Red Hat Decision Manager 7.8

Managing and monitoring KIE Server

Last Updated: 2020-08-12

Red Hat Decision Manager 7.8 Managing and monitoring KIE Server

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

Legal Notice

Abstract

This document explains how install, configure, and performance tune Red Hat Decision Manager 7.8

Table of Contents

PREFACE

As a systems administrator, you can install, configure, and upgrade Red Hat Decision Manager for production environments, quickly and easily troubleshoot system failures, and ensure that systems are running optimally.

Prerequisites

- Red Hat JBoss Enterprise Application Platform 7.3 is installed. For more information, see *Red Hat JBoss Enterprise Application Platform 7.3 Installation Guide*.
- Red Hat Decision Manager is installed. For more information, see *Planning a Red Hat Decision Manager installation*.
- Red Hat Decision Manager is running and you can log in to Business Central with the **admin** role. For more information, see *Planning a Red Hat Decision Manager installation* .

CHAPTER 1. RED HAT DECISION MANAGER COMPONENTS

Red Hat Decision Manager is made up of Business Central and KIE Server.

- Business Central is the graphical user interface where you create and manage business rules. You can install Business Central in a Red Hat JBoss EAP instance or on the Red Hat OpenShift Container Platform (OpenShift).
  Business Central is also available as a standalone JAR file. You can use the Business Central standalone JAR file to run Business Central without needing to deploy it to an application server.

- KIE Server is the server where rules and other artifacts are executed. It is used to instantiate and execute rules and solve planning problems. You can install KIE Server in a Red Hat JBoss EAP instance, on OpenShift, in an Oracle WebLogic server instance, in an IBM WebSphere Application Server instance, or as a part of Spring Boot application.
  You can configure KIE Server to run in managed or unmanaged mode. If KIE Server is unmanaged, you must manually create and maintain KIE containers (deployment units). A KIE container is a specific version of a project. If KIE Server is managed, the Decision Manager controller manages the KIE Server configuration and you interact with the Decision Manager controller to create and maintain KIE containers.

CHAPTER 2. SYSTEM INTEGRATION WITH MAVEN

Red Hat Decision Manager is designed to be used with Red Hat JBoss Middleware Maven Repository and Maven Central repository as dependency sources. Ensure that both the dependencies are available for projects builds.

Ensure that your project depends on specific versions of an artifact. **LATEST** or **RELEASE** are commonly used to specify and manage dependency versions in your application.

- **LATEST** refers to the latest deployed (snapshot) version of an artifact.
- **RELEASE** refers to the last non-snapshot version release in the repository.

By using **LATEST** or **RELEASE**, you do not have to update version numbers when a new release of a third-party library is released, however, you lose control over your build being affected by a software release.

2.1. PREEMPTIVE AUTHENTICATION FOR LOCAL PROJECTS

If your environment does not have access to the internet, set up an in-house Nexus and use it instead of Maven Central or other public repositories. To import JARs from the remote Maven repository of Red Hat Decision Manager server to a local Maven project, turn on pre-emptive authentication for the repository server. You can do this by configuring authentication for **guvnor-m2-repo** in the **pom.xml** file as shown below:

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

Alternatively, you can set Authorization HTTP header with Base64 encoded credentials:

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

2.2. DUPLICATE GAV DETECTION IN BUSINESS CENTRAL

In Business Central, all Maven repositories are checked for any duplicated **GroupId**, **ArtifactId**, and **Version** (GAV) values in a project. If a GAV duplicate exists, the performed operation is canceled.

NOTE

Duplicate GAV detection is disabled for projects in **Development Mode**. To enable duplicate GAV detection in Business Central, go to project **Settings → General Settings → Version** and toggle the **Development Mode** option to **OFF** (if applicable).

Duplicate GAV detection is executed every time you perform the following operations:

- Save a project definition for the project.
- Save the **pom.xml** file.
- Install, build, or deploy a project.

The following Maven repositories are checked for duplicate GAVs:

- Repositories specified in the **<repositories>** and **<distributionManagement>** elements of the **pom.xml** file.
- Repositories specified in the Maven **settings.xml** configuration file.

2.3. MANAGING DUPLICATE GAV DETECTION SETTINGS IN BUSINESS CENTRAL

Business Central users with the **admin** role can modify the list of repositories that are checked for duplicate **GroupId**, **ArtifactId**, and **Version** (GAV) values for a project.

NOTE

Duplicate GAV detection is disabled for projects in **Development Mode**. To enable duplicate GAV detection in Business Central, go to project **Settings → General Settings → Version** and toggle the **Development Mode** option to **OFF** (if applicable).

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.
2. Click the project **Settings** tab and then click **Validation** to open the list of repositories.
3. Select or clear any of the listed repository options to enable or disable duplicate GAV detection. In the future, duplicate GAVs will be reported for only the repositories you have enabled for validation.

NOTE

To disable this feature, set the **org.guvnor.project.gav.check.disabled** system property to **true** for Business Central at system startup:

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

CHAPTER 3. APPLYING PATCH UPDATES AND MINOR RELEASE UPGRADES TO RED HAT DECISION MANAGER

Automated update tools are often provided with both patch updates and new minor versions of Red Hat Decision Manager to facilitate updating certain components of Red Hat Decision Manager, such as Business Central, KIE Server, and the headless Decision Manager controller. Other Red Hat Decision Manager artifacts, such as the decision engine and standalone Business Central, are released as new artifacts with each minor release and you must re-install them to apply the update.

You can use the same automated update tool to apply both patch updates and minor release upgrades to Red Hat Decision Manager 7.8. Patch updates of Red Hat Decision Manager, such as an update from version 7.8 to 7.8.1, include the latest security updates and bug fixes. Minor release upgrades of Red Hat Decision Manager, such as an upgrade from version 7.7.x to 7.8, include enhancements, security updates, and bug fixes.

NOTE

Only updates for Red Hat Decision Manager are included in Red Hat Decision Manager update tools. Updates to Red Hat JBoss EAP must be applied using Red Hat JBoss EAP patch distributions. For more information about Red Hat JBoss EAP patching, see the *Red Hat JBoss EAP patching and upgrading guide* .

Prerequisites

- Your Red Hat Decision Manager and KIE Server instances are not running. Do not apply updates while you are running an instance of Red Hat Decision Manager or KIE Server.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options.
   If you are upgrading to a new minor release of Red Hat Decision Manager, such as an upgrade from version 7.7.x to 7.8, first apply the latest patch update to your current version of Red Hat Decision Manager and then follow this procedure again to upgrade to the new minor release.

2. Click **Patches**, download the **Red Hat Decision Manager [VERSION] Update Tool** and extract the downloaded **rhdm-$VERSION-update.zip** file to a temporary directory.
   This update tool automates the update of certain components of Red Hat Decision Manager, such as Business Central, KIE Server, and the headless Decision Manager controller. Use this update tool first to apply updates and then install any other updates or new release artifacts that are relevant to your Red Hat Decision Manager distribution.

3. If you want to preserve any files from being updated by the update tool, navigate to the extracted **rhdm-$VERSION-update** folder, open the **blacklist.txt** file, and add the relative paths to the files that you do not want to be updated.
   When a file is listed in the **blacklist.txt** file, the update script does not replace the file with the new version but instead leaves the file in place and in the same location adds the new version with a **.new** suffix. If you blacklist files that are no longer being distributed, the update tool creates an empty marker file with a **.removed** suffix. You can then choose to retain, merge, or delete these new files manually.

   Example files to be excluded in **blacklist.txt** file:

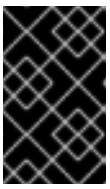
```
WEB-INF/web.xml  // Custom file
styles/base.css  // Obsolete custom file kept for record
```

The contents of the blacklisted file directories after the update:

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

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

4. In your command terminal, navigate to the temporary directory where you extracted the **rhdm-$VERSION-update.zip** file and run the **apply-updates** script in the following format:



IMPORTANT

Make sure that your Red Hat Decision Manager and KIE Server instances are not running before you apply updates. Do not apply updates while you are running an instance of Red Hat Decision Manager or KIE Server.

On Linux or Unix-based systems:

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

On Windows:

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

The **$DISTRO_PATH** portion is the path to the relevant distribution directory and the **$DISTRO_TYPE** portion is the type of distribution that you are updating with this update.

The following distribution types are supported in Red Hat Decision Manager update tool:

- **rhdm-decision-central-eap7-deployable**: Updates Business Central (**decision-central.war**)
- **rhdm-kie-server-ee8**: Updates KIE Server (**kie-server.war**)

NOTE

The update tool will update Red Hat JBoss EAP EE7 to Red Hat JBoss EAP EE8.

- **rhdm-kie-server-jws**: Updates KIE Server on Red Hat JBoss Web Server (**kie-server.war**)
- **rhdm-controller-ee7**: Updates the headless Decision Manager controller (**controller.war**)
- **rhdm-controller-jws**: Updates the headless Decision Manager controller on Red Hat JBoss Web Server (**controller.war**)
  Example update to Business Central and KIE Server for a full Red Hat Decision Manager distribution on Red Hat JBoss EAP:

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

./apply-updates.sh ~EAP_HOME/standalone/deployments/kie-server.war rhdm-kie-server-ee8
```

Example update to headless Decision Manager controller, if used:

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

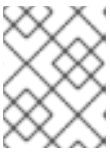
The update script creates a **backup** folder in the extracted **rhdm-$VERSION-update** folder with a copy of the specified distribution, and then proceeds with the update.

5. After the update tool completes, return to the **Software Downloads** page of the Red Hat Customer Portal where you downloaded the update tool and install any other updates or new release artifacts that are relevant to your Red Hat Decision Manager distribution.
   For files that already exist in your Red Hat Decision Manager distribution, such as **.jar** files for the decision engine or other add-ons, replace the existing version of the file with the new version from the Red Hat Customer Portal.

6. If you use the standalone **Red Hat Decision Manager 7.8.0 Maven Repository**artifact (**rhdm-7.8.0-maven-repository.zip**), such as in air-gap environments, download **Red Hat Decision Manager 7.8.x Maven Repository** and extract the downloaded **rhdm-7.8.x-maven-repository.zip** file to your existing **~/maven-repository** directory to update the relevant contents.
   Example Maven repository update:

```
$ unzip -o rhdm-7.8.x-maven-repository.zip 'rhba-7.8.1.GA-maven-repository/maven-repository/*' -d /tmp/rhbaMavenRepoUpdate

$ mv /tmp/rhbaMavenRepoUpdate/rhba-7.8.0.GA-maven-repository/maven-repository/ $REPO_PATH/
```



NOTE

You can remove the **/tmp/rhbaMavenRepoUpdate** folder after you complete the update.

7. After you finish applying all relevant updates, start Red Hat Decision Manager and KIE Server and log in to Business Central.

8. Verify that all project data is present and accurate in Business Central, and in the top-right corner of the Business Central window, click your profile name and click **About** to verify the updated product version number.
   If you encounter errors or notice any missing data in Business Central, you can restore the contents in the **backup** folder within the **rhdm-$VERSION-update** folder to revert the update tool changes. You can also re-install the relevant release artifacts from your previous version of Red Hat Decision Manager in the Red Hat Customer Portal. After restoring your previous distribution, you can try again to run the update.

CHAPTER 4. CONFIGURING AND STARTING KIE SERVER

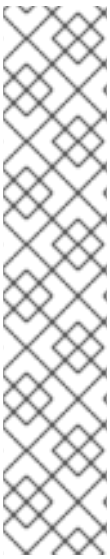
You can configure your KIE Server location, user name, password, and other related properties by defining the necessary configurations when you start KIE Server.

Procedure

Navigate to the Red Hat Decision Manager 7.8 **bin** directory and start the new KIE Server with the following properties. Adjust the specific properties according to your environment.

