Click **Clients** and click **Create**.
   The **Add Client** page opens.
3. On the **Add Client** page, provide the required information to create a KIE Server client for your realm, then click **Save**. For example:
   - Client ID: **kie-execution-server**
   - Root URL: **http://*localhost*:8080/kie-server**
   - Client protocol: **openid-connect**



NOTE

If you are configuring RH-SSO with Red Hat OpenShift Container Platform, enter the URL that is exposed by the KIE Server routes. Your OpenShift administrator can provide this URL if necessary.

4. The new client **Access Type** is set to **public** by default. Change it to **confidential** and click **Save** again.
5. Navigate to the **Credentials** tab and copy the secret key. The secret key is required to configure the **kie-execution-server** client.

NOTE

The RH-SSO server client uses one URL to a single KIE Server deployment. The following error message might be displayed if there are two or more deployment configurations:

We are sorry... **Invalid parameter: redirect_uri**

To resolve this error, append **/*** to the **Valid Redirect URIs** field in the client configuration.

On the **Configure** page, go to **Clients** > **kie-execution-server** > **Settings**, and append the **Valid Redirect URIs** field with **/***, for example:

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

19.2. INSTALLING AND CONFIGURING KIE SERVER WITH THE CLIENT ADAPTER

After you install RH-SSO, you must install the RH-SSO client adapter for Red Hat JBoss EAP and configure it for KIE Server.

Prerequisites

- KIE Server is installed in a Red Hat JBoss EAP 7.4 server, as described in *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.4*.

- RH-SSO is installed as described in Chapter 16, *Installing and configuring RH-SSO* .
- At least one user with the **kie-server** role has been added to RH-SSO as described in Section 17.1, “Adding Red Hat Decision Manager users” .

NOTE

If you deployed KIE Server to a different application server than Business Central, install and configure RH-SSO on your second server as well.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required) and then select the product and version from the drop-down options:
   - **Product:** Red Hat Single Sign-On
   - **Version:** 7.5
2. Download **Red Hat Single Sign-On 7.5 Client Adapter for JBoss EAP 7 (rh-sso-7.5.0-eap7-adapter.zip** or the latest version).
3. Extract and install the adapter zip file. For installation instructions, see the "JBoss EAP Adapter" section of the *Red Hat Single Sign On Securing Applications and Services Guide* .
4. Go to ***EAP_HOME*/standalone/configuration** and open the **standalone-full.xml** file.
5. Delete the **<single-sign-on/>** element from both of the files.
6. Navigate to ***EAP_HOME*/standalone/configuration** directory in your Red Hat JBoss EAP installation and edit the **standalone-full.xml** file to add the RH-SSO subsystem configuration. For example:
7. Navigate to ***EAP_HOME*/standalone/configuration** in your Red Hat JBoss EAP installation and edit the **standalone-full.xml** file to add the RH-SSO subsystem configuration. For example:

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
  <secure-deployment name="kie-server.war">
    <realm>demo</realm>
    <realm-public-
key>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCrVrCuTtArbgaZzL1hvh0xtL5mc
7o0NqPVnYXkLvgcwiC3BjLGw1tGEGoJaXDuSaRllobm53JBhjx33UNv+5z/UMG4kytBWxheNV
KnL6GgqlNabMaFfPLPCF8kAgKnsi79NMo+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>
```

In this example:

- **secure-deployment name** is the name of your application WAR file.
- **realm** is the name of the realm that you created for the applications to use.
- **realm-public-key** is the public key of the realm you created. You can find the key in the **Keys** tab in the **Realm settings** page of the realm you created in the RH-SSO Admin Console. If you do not provide a value for this public key, the server retrieves it automatically.
- **auth-server-url** is the URL for the RH-SSO authentication server.
- **resource** is the name for the server client that you created.
- **enable-basic-auth** is the setting to enable basic authentication mechanism, so that the clients can use both token-based and basic authentication approaches to perform the requests.
- **credential name** is the secret key of the server client you created. You can find the key in the **Credentials** tab on the **Clients** page of the RH-SSO Admin Console.
- **principal-attribute** is the attribute for displaying the user name in the application. If you do not provide this value, your User Id is displayed in the application instead of your user name.

8. Save your configuration changes.
9. Use the following command to restart the Red Hat JBoss EAP server and run KIE Server.