```
$ ~/EAP_HOME/bin/standalone.sh --server-config=standalone-full.xml (1)
-Dorg.kie.server.id=myserver (2)
-Dorg.kie.server.user=kie_server_username (3)
-Dorg.kie.server.pwd=kie_server_password (4)
-Dorg.kie.server.controller=http://localhost:8080/decision-central/rest/controller (5)
-Dorg.kie.server.controller.user=controller_username (6)
-Dorg.kie.server.controller.pwd=controller_password (7)
-Dorg.kie.server.location=http://localhost:8080/kie-server/services/rest/server (8)
```

1. Start command with **standalone-full.xml** server profile
2. Server ID that must match the server configuration name defined in Business Central
3. User name to connect with KIE Server from the Decision Manager controller
4. Password to connect with KIE Server from the Decision Manager controller
5. Decision Manager controller location, Business Central URL with **/rest/controller** suffix
6. User name to connect to the Decision Manager controller REST API
7. Password to connect to the Decision Manager controller REST API
8. KIE Server location (on the same instance as Business Central in this example)



NOTE

If Business Central and KIE Server are installed on separate application server instances (Red Hat JBoss EAP or other), use a separate port for the KIE Server location to avoid port conflicts with Business Central. If a separate KIE Server port has not already been configured, you can add a port offset and adjust the KIE Server port value accordingly in the KIE Server properties.

Example:

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

If the Business Central port is 8080, as in this example, then the KIE Server port, with a defined offset of 150, is 8230.

KIE Server connects to the new Business Central and collects the list of deployment units (KIE containers) to be deployed.

NOTE

When you use a class inside a dependency JAR file to access KIE Server from KIE Server client, you get the **ConversionException** and **ForbiddenClassException** in Business Central. To avoid generating these exceptions in Business Central, do one of the following:

- If the exceptions are generated on the client-side, add following system property to the kie-server client:

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

- If the exceptions are generated on the server-side, open **standalone-full.xml** from the Red Hat Decision Manager installation directory, set the following property under the <system-properties> tag:

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

- Set the following JVM property:

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

It is expected that you do not configure the classes that exists in KJAR using these system property. Ensure that only known classes are used in the system property to avoid any vulnerabilities.

The **org.example** is an example package, you can define any package that you want to use. You can specify multiple packages separated by comma , for example, **org.example1.* * , org.example2.* * , org.example3.* *.**

You can also add specific classes , for example, **org.example1.Mydata1, org.example2.Mydata2**.

CHAPTER 5. CONFIGURING KIE SERVER WITH THE INTEGRATED DECISION MANAGER CONTROLLER

NOTE

Only make the changes described in this section if KIE Server will be managed by Business Central and you installed Red Hat Decision Manager from the ZIP files. If you did not install Business Central, you can use the headless Decision Manager controller to manage KIE Server, as described in Chapter 6, *Installing and running the headless Decision Manager controller*.

KIE Server can be managed or it can be unmanaged. If KIE Server is unmanaged, you must manually create and maintain KIE containers (deployment units). If KIE Server is managed, the Decision Manager controller manages the KIE Server configuration and you interact with the Decision Manager controller to create and maintain KIE containers.

The Decision Manager controller is integrated with Business Central. If you install Business Central, you can use the **Execution Server** page in Business Central to interact with the Decision Manager controller.

If you installed Red Hat Decision Manager from the ZIP files, you must edit the **standalone-full.xml** file in both the KIE Server and Business Central installations to configure KIE Server with the integrated Decision Manager controller.

Prerequisites

- Business Central and KIE Server are installed in the base directory of the Red Hat JBoss EAP installation (***EAP_HOME***).

NOTE

You should install Business Central and KIE Server on different servers in production environments. However, if you install KIE Server and Business Central on the same server, for example in a development environment, make the changes described in this section in the shared **standalone-full.xml** file.

- On Business Central server nodes, a user with the **rest-all** role exists.

Procedure

1. In the Business Central ***EAP_HOME*/standalone/configuration/standalone-full.xml** file, uncomment the following properties in the **<system-properties>** section and replace **<USERNAME>** and **<USER_PWD>** with the credentials of a user with the **kie-server** role:

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

2. In the KIE Server ***EAP_HOME*/standalone/configuration/standalone-full.xml** file, uncomment the following properties in the **<system-properties>** section.

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

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

3. Replace the following values:

- Replace **<CONTROLLER_USER>** and **<CONTROLLER_PWD>** with the credentials of a user with the **rest-all** role.
- Replace **<KIE_SERVER_ID>** with the ID or name of the KIE Server installation, for example, **rhdm-7.8.0-kie-server-1**.
- Replace **<HOST>** with the ID or name of the KIE Server host, for example, **localhost** or **192.7.8.9**.
- Replace **<PORT>** with the port of the KIE Server host, for example, **8080**.

NOTE

The **org.kie.server.location** property specifies the location of KIE Server.

- Replace **<CONTROLLER_URL>** with the URL of Business Central. KIE Server connects to this URL during startup.
  - If you installed Business Central using the installer or Red Hat JBoss EAP zip installations, **<CONTROLLER_URL>** has this format:
    http://<HOST>:<PORT>/decision-central/rest/controller
  - If you are running Business Central using the **standalone.jar** file, **<CONTROLLER_URL>** has this format:
    http://<HOST>:<PORT>/rest/controller

CHAPTER 6. INSTALLING AND RUNNING THE HEADLESS DECISION MANAGER CONTROLLER

You can configure KIE Server to run in managed or unmanaged mode. If KIE Server is unmanaged, you must manually create and maintain KIE containers (deployment units). If KIE Server is managed, the Decision Manager controller manages the KIE Server configuration and you interact with the Decision Manager controller to create and maintain KIE containers.

Business Central has an embedded Decision Manager controller. If you install Business Central, use the **Execution Server** page to create and maintain KIE containers. If you want to automate KIE Server management without Business Central, you can use the headless Decision Manager controller.

6.1. USING THE INSTALLER TO CONFIGURE KIE SERVER WITH THE DECISION MANAGER CONTROLLER

KIE Server can be managed by the Decision Manager controller or it can be unmanaged. If KIE Server is unmanaged, you must manually create and maintain KIE containers (deployment units). If KIE Server is managed, the Decision Manager controller manages the KIE Server configuration and you interact with the Decision Manager controller to create and maintain KIE containers.

The Decision Manager controller is integrated with Business Central. If you install Business Central, you can use the **Execution Server** page in Business Central to interact with the Decision Manager controller.

You can use the installer in interactive or CLI mode to install Business Central and KIE Server, and then configure KIE Server with the Decision Manager controller.

NOTE

If you do not install Business Central, see Chapter 6, *Installing and running the headless Decision Manager controller* for information about using the headless Decision Manager controller.

Prerequisites

- Two computers with backed-up Red Hat JBoss EAP 7.3 server installations are available.
- Sufficient user permissions to complete the installation are granted.

Procedure

1. On the first computer, run the installer in interactive mode or CLI mode. See *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.3* for more information.
2. On the **Component Selection** page, clear the **KIE Server** box.
3. Complete the Business Central installation.
4. On the second computer, run the installer in interactive mode or CLI mode.
5. On the **Component Selection** page, clear the **Business Central** box.
6. On the **Configure Runtime Environment** page, select **Perform Advanced Configuration**.

7. Select **Customize KIE Server properties** and click **Next**.
8. Enter the controller URL for Business Central and configure additional properties for KIE Server. The controller URL has the following form where **<HOST:PORT>** is the address of Business Central on the second computer:

```
<HOST:PORT>/business-central/rest/controller
```

9. Complete the installation.
10. To verify that the Decision Manager controller is now integrated with Business Central, go to the **Execution Servers** page in Business Central and confirm that the KIE Server that you configured appears under **REMOTE SERVERS**.

6.2. INSTALLING THE HEADLESS DECISION MANAGER CONTROLLER

You can install the headless Decision Manager controller and use the REST API or the KIE Server Java Client API to interact with it.

Prerequisites

- A backed-up Red Hat JBoss EAP installation version 7.3 is available. The base directory of the Red Hat JBoss EAP installation is referred to as ***EAP_HOME***.
- Sufficient user permissions to complete the installation are granted.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options:
   - **Product:** Decision Manager
   - **Version:** 7.8
2. Download **Red Hat Decision Manager 7.8.0 Add Ons** (the **rhdm-7.8.0-add-ons.zip** file).
3. Unzip the **rhdm-7.8.0-add-ons.zip** file. The **rhdm-7.8.0-controller-ee7.zip** file is in the unzipped directory.
4. Extract the **rhdm-7.8.0-controller-ee7** archive to a temporary directory. In the following examples this directory is called ***TEMP_DIR***.
5. Copy the ***TEMP_DIR*/rhdm-7.8.0-controller-ee7/controller.war** directory to ***EAP_HOME*/standalone/deployments/**.

WARNING

Ensure that the names of the headless Decision Manager controller deployments you copy do not conflict with your existing deployments in the Red Hat JBoss EAP instance.

6. Copy the contents of the ***TEMP_DIR*/rhdm-7.8.0-controller-ee7/SecurityPolicy/** directory to ***EAP_HOME*/bin**. When asked to overwrite files, select **Yes**.

7. In the ***EAP_HOME*/standalone/deployments/** directory, create an empty file named **controller.war.dodeploy**. This file ensures that the headless Decision Manager controller is automatically deployed when the server starts.

6.2.1. Creating a headless Decision Manager controller user

Before you can use the headless Decision Manager controller, you must create a user that has the **kie-server** role.

Prerequisites

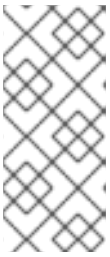
- The headless Decision Manager controller is installed in the base directory of the Red Hat JBoss EAP installation (***EAP_HOME***).

Procedure

1. In a terminal application, navigate to the ***EAP_HOME*/bin** directory.

2. Enter the following command and replace **<USER_NAME>** and **<PASSWORD>** with the user name and password of your choice.

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



NOTE

Make sure that the specified user name is not the same as an existing user, role, or group. For example, do not create a user with the user name **admin**.

The password must have at least eight characters and must contain at least one number and one non-alphanumeric character, but not & (ampersand).

3. Make a note of your user name and password.

6.2.2. Configuring KIE Server and the headless Decision Manager controller

If KIE Server will be managed by the headless Decision Manager controller, you must edit the **standalone-full.xml** file in KIE Server installation and the **standalone.xml** file in the headless Decision Manager controller installation, as described in this section.

Prerequisites

- KIE Server is installed in the base directory of the Red Hat JBoss EAP installation (***EAP_HOME***).

- The headless Decision Manager controller is installed in an ***EAP_HOME***.

NOTE

You should install KIE Server and the headless Decision Manager controller on different servers in production environments. However, if you install KIE Server and the headless Decision Manager controller on the same server, for example in a development environment, make these changes in the shared **standalone-full.xml** file.

- On KIE Server nodes, a user with the **kie-server** role exists.
- On the server nodes, a user with the **kie-server** role exists.

Procedure

1. In the ***EAP_HOME*/standalone/configuration/standalone-full.xml** file, add the following properties to the **<system-properties>** section and replace **<USERNAME>** and **<USER_PWD>** with the credentials of a user with the **kie-server** role:

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

2. In the KIE Server ***EAP_HOME*/standalone/configuration/standalone-full.xml** file, add the following properties to the **<system-properties>** section:

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

3. In this file, replace the following values:

- Replace **<CONTROLLER_USER>** and **<CONTROLLER_PWD>** with the credentials of a user with the **kie-server** role.
- Replace **<KIE_SERVER_ID>** with the ID or name of the KIE Server installation, for example, **rhdm-7.8.0-kie-server-1**.
- Replace **<HOST>** with the ID or name of the KIE Server host, for example, **localhost** or **192.7.8.9**.
- Replace **<PORT>** with the port of the KIE Server host, for example, **8080**.

NOTE

The **org.kie.server.location** property specifies the location of KIE Server.

- Replace **<CONTROLLER_URL>** with the URL of the headless Decision Manager controller.

1. KIE Server connects to this URL during startup.

6.3. RUNNING THE HEADLESS DECISION MANAGER CONTROLLER

After you have installed the headless Decision Manager controller on Red Hat JBoss EAP, use this procedure to run the headless Decision Manager controller.

Prerequisites

- The headless Decision Manager controller is installed and configured in the base directory of the Red Hat JBoss EAP installation (***EAP_HOME***).

Procedure

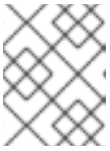
1. In a terminal application, navigate to ***EAP_HOME*/bin**.
2. If you installed the headless Decision Manager controller on the same Red Hat JBoss EAP instance as the Red Hat JBoss EAP instance where you installed the KIE Server, enter one of the following commands:
   - On Linux or UNIX-based systems:

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

   - On Windows:

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

3. If you installed the headless Decision Manager controller on a separate Red Hat JBoss EAP instance from the Red Hat JBoss EAP instance where you installed the KIE Server, you can start the headless Decision Manager controller with the **standalone.sh** script:

   NOTE

   In this case, ensure that you made all required configuration changes to the **standalone.xml** file.

   - On Linux or UNIX-based systems:

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

   - On Windows:

     ```
     standalone.bat
     ```

4. To verify that the headless Decision Manager controller is working on Red Hat JBoss EAP, enter the following command where **<CONTROLLER>** and **<CONTROLLER_PWD>** is the user name and password. The output of this command provides information about the KIE Server instance.

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

NOTE

Alternatively, you can use the KIE Server Java API Client to access the headless Decision Manager controller.

6.4. CLUSTERING KIE SERVERS WITH THE HEADLESS DECISION MANAGER CONTROLLER

The Decision Manager controller is integrated with Business Central. However, if you do not install Business Central, you can install the headless Decision Manager controller and use the REST API or the KIE Server Java Client API to interact with it.

Prerequisites

- A backed-up Red Hat JBoss EAP installation version 7.3 or later is available. The base directory of the Red Hat JBoss EAP installation is referred to as ***EAP_HOME***.
- Sufficient user permissions to complete the installation are granted.
- An NFS server with a mounted partition is available as described in *Installing and configuring Red Hat Decision Manager in a Red Hat JBoss EAP clustered environment*.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options:
   - **Product: Decision Manager**
   - **Version: 7.8**
2. Download **Red Hat Decision Manager 7.8.0 Add Ons**(the **rhdm-7.8.0-add-ons.zip** file).
3. Unzip the **rhdm-7.8.0-add-ons.zip** file. The **rhdm-7.8.0-controller-ee7.zip** file is in the unzipped directory.
4. Extract the **rhdm-7.8.0-controller-ee7** archive to a temporary directory. In the following examples this directory is called ***TEMP_DIR***.
5. Copy the ***TEMP_DIR*/rhdm-7.8.0-controller-ee7/controller.war** directory to ***EAP_HOME*/standalone/deployments/**.

WARNING

Ensure that the names of the headless Decision Manager controller deployments you copy do not conflict with your existing deployments in the Red Hat JBoss EAP instance.

6. Copy the contents of the ***TEMP_DIR*/rhdm-7.8.0-controller-ee7/SecurityPolicy/** directory to ***EAP_HOME*/bin**. When asked to overwrite files, select **Yes**.
7. In the ***EAP_HOME*/standalone/deployments/** directory, create an empty file named **controller.war.dodeploy**. This file ensures that the headless Decision Manager controller is automatically deployed when the server starts.
8. Open the ***EAP_HOME*/standalone/configuration/standalone.xml** file in a text editor.

9. Add the following properties to the **<system-properties>** element and replace **<NFS_STORAGE>** with the absolute path to the NFS storage where the template configuration is stored:

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

   Template files contain default configurations for specific deployment scenarios.

   If the value of the **org.kie.server.controller.templatefile.watcher.enabled** property is set to true, a separate thread is started to watch for modifications of the template file. The default interval for these checks is 30000 milliseconds and can be further controlled by the **org.kie.server.controller.templatefile.watcher.interval** system property. If the value of this property is set to false, changes to the template file are detected only when the server restarts.

10. To start the headless Decision Manager controller, navigate to ***EAP_HOME*/bin** and enter the following command:

    - On Linux or UNIX-based systems:

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

    - On Windows:

```
standalone.bat
```

For more information about running Red Hat Decision Manager in a Red Hat JBoss Enterprise Application Platform clustered environment, see *Installing and configuring Red Hat Decision Manager in a Red Hat JBoss EAP clustered environment*.

CHAPTER 7. CONFIGURING A KIE SERVER TO CONNECT TO BUSINESS CENTRAL

If a KIE Server is not already configured in your Red Hat Decision Manager environment, or if you require additional KIE Servers in your Red Hat Decision Manager environment, you must configure a KIE Server to connect to Business Central.

NOTE

If you are deploying KIE Server on Red Hat OpenShift Container Platform, see *Deploying a Red Hat Decision Manager authoring or managed server environment on Red Hat OpenShift Container Platform* for instructions about configuring it to connect to Business Central.

Prerequisites

- KIE Server is installed. For installation options, see *Planning a Red Hat Decision Manager installation*.

Procedure

1. In your Red Hat Decision Manager installation directory, navigate to the **standalone-full.xml** file. For example, if you use a Red Hat JBoss EAP installation for Red Hat Decision Manager, go to **$EAP_HOME/standalone/configuration/standalone-full.xml**.

2. Open **standalone-full.xml** and under the **<system-properties>** tag, set the following properties:

   - **org.kie.server.controller.user:** The user name of a user who can log in to the Business Central.

   - **org.kie.server.controller.pwd:** The password of the user who can log in to the Business Central.

   - **org.kie.server.controller:** The URL for connecting to the API of Business Central. Normally, the URL is **http://<centralhost>:<centralport>/decision-central/rest/controller**, where **<centralhost>** and **<centralport>** are the host name and port for Business Central. If Business Central is deployed on OpenShift, remove **decision-central/** from the URL.

   - **org.kie.server.location:** The URL for connecting to the API of KIE Server. Normally, the URL is **http://<serverhost>:<serverport>/kie-server/services/rest/server**, where **<serverhost>** and **<serverport>** are the host name and port for KIE Server.

   - **org.kie.server.id:** The name of a server configuration. If this server configuration does not exist in Business Central, it is created automatically when KIE Server connects to Business Central.

   Example:

```
<property name="org.kie.server.controller.user" value="central_user"/>
<property name="org.kie.server.controller.pwd" value="central_password"/>
<property name="org.kie.server.controller" value="http://central.example.com:8080/decision-central/rest/controller"/>
```