```
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=
<CONTOLLER_PASSWORD>
```

For example:

```
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/business-
central/rest/controller -Dorg.kie.server.controller.user=kiecontroller -
Dorg.kie.server.controller.pwd=password
```

10. When KIE Server is running, enter the following command to check the server status, where **<KIE_SERVER_USER>** is a user with the **kie-server** role and **<PASSWORD>** is the password for that user:

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

19.3. KIE SERVER TOKEN-BASED AUTHENTICATION

You can also use token-based authentication for communication between Red Hat Decision Manager and KIE Server. You can use the complete token as a system property of your application server, instead

of the user name and password, for your applications. However, you must ensure that the token does not expire while the applications are interacting because the token is not automatically refreshed. To get the token, see Section 20.2, "Token-based authentication".

Procedure

1. To configure Business Central to manage KIE Server using tokens:
   a. Set the **org.kie.server.token** property.
   b. Make sure that the **org.kie.server.user** and **org.kie.server.pwd** properties are not set. Red Hat Decision Manager will then use the **Authorization: Bearer $TOKEN** authentication method.
2. To use the REST API using the token-based authentication:
   a. Set the **org.kie.server.controller.token** property.
   b. Make sure that the **org.kie.server.controller.user** and **org.kie.server.controller.pwd** properties are not set.

NOTE

Because KIE Server is unable to refresh the token, use a high-lifespan token. A token's lifespan must not exceed January 19, 2038. Check with your security best practices to see whether this is a suitable solution for your environment.

CHAPTER 20. AUTHENTICATING THIRD-PARTY CLIENTS THROUGH RH-SSO

To use the different remote services provided by Business Central or by KIE Server, your client, such as curl, wget, web browser, or a custom REST client, must authenticate through the RH-SSO server and have a valid token to perform the requests. To use the remote services, the authenticated user must have the following roles:

- **rest-all** for using Business Central remote services.
- **kie-server** for using the KIE Server remote services.

Use the RH-SSO Admin Console to create these roles and assign them to the users that will consume the remote services.

Your client can authenticate through RH-SSO using one of these options:

- Basic authentication, if it is supported by the client
- Token-based authentication

20.1. BASIC AUTHENTICATION

If you enabled basic authentication in the RH-SSO client adapter configuration for both Business Central and KIE Server, you can avoid the token grant and refresh calls and call the services as shown in the following examples:

- For web based remote repositories endpoint:

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

- For KIE Server:

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

20.2. TOKEN-BASED AUTHENTICATION

If you want a more secure option of authentication, you can consume the remote services from both Business Central and KIE Server by using a granted token provided by RH-SSO.

Procedure

1. In the RH-SSO Admin Console, click the **Clients** menu item and click **Create** to create a new client.
   The **Add Client** page opens.
2. On the **Add Client** page, provide the required information to create a new client for your realm. For example:
   - **Client ID**: **kie-remote**
   - **Client protocol**: **openid-connect**
3. Click **Save** to save your changes.

4. Change the token settings in **Realm Settings**:
   a. In the RH-SSO Admin Console, click the **Realm Settings** menu item.
   b. Click the **Tokens** tab.
   c. Change the value for **Access Token Lifespan** to **15** minutes.
   This gives you enough time to get a token and invoke the service before it expires.
   d. Click **Save** to save your changes.
5. After a public client for your remote clients is created, you can now obtain the token by making an HTTP request to the RH-SSO server's token endpoint using:

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

   The user in this command is a Business Central RH-SSO user. For more information, see Section 17.1, "Adding Red Hat Decision Manager users" .

6. To view the token obtained from the RH-SSO server, use the following command:

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

You can now use this token to authorize the remote calls. For example, if you want to check the internal Red Hat Decision Manager repositories, use the token as shown below:

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

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Thursday, March 14th, 2024.

APPENDIX B. CONTACT INFORMATION

Red Hat Decision Manager documentation team: brms-docs@redhat.com