```
<property name="org.kie.server.location" value="http://kieserver.example.com:8080/kie-server/services/rest/server"/>
<property name="org.kie.server.id" value="production-servers"/>
```

3. Start or restart the KIE Server.

CHAPTER 8. CONFIGURING THE ENVIRONMENT MODE IN KIE SERVER AND BUSINESS CENTRAL

You can set KIE Server to run in **production** mode or in **development** mode. Development mode provides a flexible deployment policy that enables you to update existing deployment units (KIE containers) while maintaining active process instances for small changes. It also enables you to reset the deployment unit state before updating active process instances for larger changes. Production mode is optimal for production environments, where each deployment creates a new deployment unit.

In a development environment, you can click **Deploy** in Business Central 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 in Business Central is disabled and you can click only **Deploy** to deploy the built KJAR file to a new deployment unit (KIE container) on a KIE Server.

Procedure

1. 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**.

2. To configure the deployment behavior for a project in Business Central, go to project **Settings → General Settings → Version** and toggle the **Development Mode** option.

NOTE

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.

CHAPTER 9. CONFIGURING KIE SERVER MANAGED BY BUSINESS CENTRAL

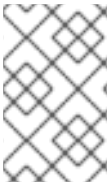
WARNING

This section provides a sample setup that you can use for testing purposes. Some of the values are unsuitable for a production environment, and are marked as such.

Use this procedure to configure Business Central to manage a KIE Server instance.

Prerequisites

- Users with the following roles exist:
  - In Business Central, a user with the role **rest-all**
  - On the KIE Server, a user with the role **kie-server**



NOTE

In production environments, use two distinct users, each with one role. In this sample situation, we use only one user named **controllerUser** that has both the **rest-all** and the **kie-server** roles.

Procedure

1. Set the following JVM properties.
   The location of Business Central and the KIE Server may be different. In such case, ensure you set the properties on the correct server instances.
   - On Red Hat JBoss EAP, modify the **<system-properties>** section in:
     - ***EAP_HOME*/standalone/configuration/standalone*.xml** for standalone mode.
     - ***EAP_HOME*/domain/configuration/domain.xml** for domain mode.

   Table 9.1. JVM Properties for Managed KIE Server Instance

Property	Value	Note
org.kie.server.id	**default-kie-server**	The KIE Server ID.
org.kie.server.controller	**http://localhost:8080/decision-central/rest/controller**	The location of Business Central.
org.kie.server.controller.user	**controllerUser**	The user name with the role **rest-all** as mentioned in the previous step.

Property	Value	Note
org.kie.server.controller.pwd	**controllerUser1234;**	The password of the user mentioned in the previous step.
org.kie.server.location	**http://localhost:8080/kie-server/services/rest/server**	The location of the KIE Server.

Table 9.2. JVM Properties for Business Central Instance

Property	Value	Note
org.kie.server.user	**controllerUser**	The user name with the role **kie-server** as mentioned in the previous step.
org.kie.server.pwd	**controllerUser1234;**	The password of the user mentioned in the previous step.

2. Verify the successful start of the KIE Server by sending a GET request to **http://*SERVER:PORT*/kie-server/services/rest/server/**. Once authenticated, you get an XML response similar to this:

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

3. Verify successful registration:

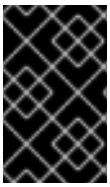
   a. Log in to Business Central.

b. Click **Menu → Deploy → Execution Servers**.
If registration is successful, you can see the registered server ID.

CHAPTER 10. MANAGED KIE SERVER

A managed instance requires an available Decision Manager controller to start the KIE Server.

A Decision Manager controller manages the KIE Server configuration in a centralized way. Each Decision Manager controller can manage multiple configurations at once, and there can be multiple Decision Manager controllers in the environment. Managed KIE Server can be configured with a list of Decision Manager controllers, but will only connect to one at a time.



IMPORTANT

All Decision Manager controllers should be synchronized to ensure that the same set of configuration is provided to the server, regardless of the Decision Manager controller to which it connects.

When the KIE Server is configured with a list of Decision Manager controllers, it will attempt to connect to each of them at startup until a connection is successfully established with one of them. If a connection cannot be established, the server will not start, even if there is a local storage available with configuration. This ensures consistency and prevents the server from running with redundant configuration.

NOTE

To run the KIE Server in standalone mode without connecting to Decision Manager controllers, see Chapter 11, *Unmanaged KIE Server*.

CHAPTER 11. UNMANAGED KIE SERVER

An unmanaged KIE Server is a standalone instance, and therefore must be configured individually using REST/JMS API from the KIE Server itself. The configuration is automatically persisted by the server into a file and that is used as the internal server state, in case of restarts.

The configuration is updated during the following operations:

- Deploy KIE container
- Undeploy KIE container
- Start KIE container
- Stop KIE container

NOTE

If the KIE Server is restarted, it will attempt to re-establish the same state that was persisted before shutdown. Therefore, KIE containers (deployment units) that were running will be started, but the ones that were stopped will not.

CHAPTER 12. DEPLOYMENT DESCRIPTORS

Processes and rules are stored in Apache Maven based packaging and are known as knowledge archives, or KJAR. The rules, processes, assets, and other project artifacts are part of a JAR file built and managed by Maven. A file kept inside the **META-INF** directory of the KJAR called **kmodule.xml** can be used to define the KIE bases and sessions. This **kmodule.xml** file, by default, is empty.

Whenever a runtime component such as KIE Server is about to process the KJAR, it looks up **kmodule.xml** to build the runtime representation.

Deployment descriptors supplement the **kmodule.xml** file and provide granular control over your deployment. The presence of these descriptors is optional and your deployment will proceed successfully without them. You can set purely technical properties using these descriptors, including meta values such as persistence, auditing, and runtime strategy.

These descriptors allow you to configure the KIE Server on multiple levels, including server level default, different deployment descriptor per KJAR, and other server configurations. You can use descriptors to make simple customizations to the default KIE Server configuration, possibly per KJAR.

You can define these descriptors in a file called **kie-deployment-descriptor.xml** and place this file next to your **kmodule.xml** file in the **META-INF** folder. You can change this default location and the file name by specifying it as a system parameter:

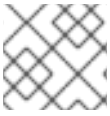
```
-Dorg.kie.deployment.desc.location=file:/path/to/file/company-deployment-descriptor.xml
```

12.1. DEPLOYMENT DESCRIPTOR CONFIGURATION

Deployment descriptors allow the user to configure the execution server on multiple levels:

- *Server level*: The main level and the one that applies to all KJARs deployed on the server.
- *KJAR level*: This enables you to configure descriptors on a per KJAR basis.
- *Deploy time level*: Descriptors that apply while a KJAR is being deployed.

The granular configuration items specified by the deployment descriptors take precedence over the server level ones, except in case of configuration items that are collection based, which are merged. The hierarchy works like this: *deploy time configuration > KJAR configuration > server configuration* .



NOTE

The deploy time configuration applies to deployments done via the REST API.

For example, if the persistence mode (one of the items you can configure) defined at the server level is **NONE** but the same mode is specified as **JPA** at the KJAR level, the actual mode will be **JPA** for that KJAR. If nothing is specified for the persistence mode in the deployment descriptor for that KJAR (or if there is no deployment descriptor), it will fall back to the server level configuration, which in this case is **NONE** (or to **JPA** if there is no server level deployment descriptor).

What Can You Configure?

High level technical configuration details can be configured via deployment descriptors. The following table lists these along with the permissible and default values for each.

Table 12.1. Deployment Descriptors

Configuration	XML Entry	Permissible Values	Default Value
Persistence unit name for runtime data	persistence-unit	Any valid persistence package name	org.jbpm.domain
Persistence unit name for audit data	audit-persistence-unit	Any valid persistence package name	org.jbpm.domain
Persistence mode	persistence-mode	JPA, NONE	JPA
Audit mode	audit-mode	JPA, JMS or NONE	JPA
Runtime Strategy	runtime-strategy	SINGLETON, PER_REQUEST or PER_PROCESS_INSTANCE	SINGLETON
List of Event Listeners to be registered	event-listeners	Valid listener class names as **ObjectModel**	No default value
List of Task Event Listeners to be registered	task-event-listeners	Valid listener class names as **ObjectModel**	No default value
List of Work Item Handlers to be registered	work-item-handlers	Valid Work Item Handler classes given as **NamedObjectHandler**	No default value
List of Globals to be registered	globals	Valid Global variables given as **NamedObjectModel**	No default value
Marshalling strategies to be registered (for pluggable variable persistence)	marshalling-strategies	Valid **ObjectModel** classes	No default value
Required Roles to be granted access to the resources of the KJAR	required-roles	String role names	No default value
Additional Environment Entries for KIE session	environment-entries	Valid **NamedObjectModel**	No default value
Additional configuration options of KIE session	configurations	Valid **NamedObjectModel**	No default value

Configuration	XML Entry	Permissible Values	Default Value
Classes used for serialization in the remote services	remoteable-class	Valid **CustomClass**	No default value

WARNING

Do not use the Singleton runtime strategy with the EJB Timer Scheduler (the default scheduler in KIE Server) in a production environment. This combination can result in Hibernate problems under load. Per process instance runtime strategy is recommended if there is no specific reason to use other strategies. For more information about this limitation, see Hibernate issues with Singleton strategy and EJBTimerScheduler.

12.2. MANAGING DEPLOYMENT DESCRIPTORS

Deployment descriptors can be configured in Business Central in **Menu → Design → $PROJECT_NAME → Settings → Deployments**.

Every time a project is created, a stock **kie-deployment-descriptor.xml** file is generated with default values.

It is not necessary to provide a full deployment descriptor for all KJARs. Providing partial deployment descriptors is possible and recommended. For example, if you need to use a different audit mode, you can specify that for the KJAR only, all other properties will have the default value defined at the server level.

When using **OVERRIDE_ALL** merge mode, all configuration items must be specified, because the relevant KJAR will always use specified configuration and will not merge with any other deployment descriptor in the hierarchy.

12.3. RESTRICTING ACCESS TO THE RUNTIME ENGINE

The **required-roles** configuration item can be edited in the deployment descriptors. This property restricts access to the runtime engine on a per-KJAR or per-server level by ensuring that access to certain processes is only granted to users that belong to groups defined by this property.

The security role can be used to restrict access to process definitions or restrict access at run time.

The default behavior is to add required roles to this property based on repository restrictions. You can edit these properties manually if required by providing roles that match actual roles defined in the security realm.

Procedure

1. To open the project deployment descriptors configuration in Business Central, open **Menu → Design → $PROJECT_NAME → Settings → Deployments**.

2. From the list of configuration settings, click **Required Roles**, then click **Add Required Role**.
3. In the **Add Required Role** window, type the name of the role that you want to have permission to access this deployment, then click **Add**.
4. To add more roles with permission to access the deployment, repeat the previous steps.
5. When you have finished adding all required roles, click **Save**.

CHAPTER 13. PROMETHEUS METRICS MONITORING IN RED HAT DECISION MANAGER

Prometheus is an open-source systems monitoring toolkit that you can use with Red Hat Decision Manager to collect and store metrics related to the execution of business rules, processes, Decision Model and Notation (DMN) models, and other Red Hat Decision Manager assets. You can access the stored metrics through a REST API call to the KIE Server, through the Prometheus expression browser, or using a data-graphing tool such as Grafana.

You can configure Prometheus metrics monitoring for an on-premise KIE Server instance, for KIE Server on Spring Boot, or for a KIE Server deployment on Red Hat OpenShift Container Platform.

For the list of available metrics that KIE Server exposes with Prometheus, download the **Red Hat Decision Manager 7.8.0 Source Distribution** from the Red Hat Customer Portal and navigate to **~/rhdm-7.8.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus**.

IMPORTANT

Red Hat support for Prometheus is limited to the setup and configuration recommendations provided in Red Hat product documentation.

13.1. CONFIGURING PROMETHEUS METRICS MONITORING FOR KIE SERVER

You can configure your KIE Server instances to use Prometheus to collect and store metrics related to your business asset activity in Red Hat Decision Manager. For the list of available metrics that KIE Server exposes with Prometheus, download the **Red Hat Decision Manager 7.8.0 Source Distribution** from the Red Hat Customer Portal and navigate to **~/rhdm-7.8.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus**.

Prerequisites

- KIE Server is installed.
- You have **kie-server** user role access to KIE Server.
- Prometheus is installed. For information about downloading and using Prometheus, see the Prometheus documentation page.

Procedure

1. In your KIE Server instance, set the **org.kie.prometheus.server.ext.disabled** system property to **false** to enable the Prometheus extension. You can define this property when you start KIE Server or in the **standalone.xml** or **standalone-full.xml** file of Red Hat Decision Manager distribution.
2. If you are running Red Hat Decision Manager on Spring Boot, configure the required key in the **application.properties** system property:

   Spring Boot application.properties key for Red Hat Decision Manager and Prometheus

   ■

```
kieserver.drools.enabled=true
kieserver.dmn.enabled=true
kieserver.prometheus.enabled=true
```

3. In the **prometheus.yaml** file of your Prometheus distribution, add the following settings in the **scrape_configs** section to configure Prometheus to scrape metrics from KIE Server:

   Scrape configurations in prometheus.yaml file

```
scrape_configs:
  - job_name: 'kie-server'
    metrics_path: /SERVER_PATH/services/rest/metrics
    basicAuth:
      username: USER_NAME
      password: PASSWORD
    static_configs:
      - targets: ["HOST:PORT"]
```

   Scrape configurations in prometheus.yaml file for Spring Boot (if applicable)

```
scrape_configs:
  - job_name: 'kie'
    metrics_path: /rest/metrics
    static_configs:
      - targets: ["HOST:PORT"]
```

   Replace the values according to your KIE Server location and settings.

4. Start the KIE Server instance.

   Example start command for Red Hat Decision Manager on Red Hat JBoss EAP

```
$ cd ~/EAP_HOME/bin
$ ./standalone.sh --c standalone-full.xml
```

   After you start the configured KIE Server instance, Prometheus begins collecting metrics and KIE Server publishes the metrics to the REST API endpoint **http://HOST:PORT/SERVER/services/rest/metrics** (or on Spring Boot, to **http://HOST:PORT/rest/metrics**).

5. In a REST client or curl utility, send a REST API request with the following components to verify that KIE Server is publishing the metrics:
   For REST client:

   - **Authentication**: Enter the user name and password of the KIE Server user with the **kie-server** role.
   - **HTTP Headers**: Set the following header:
     - **Accept**: **application/json**
   - **HTTP method**: Set to **GET**.

- **URL**: Enter the KIE Server REST API base URL and metrics endpoint, such as **http://localhost:8080/kie-server/services/rest/metrics** (or on Spring Boot, **http://localhost:8080/rest/metrics**).

For curl utility:

- **-u**: Enter the user name and password of the KIE Server user with the **kie-server** role.
- **-H**: Set the following header:
  - **accept**: **application/json**
- **-X**: Set to **GET**.
- **URL**: Enter the KIE Server REST API base URL and metrics endpoint, such as **http://localhost:8080/kie-server/services/rest/metrics** (or on Spring Boot, **http://localhost:8080/rest/metrics**).

Example curl command for Red Hat Decision Manager on Red Hat JBoss EAP

```
curl -u 'baAdmin:password@1' -X GET "http://localhost:8080/kie-server/services/rest/metrics"
```

Example curl command for Red Hat Decision Manager on Spring Boot

```
curl -u 'baAdmin:password@1' -X GET "http://localhost:8080/rest/metrics"
```

Example server response

```
# HELP kie_server_container_started_total Kie Server Started Containers
# TYPE kie_server_container_started_total counter
kie_server_container_started_total{container_id="task-assignment-kjar-1.0",} 1.0
# HELP solvers_running Number of solvers currently running
# TYPE solvers_running gauge
solvers_running 0.0
# HELP dmn_evaluate_decision_nanosecond DMN Evaluation Time
# TYPE dmn_evaluate_decision_nanosecond histogram
# HELP solver_duration_seconds Time in seconds it took solver to solve the constraint
problem
# TYPE solver_duration_seconds summary
solver_duration_seconds_count{solver_id="100tasks-5employees.xml",} 1.0
solver_duration_seconds_sum{solver_id="100tasks-5employees.xml",} 179.828255925
solver_duration_seconds_count{solver_id="24tasks-8employees.xml",} 1.0
solver_duration_seconds_sum{solver_id="24tasks-8employees.xml",} 179.995759653
# HELP drl_match_fired_nanosecond Drools Firing Time
# TYPE drl_match_fired_nanosecond histogram
# HELP dmn_evaluate_failed_count DMN Evaluation Failed
# TYPE dmn_evaluate_failed_count counter
# HELP kie_server_start_time Kie Server Start Time
# TYPE kie_server_start_time gauge
kie_server_start_time{name="myapp-kieserver",server_id="myapp-
kieserver",location="http://myapp-kieserver-demo-
monitoring.127.0.0.1.nip.io:80/services/rest/server",version="7.4.0.redhat-20190428",}
1.557221271502E12
# HELP kie_server_container_running_total Kie Server Running Containers
# TYPE kie_server_container_running_total gauge
```

```
kie_server_container_running_total{container_id="task-assignment-kjar-1.0",} 1.0
# HELP solver_score_calculation_speed Number of moves per second for a particular solver solving the constraint problem
# TYPE solver_score_calculation_speed summary
solver_score_calculation_speed_count{solver_id="100tasks-5employees.xml",} 1.0
solver_score_calculation_speed_sum{solver_id="100tasks-5employees.xml",} 6997.0
solver_score_calculation_speed_count{solver_id="24tasks-8employees.xml",} 1.0
solver_score_calculation_speed_sum{solver_id="24tasks-8employees.xml",} 19772.0
```

If the metrics are not available in KIE Server, review and verify the KIE Server and Prometheus configurations described in this section.

You can also interact with your collected metrics in the Prometheus expression browser at **http://HOST:PORT/graph**, or integrate your Prometheus data source with a data-graphing tool such as Grafana:

Figure 13.1. Prometheus expression browser with KIE Server metrics

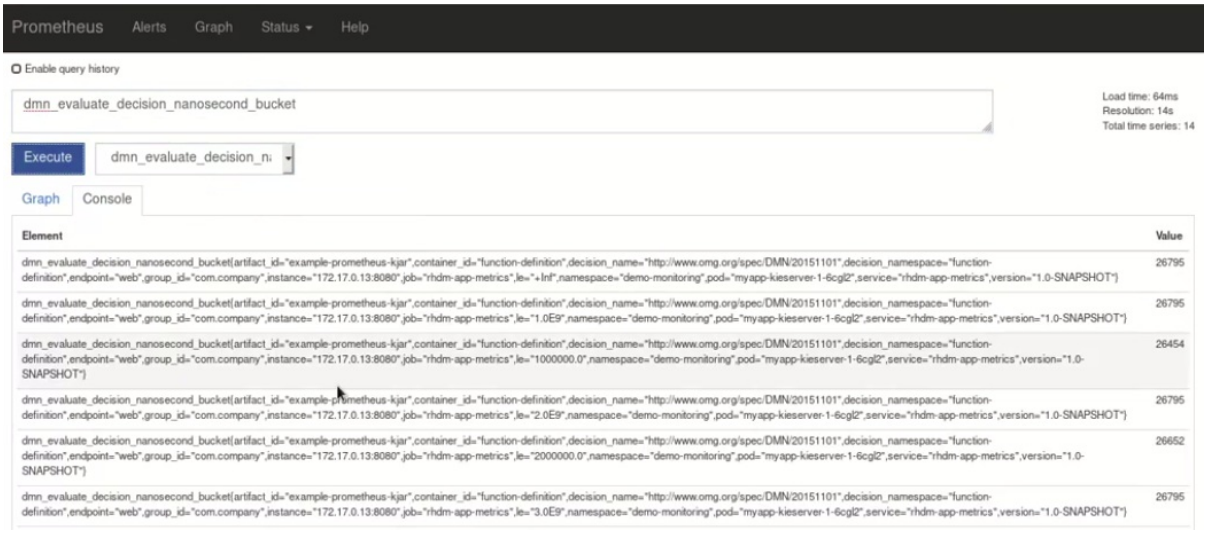


Figure 13.2. Prometheus expression browser with KIE Server target

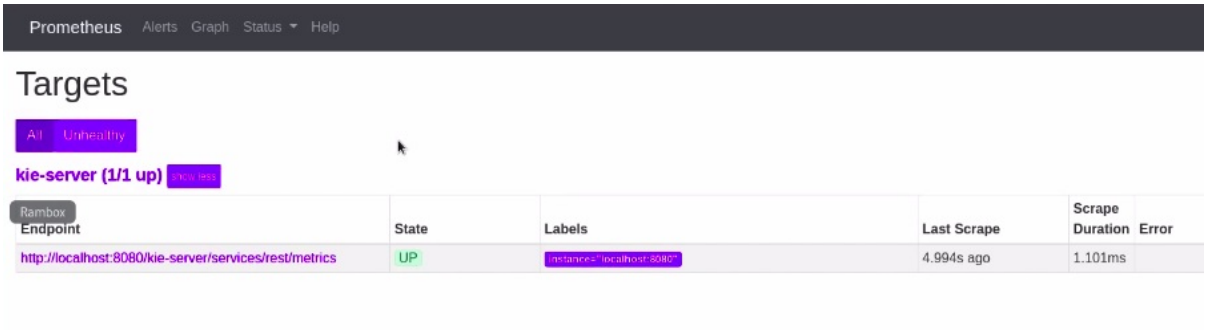
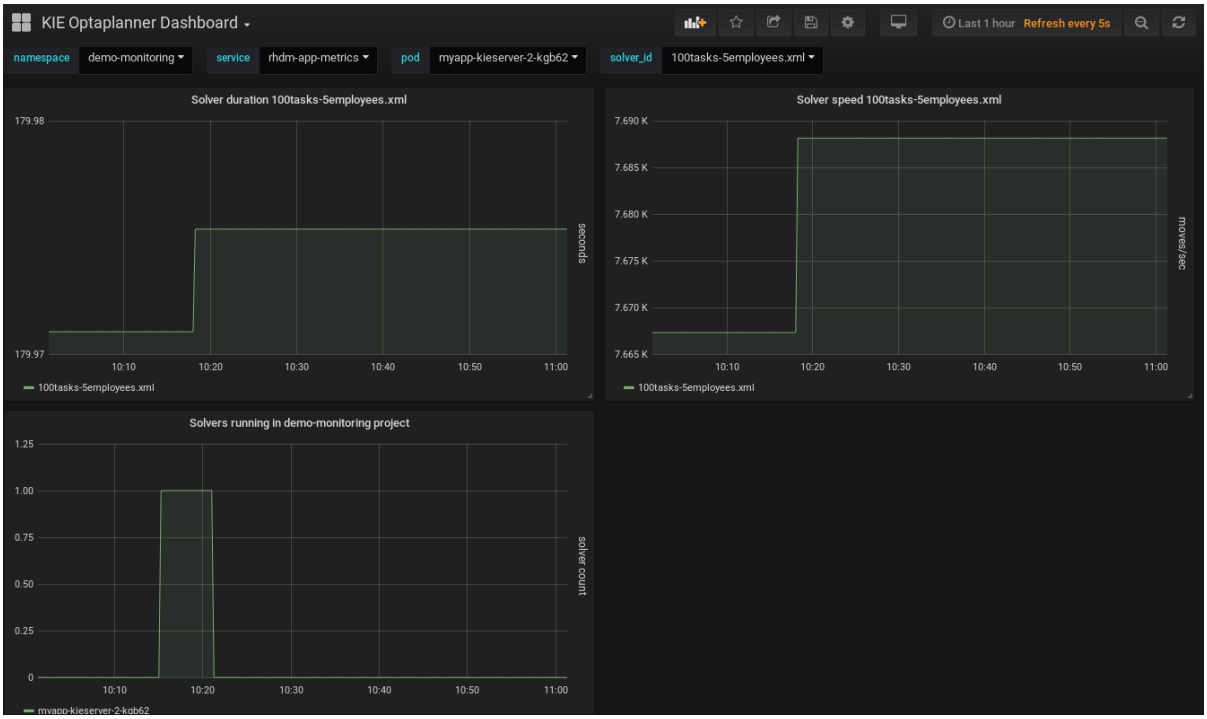


Figure 13.3. Grafana dashboard with KIE Server metrics for DMN models

Figure 13.4. Grafana dashboard with KIE Server metrics for solvers



Additional resources

- Getting Started with Prometheus
- Grafana Support for Prometheus
- Using Prometheus in Grafana

13.2. CONFIGURING PROMETHEUS METRICS MONITORING FOR KIE SERVER ON RED HAT OPENSHIFT CONTAINER PLATFORM

You can configure your KIE Server deployment on Red Hat OpenShift Container Platform to use Prometheus to collect and store metrics related to your business asset activity in Red Hat Decision Manager. For the list of available metrics that KIE Server exposes with Prometheus, download the **Red Hat Decision Manager 7.8.0 Source Distribution** from the Red Hat Customer Portal and navigate to **~/rhdm-7.8.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus**.

Prerequisites

Prerequisites

- KIE Server is installed and deployed on Red Hat OpenShift Container Platform. For more information about KIE Server on OpenShift, see the relevant OpenShift deployment option in the Product documentation for Red Hat Decision Manager 7.8 .
- You have **kie-server** user role access to KIE Server.
- Prometheus Operator is installed. For information about downloading and using Prometheus Operator, see the Prometheus Operator project in GitHub.

Procedure

1. In the **DeploymentConfig** object of your KIE Server deployment on OpenShift, set the **PROMETHEUS_SERVER_EXT_DISABLED** environment variable to **false** to enable the Prometheus extension. You can set this variable in the OpenShift web console or use the **oc** command in a command terminal:

```
oc set env dc/<dc_name> PROMETHEUS_SERVER_EXT_DISABLED=false -n
<namespace>
```

   If you have not yet deployed your KIE Server on OpenShift, then in the OpenShift template that you plan to use for your OpenShift deployment (for example, **rhdm78-prod-immutable-kieserver.yaml**), you can set the **PROMETHEUS_SERVER_EXT_DISABLED** template parameter to **false** to enable the Prometheus extension.

   If you are using the OpenShift Operator to deploy KIE Server on OpenShift, then in your KIE Server configuration, set the **PROMETHEUS_SERVER_EXT_DISABLED** environment variable to **false** to enable the Prometheus extension:

```
apiVersion: app.kiegroup.org/v1
kind: KieApp
metadata:
  name: enable-prometheus
spec:
  environment: rhpam-trial
  objects:
    servers:
    - env:
      - name: PROMETHEUS_SERVER_EXT_DISABLED
        value: "false"
```

2. Create a **service-metrics.yaml** file to add a service that exposes the metrics from KIE Server to Prometheus:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    description: RHDM Prometheus metrics exposed
  labels:
    app: myapp-kieserver
    application: myapp-kieserver
    template: myapp-kieserver
    metrics: rhdm
  name: rhdm-app-metrics
```

```
spec:
  ports:
    - name: web
      port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    deploymentConfig: myapp-kieserver
  sessionAffinity: None
  type: ClusterIP
```

3. In a command terminal, use the **oc** command to apply the **service-metrics.yaml** file to your OpenShift deployment:

```
oc apply -f service-metrics.yaml
```

4. Create an OpenShift secret, such as **metrics-secret**, to access the Prometheus metrics on KIE Server. The secret must contain the "username" and "password" elements with KIE Server user credentials. For information about OpenShift secrets, see the Secrets chapter in the OpenShift *Developer Guide*.

5. Create a **service-monitor.yaml** file that defines the **ServiceMonitor** object. A service monitor enables Prometheus to connect to the KIE Server metrics service.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: rhdm-service-monitor
  labels:
    team: frontend
spec:
  selector:
    matchLabels:
      metrics: rhdm
  endpoints:
    - port: web
      path: /services/rest/metrics
      basicAuth:
        password:
          name: metrics-secret
          key: password
        username:
          name: metrics-secret
          key: username
```

6. In a command terminal, use the **oc** command to apply the **service-monitor.yaml** file to your OpenShift deployment:

```
oc apply -f service-monitor.yaml
```

After you complete these configurations, Prometheus begins collecting metrics and KIE Server publishes the metrics to the REST API endpoint **http://HOST:PORT/kie-server/services/rest/metrics**.

You can interact with your collected metrics in the Prometheus expression browser at **http://HOST:PORT/graph**, or integrate your Prometheus data source with a data-graphing tool such as Grafana.

The host and port for the Prometheus expression browser location **http://HOST:PORT/graph** was defined in the route where you exposed the Prometheus web console when you installed the Prometheus Operator. For information about OpenShift routes, see the Routes chapter in the OpenShift *Architecture* documentation.

Figure 13.5. Prometheus expression browser with KIE Server metrics

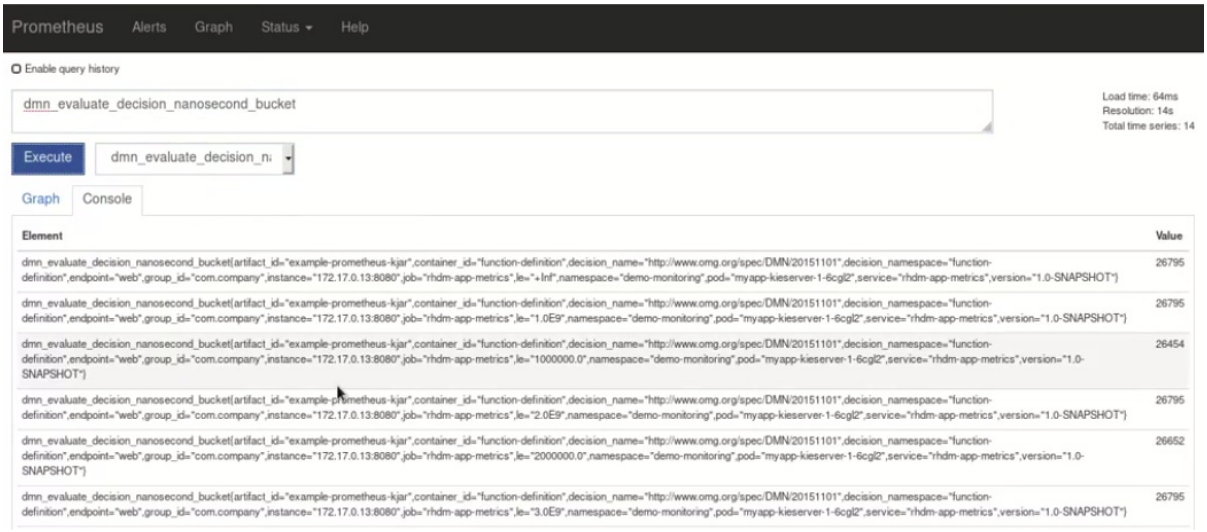


Figure 13.6. Prometheus expression browser with KIE Server target

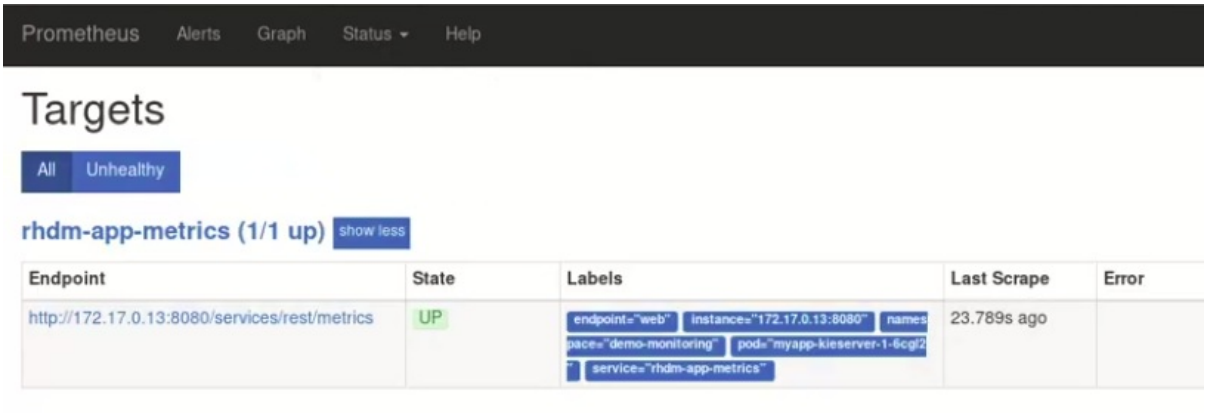
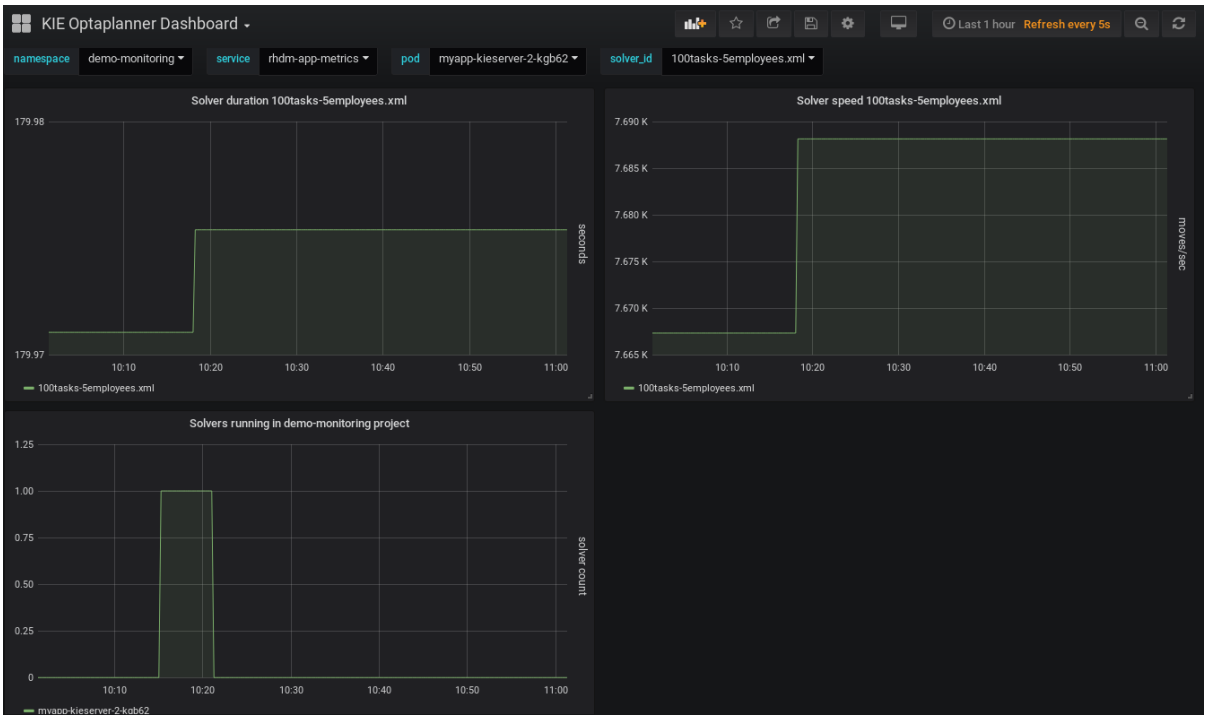


Figure 13.7. Grafana dashboard with KIE Server metrics for DMN models

Figure 13.8. Grafana dashboard with KIE Server metrics for solvers



Additional resources

- Prometheus Operator
- Getting started with the Prometheus Operator
- Prometheus RBAC
- Grafana Support for Prometheus
- Using Prometheus in Grafana
- OpenShift deployment options in Product documentation for Red Hat Decision Manager 7.8

13.3. EXTENDING PROMETHEUS METRICS MONITORING IN KIE SERVER WITH CUSTOM METRICS

After you configure your KIE Server instance to use Prometheus metrics monitoring, you can extend the Prometheus functionality in KIE Server to use custom metrics according to your business needs. Prometheus then collects and stores your custom metrics along with the default metrics that KIE Server exposes with Prometheus.

As an example, this procedure defines custom Decision Model and Notation (DMN) metrics to be collected and stored by Prometheus.

Prerequisites

- Prometheus metrics monitoring is configured for your KIE Server instance. For information about Prometheus configuration with KIE Server on-premise, see Section 13.1, "Configuring Prometheus metrics monitoring for KIE Server". For information about Prometheus configuration with KIE Server on Red Hat OpenShift Container Platform, see Section 13.2, "Configuring Prometheus metrics monitoring for KIE Server on Red Hat OpenShift Container Platform".

Procedure

1. Create an empty Maven project and define the following packaging type and dependencies in the **pom.xml** file for the project:

Example pom.xml file in the sample project

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.39.0.Final-redhat-00005</version.org.kie>
</properties>

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

```
    <artifactId>jbpm-executor</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-core</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>simpleclient</artifactId>
    <version>0.5.0</version>
  </dependency>
</dependencies>
```

2. Implement the relevant listener from the **org.kie.server.services.prometheus.PrometheusMetricsProvider** interface as part of the custom listener class that defines your custom Prometheus metrics, as shown in the following example:

Sample implementation of the DMNRuntimeEventListener listener in a custom listener class

```
package org.kie.server.ext.prometheus;

import io.prometheus.client.Gauge;
import org.kie.dmn.api.core.ast.DecisionNode;
import org.kie.dmn.api.core.event.AfterEvaluateBKMEvent;
import org.kie.dmn.api.core.event.AfterEvaluateContextEntryEvent;
import org.kie.dmn.api.core.event.AfterEvaluateDecisionEvent;
import org.kie.dmn.api.core.event.AfterEvaluateDecisionServiceEvent;
import org.kie.dmn.api.core.event.AfterEvaluateDecisionTableEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateBKMEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateContextEntryEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateDecisionEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateDecisionServiceEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateDecisionTableEvent;
import org.kie.dmn.api.core.event.DMNRuntimeEventListener;
import org.kie.server.api.model.ReleaseId;
import org.kie.server.services.api.KieContainerInstance;

public class ExampleCustomPrometheusMetricListener implements
DMNRuntimeEventListener {

    private final KieContainerInstance kieContainer;

    private final Gauge randomGauge = Gauge.build()
            .name("random_gauge_nanosecond")
            .help("Random gauge as an example of custom KIE Prometheus metric")
            .labelNames("container_id", "group_id", "artifact_id", "version",
"decision_namespace", "decision_name")
            .register();

    public ExampleCustomPrometheusMetricListener(KieContainerInstance
containerInstance) {
```

```
        kieContainer = containerInstance;
    }

    public void beforeEvaluateDecision(BeforeEvaluateDecisionEvent e) {
    }

    public void afterEvaluateDecision(AfterEvaluateDecisionEvent e) {
        DecisionNode decisionNode = e.getDecision();
        ReleaseId releaseId = kieContainer.getResource().getReleaseId();
        randomGauge.labels(kieContainer.getContainerId(), releaseId.getGroupId(),
                releaseId.getArtifactId(), releaseId.getVersion(),
                decisionNode.getModelName(), decisionNode.getModelNamespace())
            .set((int) (Math.random() * 100));
    }

    public void beforeEvaluateBKM(BeforeEvaluateBKMEvent event) {
    }

    public void afterEvaluateBKM(AfterEvaluateBKMEvent event) {
    }

    public void beforeEvaluateContextEntry(BeforeEvaluateContextEntryEvent event) {
    }

    public void afterEvaluateContextEntry(AfterEvaluateContextEntryEvent event) {
    }

    public void beforeEvaluateDecisionTable(BeforeEvaluateDecisionTableEvent event) {
    }

    public void afterEvaluateDecisionTable(AfterEvaluateDecisionTableEvent event) {
    }

    public void beforeEvaluateDecisionService(BeforeEvaluateDecisionServiceEvent event) {
    }

    public void afterEvaluateDecisionService(AfterEvaluateDecisionServiceEvent event) {
    }
}
```

The **PrometheusMetricsProvider** interface contains the required listeners for collecting Prometheus metrics. The interface is incorporated by the **kie-server-services-prometheus** dependency that you declared in your project **pom.xml** file.

In this example, the **ExampleCustomPrometheusMetricListener** class implements the **DMNRuntimeEventListener** listener (from the **PrometheusMetricsProvider** interface) and defines the custom DMN metrics to be collected and stored by Prometheus.

3. Implement the **PrometheusMetricsProvider** interface as part of a custom metrics provider class that associates your custom listener with the **PrometheusMetricsProvider** interface, as shown in the following example:

Sample implementation of the PrometheusMetricsProvider interface in a custom metrics provider class

```
package org.kie.server.ext.prometheus;
```

```
import org.jbpm.executor.AsynchronousJobListener;
import org.jbpm.services.api.DeploymentEventListener;
import org.kie.api.event.rule.AgendaEventListener;
import org.kie.api.event.rule.DefaultAgendaEventListener;
import org.kie.dmn.api.core.event.DMNRuntimeEventListener;
import org.kie.server.services.api.KieContainerInstance;
import org.kie.server.services.prometheus.PrometheusMetricsProvider;
import org.optaplanner.core.impl.phase.event.PhaseLifecycleListener;
import org.optaplanner.core.impl.phase.event.PhaseLifecycleListenerAdapter;

public class MyPrometheusMetricsProvider implements PrometheusMetricsProvider {

    public DMNRuntimeEventListener createDMNRuntimeEventListener(KieContainerInstance
kContainer) {
        return new ExampleCustomPrometheusMetricListener(kContainer);
    }

    public AgendaEventListener createAgendaEventListener(String kieSessionId,
KieContainerInstance kContainer) {
        return new DefaultAgendaEventListener();
    }

    public PhaseLifecycleListener createPhaseLifecycleListener(String solverId) {
        return new PhaseLifecycleListenerAdapter() {
        };
    }

    public AsynchronousJobListener createAsynchronousJobListener() {
        return null;
    }

    public DeploymentEventListener createDeploymentEventListener() {
        return null;
    }
}
```

In this example, the **MyPrometheusMetricsProvider** class implements the **PrometheusMetricsProvider** interface and includes your custom **ExampleCustomPrometheusMetricListener** listener class.

4. To make the new metrics provider discoverable for KIE Server, create a **META-INF/services/org.kie.server.services.prometheus.PrometheusMetricsProvider** file in your Maven project and add the fully qualified class name of the **PrometheusMetricsProvider** implementation class within the file. For this example, the file contains the single line **org.kie.server.ext.prometheus.MyPrometheusMetricsProvider**.

5. Build your project and copy the resulting JAR file into the ~/**kie-server.war/WEB-INF/lib** directory of your project. For example, on Red Hat JBoss EAP, the path to this directory is ***EAP_HOME*/standalone/deployments/kie-server.war/WEB-INF/lib**.

6. Start the KIE Server and deploy the built project to the running KIE Server. You can deploy the project using the Business Central interface or the KIE Server REST API (a **PUT** request to **http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}**).
After your project is deployed on a running KIE Server, Prometheus begins collecting metrics and KIE Server publishes the metrics to the REST API endpoint

http://HOST:PORT/SERVER/services/rest/metrics (or on Spring Boot, to **http://HOST:PORT/rest/metrics**).

CHAPTER 14. CONFIGURING OPENSHIFT CONNECTION TIMEOUT

By default, the OpenShift route is configured to time out HTTP requests that are longer than 30 seconds. This may cause session timeout issues in Business Central resulting in the following behaviors:

- "Unable to complete your request. The following exception occurred: (TypeError) : Cannot read property 'indexOf' of null."
- "Unable to complete your request. The following exception occurred: (TypeError) : b is null."
- A blank page is displayed when clicking the **Project** or **Server** links in Business Central.

All Business Central templates already include extended timeout configuration.

To configure longer timeout on Business Central OpenShift routes, add the **haproxy.router.openshift.io/timeout: 60s** annotation on the target route:

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

For a full list of global route-specific timeout annotations, see the OpenShift Documentation.

CHAPTER 15. DEFINE THE LDAP LOGIN DOMAIN

When you are setting up Red Hat Decision Manager to use LDAP for authentication and authorization, define the LDAP login domain because the Git SSH authentication may use another security domain.

To define the LDAP login domain, use the **org.uberfire.domain** system property. For example, on Red Hat JBoss Enterprise Application Platform, add this property in the **standalone.xml** file as shown:

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

Ensure that the authenticated user has appropriate roles (**admin**,**analyst**,**reviewer**) associated with it in LDAP.

CHAPTER 16. 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

16.1. 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/decision-central/rest/repositories
```

- For KIE Server:

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

CHAPTER 17. KIE SERVER SYSTEM PROPERTIES

The KIE Server accepts the following system properties (bootstrap switches) to configure the behavior of the server:

Table 17.1. System properties for disabling KIE Server extensions

Property	Values	Default	Description
org.drools.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the Business Rule Management (BRM) support (for example, rules support).
org.optaplanner.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the Red Hat Business Optimizer support.
org.kie.prometheus.server.ext.disabled	**true**, **false**	**true**	If set to **true**, disables the Prometheus Server extension.
org.kie.scenariosimulation.server.ext.disabled	**true**, **false**	**true**	If set to **true**, disables the Test scenario Server extension.
org.kie.dmn.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the KIE Server DMN support.
org.kie.swagger.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the KIE Server swagger documentation support

NOTE

Some Decision Manager controller properties listed in the following table are marked as required. Set these properties when you create or remove KIE Server containers in Business Central. If you use the KIE Server separately without any interaction with Business Central, you do not need to set the required properties.

Table 17.2. System properties required for Decision Manager controller

Property	Values	Default	Description
org.kie.server.id	String	N/A	An arbitrary ID to be assigned to the server. If a headless Decision Manager controller is configured outside of Business Central, this is the ID under which the server connects to the headless Decision Manager controller to fetch the KIE container configurations. If not provided, the ID is automatically generated.

Property	Values	Default	Description
org.kie.server.user	String	**kieserver**	The user name used to connect with the KIE Server from the Decision Manager controller, required when running in managed mode. Set this property in Business Central system properties. Set this property when using a Decision Manager controller.
org.kie.server.pwd	String	**kieserver1!**	The password used to connect with the KIE Server from the Decision Manager controller, required when running in managed mode. Set this property in Business Central system properties. Set this property when using a Decision Manager controller.
org.kie.server.token	String	N/A	A property that enables you to use token-based authentication between the Decision Manager controller and the KIE Server instead of the basic user name and password authentication. The Decision Manager controller sends the token as a parameter in the request header. The server requires long-lived access tokens because the tokens are not refreshed.
org.kie.server.location	URL	N/A	The URL of the KIE Server instance used by the Decision Manager controller to call back on this server, for example, **http://localhost:8230/kie-server/services/rest/server**. Setting this property is required when using a Decision Manager controller.
org.kie.server.controller	Comma-separated list	N/A	A comma-separated list of URLs to the Decision Manager controller REST endpoints, for example, **http://localhost:8080/decision-central/rest/controller**. Setting this property is required when using a Decision Manager controller.
org.kie.server.controller.user	String	**kieserver**	The user name to connect to the Decision Manager controller REST API. Setting this property is required when using a Decision Manager controller.
org.kie.server.controller.pwd	String	**kieserver1!**	The password to connect to the Decision Manager controller REST API. Setting this property is required when using a Decision Manager controller.

Property	Values	Default	Description
org.kie.server.controller.token	String	N/A	A property that enables you to use token-based authentication between the KIE Server and the Decision Manager controller instead of the basic user name and password authentication. The server sends the token as a parameter in the request header. The server requires long-lived access tokens because the tokens are not refreshed.
org.kie.server.controller.connect	Long	**10000**	The waiting time in milliseconds between repeated attempts to connect the KIE Server to the Decision Manager controller when the server starts.

Table 17.3. System properties for loading keystore

Property	Values	Default	Description
kie.keystore.keyStoreURL	URL	N/A	The URL is used to load a Java Cryptography Extension KeyStore (JCEKS). For example, **file:///home/kie/keystores/keystore.jceks**.
kie.keystore.keyStorePwd	String	N/A	The password is used for the JCEKS.
kie.keystore.key.server.alias	String	N/A	The alias name of the key for REST services where the password is stored.
kie.keystore.key.server.pwd	String	N/A	The password of an alias for REST services.
kie.keystore.key.ctrl.alias	String	N/A	The alias of the key for default REST Decision Manager controller.
kie.keystore.key.ctrl.pwd	String	N/A	The password of an alias for default REST Decision Manager controller.

Table 17.4. Other system properties

Property	Values	Default	Description
kie.maven.settings.custom	Path	N/A	The location of a custom **settings.xml** file for Maven configuration.

Property	Values	Default	Description
kie.server.jms.queues.response	String	**queue/KIE.SERVER.RESPONSE**	The response queue JNDI name for JMS.
org.drools.server.filter.classes	**true**, **false**	**false**	When set to **true**, the Drools KIE Server extension accepts custom classes annotated by the **XmlRootElement** or **Remotable** annotations only.
org.kie.server.domain	String	N/A	The JAAS **LoginContext** domain used to authenticate users when using JMS.
org.kie.server.repo	Path	**.**	The location where KIE Server state files are stored.
org.kie.server.sync.deploy	**true**, **false**	**false**	A property that instructs the KIE Server to hold the deployment until the Decision Manager controller provides the container deployment configuration. This property only affects servers running in managed mode. The following options are available: * **false**: The connection to the Decision Manager controller is asynchronous. The application starts, connects to the Decision Manager controller, and once successful, deploys the containers. The application accepts requests even before the containers are available. * **true**: The deployment of the server application joins the Decision Manager controller connection thread with the main deployment and awaits its completion. This option can lead to a potential deadlock in case more applications are on the same server. Use only one application on one server instance.

Property	Values	Default	Description
org.kie.server.startup.strategy	**ControllerBasedStartupStrategy**, **LocalContainersStartupStrategy**	**ControllerBasedStartupStrategy**	The Startup strategy of KIE Server used to control the KIE containers that are deployed and the order in which they are deployed.
org.kie.server.mgmt.api.disabled	**true**, **false**	**false**	When set to **true**, disables KIE Server management API.
org.kie.server.xstream.enabled.packages	Java packages like **org.kie.example**. You can also specify wildcard expressions like **org.kie.example.***.	N/A	A property that specifies additional packages to whitelist for marshalling using XStream.
org.kie.store.services.class	String	**org.drools.persistence.jpa.KnowledgeStoreServiceImpl**	Fully qualified name of the class that implements **KieStoreServices** that are responsible for bootstrapping KieSession instances.
org.kie.server.strict.id.format	**true**, **false**	**false**	While using JSON marshalling, if the property is set to **true**, it will always return a response in the proper JSON format. For example, if the original response contains only a single number, then the response is wrapped in a JSON format. For example, **{"value" : 1}**.

CHAPTER 18. KIE SERVER CAPABILITIES AND EXTENSIONS

The capabilities in KIE Server are determined by plug-in extensions that you can enable, disable, or further extend to meet your business needs. KIE Server supports the following default capabilities and extensions:

Table 18.1. KIE Server capabilities and extensions

Capability name	Extension name	Description
KieServer	**KieServer**	Provides the core capabilities of KIE Server, such as creating and disposing KIE containers on your server instance
BRM	**Drools**	Provides the Business Rule Management (BRM) capabilities, such as inserting facts and executing business rules
BRP	**OptaPlanner**	Provides the Business Resource Planning (BRP) capabilities, such as implementing solvers
DMN	**DMN**	Provides the Decision Model and Notation (DMN) capabilities, such as managing DMN data types and executing DMN models
Swagger	**Swagger**	Provides the Swagger web-interface capabilities for interacting with the KIE Server REST API

To view the supported extensions of a running KIE Server instance, send a **GET** request to the following REST API endpoint and review the XML or JSON server response:

Base URL for GET request for KIE Server information

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

Example JSON response with KIE Server information

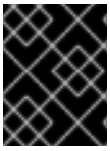
```
{
  "type": "SUCCESS",
  "msg": "Kie Server info",
  "result": {
    "kie-server-info": {
      "id": "test-kie-server",
      "version": "7.26.0.20190818-050814",
      "name": "test-kie-server",
      "location": "http://localhost:8080/kie-server/services/rest/server",
      "capabilities": [
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

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

To enable or disable KIE Server extensions, configure the related ***.server.ext.disabled** KIE Server system property. For example, to disable the **BRM** capability, set the system property **org.drools.server.ext.disabled=true**. For all KIE Server system properties, see Chapter 17, *KIE Server system properties*.

By default, KIE Server extensions are exposed through REST or JMS data transports and use predefined client APIs. You can extend existing KIE Server capabilities with additional REST endpoints, extend supported transport methods beyond REST or JMS, or extend functionality in the KIE Server client.

This flexibility in KIE Server functionality enables you to adapt your KIE Server instances to your business needs, instead of adapting your business needs to the default KIE Server capabilities.



IMPORTANT

If you extend KIE Server functionality, Red Hat does not support the custom code that you use as part of your custom implementations and extensions.

18.1. EXTENDING AN EXISTING KIE SERVER CAPABILITY WITH A CUSTOM REST API ENDPOINT

The KIE Server REST API enables you to interact with your KIE containers and business assets (such as business rules, processes, and solvers) in Red Hat Decision Manager without using the Business Central user interface. The available REST endpoints are determined by the capabilities enabled in your KIE Server system properties (for example, **org.drools.server.ext.disabled=false** for the **BRM** capability). You can extend an existing KIE Server capability with a custom REST API endpoint to further adapt the KIE Server REST API to your business needs.

As an example, this procedure extends the **Drools** KIE Server extension (for the **BRM** capability) with the following custom REST API endpoint:

Example custom REST API endpoint

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

This example custom endpoint accepts a list of facts to be inserted into the working memory of the decision engine, automatically executes all rules, and retrieves all objects from the KIE session in the specified KIE container.

Procedure

1. Create an empty Maven project and define the following packaging type and dependencies in the **pom.xml** file for the project:

Example pom.xml file in the sample project

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.39.0.Final-redhat-00005</version.org.kie>
</properties>

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

```
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
  </dependency>
</dependencies>
```

2. Implement the **org.kie.server.services.api.KieServerApplicationComponentsService** interface in a Java class in your project, as shown in the following example:

Sample implementation of the KieServerApplicationComponentsService interface

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

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

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

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

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

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

        return components;
    }

}
```

(1) Delivers REST endpoints to the KIE Server infrastructure that is deployed when the application starts.

(2) Specifies the extension that you are extending, such as the **Drools** extension in this example.

3 Returns all resources that the REST container must deploy. Each extension that is enabled in your KIE Server instance calls the **getAppComponents** method, so the **if (**

4 Lists the services from the specified extension that you want to use, such as the **RulesExecutionService** and **KieServerRegistry** services from the **Drools** extension in this example.

5 Specifies the transport type for the extension, either **REST** or **JMS** (**REST** in this example), and the **CustomResource** class that returns the resource as part of the **components** list.

3. Implement the **CustomResource** class that the KIE Server can use to provide the additional functionality for the new REST resource, as shown in the following example:

Sample implementation of the CustomResource class

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

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

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

    private RulesExecutionService rulesExecutionService;
    private KieServerRegistry registry;

    public CustomResource() {

    }

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

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

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

        // Marshalling behavior and supported actions:
        MarshallingFormat format = MarshallingFormat.fromType(contentType);
        if (format == null) {
            format = MarshallingFormat.valueOf(contentType);
        }
```

```
        try {
            KieContainerInstance kci = registry.getContainer(id);

            Marshaller marshaller = kci.getMarshaller(format);

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

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

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

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

            String result = marshaller.marshall(results);


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

    }
}
```

In this example, the **CustomResource** class for the custom endpoint specifies the following data and behavior:

- Uses the base endpoint **server/containers/instances/{containerId}/ksession**
- Uses **POST** HTTP method
- Expects the following data to be given in REST requests:
  - The **containerId** as a path argument
  - The **ksessionId** as a path argument
  - List of facts as a message payload
- Supports all KIE Server data formats:
  - XML (JAXB, XStream)
  - JSON

- Unmarshals the payload into a **List<?>** collection and, for each item in the list, creates an **InsertCommand** instance followed by **FireAllRules** and **GetObject** commands.
- Adds all commands to the **BatchExecutionCommand** instance that calls to the decision engine.

4. To make the new endpoint discoverable for KIE Server, create a **META-INF/services/org.kie.server.services.api.KieServerApplicationComponentsService** file in your Maven project and add the fully qualified class name of the **KieServerApplicationComponentsService** implementation class within the file. For this example, the file contains the single line **org.kie.server.ext.drools.rest.CusomtDroolsKieServerApplicationComponentsService**.

5. Build your project and copy the resulting JAR file into the **~/kie-server.war/WEB-INF/lib** directory of your project. For example, on Red Hat JBoss EAP, the path to this directory is ***EAP_HOME*/standalone/deployments/kie-server.war/WEB-INF/lib**.

6. Start the KIE Server and deploy the built project to the running KIE Server. You can deploy the project using either the Business Central interface or the KIE Server REST API (a **PUT** request to **http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}**).
After your project is deployed on a running KIE Server, you can start interacting with your new REST endpoint.

   For this example, you can use the following information to invoke the new endpoint:

   - Example request URL: **http://localhost:8080/kie-server/services/rest/server/containers/instances/demo/ksession/defaultKieSession**
   - HTTP method: **POST**
   - HTTP headers:
     - **Content-Type: application/json**
     - **Accept: application/json**
   - Example message payload:

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

   - Example server response: **200** (success)
   - Example server log output:

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

18.2. EXTENDING KIE SERVER TO USE A CUSTOM DATA TRANSPORT

By default, KIE Server extensions are exposed through REST or JMS data transports. You can extend KIE Server to support a custom data transport to adapt KIE Server transport protocols to your business needs.

As an example, this procedure adds a custom data transport to KIE Server that uses the **Drools** extension and that is based on Apache MINA, an open-source Java network-application framework. The example custom MINA transport exchanges string-based data that relies on existing marshalling operations and supports only JSON format.

Procedure

1. Create an empty Maven project and define the following packaging type and dependencies in the **pom.xml** file for the project:

   Example pom.xml file in the sample project

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.39.0.Final-redhat-00005</version.org.kie>
</properties>

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

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

2. Implement the **org.kie.server.services.api.KieServerExtension** interface in a Java class in your project, as shown in the following example:

Sample implementation of the KieServerExtension interface

```
public class MinaDroolsKieServerExtension implements KieServerExtension {

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

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

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

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

    // Specific to MINA:
    private IoAcceptor acceptor;

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

    public void init(KieServerImpl kieServer, KieServerRegistry registry) {

        KieServerExtension droolsExtension = registry.getServerExtension("Drools");
        if (droolsExtension == null) {
            logger.warn("No Drools extension available, quitting...");
            return;
```

```
        }

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

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

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

        }
    }

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

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

    }

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

    }

    public List<Object> getAppComponents(SupportedTransports type) {
        // Nothing for supported transports (REST or JMS)
        return Collections.emptyList();
```

```
    }

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

        return null;
    }

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

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

    public String getExtensionName() {
        return EXTENSION_NAME;
    }

    public Integer getStartOrder() {
        return 20;
    }

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

The **KieServerExtension** interface is the main extension interface that KIE Server can use to provide the additional functionality for the new MINA transport. The interface consists of the following components:

Overview of the KieServerExtension interface

```
public interface KieServerExtension {

    boolean isActive();

    void init(KieServerImpl kieServer, KieServerRegistry registry);

    void destroy(KieServerImpl kieServer, KieServerRegistry registry);

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

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

    List<Object> getAppComponents(SupportedTransports type);

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

    String getImplementedCapability(); (1)
```

```
    List<Object> getServices();

    String getExtensionName(); (2)

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

(1) Specifies the capability that is covered by this extension. The capability must be unique within KIE Server.

(2) Defines a human-readable name for the extension.

(3) Determines when the specified extension should be started. For extensions that have dependencies on other extensions, this setting must not conflict with the parent setting. For example, in this case, this custom extension depends on the **Drools** extension, which has **StartOrder** set to **0**, so this custom add-on extension must be greater than **0** (set to **20** in the sample implementation).

In the previous **MinaDroolsKieServerExtension** sample implementation of this interface, the **init** method is the main element for collecting services from the **Drools** extension and for bootstrapping the MINA server. All other methods in the **KieServerExtension** interface can remain with the standard implementation to fulfill interface requirements.

The **TextBasedIoHandlerAdapter** class is the handler on the MINA server that reacts to incoming requests.

3. Implement the **TextBasedIoHandlerAdapter** handler for the MINA server, as shown in the following example:

Sample implementation of the TextBasedIoHandlerAdapter handler

```
public class TextBasedIoHandlerAdapter extends IoHandlerAdapter {

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

    private KieContainerCommandService batchCommandService;

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

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

        String[] elements = completeMessage.split("\\|");
        logger.debug("Container id {}", elements[0]);
```

```
        try {
            ServiceResponse<String> result = batchCommandService.callContainer(elements[0],
elements[1], MarshallingFormat.JSON, null);

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

        }
    }
}
```

In this example, the handler class receives text messages and executes them in the **Drools** service.

Consider the following handler requirements and behavior when you use the **TextBasedIoHandlerAdapter** handler implementation:

- Anything that you submit to the handler must be a single line because each incoming transport request is a single line.
- You must pass a KIE container ID in this single line so that the handler expects the format **containerID|payload**.
- You can set a response in the way that it is produced by the marshaller. The response can be multiple lines.
- The handler supports a *stream mode* that enables you to send commands without disconnecting from a KIE Server session. To end a KIE Server session in stream mode, send either an **exit** or **quit** command to the server.

4. To make the new data transport discoverable for KIE Server, create a **META-INF/services/org.kie.server.services.api.KieServerExtension** file in your Maven project and add the fully qualified class name of the **KieServerExtension** implementation class within the file. For this example, the file contains the single line **org.kie.server.ext.mina.MinaDroolsKieServerExtension**.

5. Build your project and copy the resulting JAR file and the **mina-core-2.0.9.jar** file (which the extension depends on in this example) into the **~/kie-server.war/WEB-INF/lib** directory of your project. For example, on Red Hat JBoss EAP, the path to this directory is ***EAP_HOME*/standalone/deployments/kie-server.war/WEB-INF/lib**.

6. Start the KIE Server and deploy the built project to the running KIE Server. You can deploy the project using either the Business Central interface or the KIE Server REST API (a **PUT** request to **http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}**).
After your project is deployed on a running KIE Server, you can view the status of the new data transport in your KIE Server log and start using your new data transport:

New data transport in the server log

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

For this example, you can use Telnet to interact with the new MINA-based data transport in KIE Server:

Starting Telnet and connecting to KIE Server on port 9123 in a command terminal

```
telnet 127.0.0.1 9123
```

Example interactions with KIE Server in a command terminal

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

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

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

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

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

Example server log output

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

18.3. EXTENDING THE KIE SERVER CLIENT WITH A CUSTOM CLIENT API

KIE Server uses predefined client APIs that you can interact with to use KIE Server services. You can extend the KIE Server client with a custom client API to adapt KIE Server services to your business needs.

As an example, this procedure adds a custom client API to KIE Server to accommodate a custom data transport (configured previously for this scenario) that is based on Apache MINA, an open-source Java network-application framework.

Procedure

1. Create an empty Maven project and define the following packaging type and dependencies in the **pom.xml** file for the project:

   Example pom.xml file in the sample project

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.39.0.Final-redhat-00005</version.org.kie>
 </properties>

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

2. Implement the relevant **ServicesClient** interface in a Java class in your project, as shown in the following example:

   Sample RulesMinaServicesClient interface

```
public interface RulesMinaServicesClient extends RuleServicesClient {

}
```

A specific interface is required because you must register client implementations based on the interface, and you can have only one implementation for a given interface.

For this example, the custom MINA-based data transport uses the **Drools** extension, so this example **RulesMinaServicesClient** interface extends the existing **RuleServicesClient** client API from the **Drools** extension.

3. Implement the **RulesMinaServicesClient** interface that the KIE Server can use to provide the additional client functionality for the new MINA transport, as shown in the following example:

Sample implementation of the RulesMinaServicesClient interface

```
public class RulesMinaServicesClientImpl implements RulesMinaServicesClient {

    private String host;
    private Integer port;

    private Marshaller marshaller;

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

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

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

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

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

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

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

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

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

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

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

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

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

This example implementation specifies the following data and behavior:

- Uses socket-based communication for simplicity
- Relies on default configurations from the KIE Server client and uses **ServerUrl** for providing the host and port of the MINA server
- Specifies JSON as the marshalling format
- Requires received messages to be JSON objects that start with an open bracket **{**
- Uses direct socket communication with a blocking API while waiting for the first line of the response and then reads all lines that are available
- Does not use *stream mode* and therefore disconnects the KIE Server session after invoking a command

4. Implement the **org.kie.server.client.helper.KieServicesClientBuilder** interface in a Java class in your project, as shown in the following example:

Sample implementation of the KieServicesClientBuilder interface

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

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

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

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

        return services;
    }

}
```

(1) Enables you to provide additional client APIs to the generic KIE Server client infrastructure

(2) Defines the KIE Server capability (extension) that the client uses

(3) Provides a map of the client implementations, where the key is the interface and the value is the fully initialized implementation

5. To make the new client API discoverable for the KIE Server client, create a **META-INF/services/org.kie.server.client.helper.KieServicesClientBuilder** file in your Maven project and add the fully qualified class name of the **KieServicesClientBuilder** implementation class within the file. For this example, the file contains the single line **org.kie.server.ext.mina.client.MinaClientBuilderImpl**.

6. Build your project and copy the resulting JAR file into the **~/kie-server.war/WEB-INF/lib** directory of your project. For example, on Red Hat JBoss EAP, the path to this directory is ***EAP_HOME*/standalone/deployments/kie-server.war/WEB-INF/lib**.

7. Start the KIE Server and deploy the built project to the running KIE Server. You can deploy the project using either the Business Central interface or the KIE Server REST API (a **PUT** request to **http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}**).
After your project is deployed on a running KIE Server, you can start interacting with your new KIE Server client. You use your new client in the same way as the standard KIE Server client, by creating the client configuration and client instance, retrieving the service client by type, and invoking client methods.

   For this example, you can create a **RulesMinaServiceClient** client instance and invoke operations on KIE Server through the MINA transport:

Sample implementation to create the RulesMinaServiceClient client

```
protected RulesMinaServicesClient buildClient() {
```

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

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

    configuration.addJaxbClasses(extraClasses);

    KieServicesClient kieServicesClient =
KieServicesFactory.newKieServicesClient(configuration);

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

    return rulesClient;
}
```

Sample configuration to invoke operations on KIE Server through the MINA transport

```
RulesMinaServicesClient rulesClient = buildClient();

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

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

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

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

String data = response.getResult();

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

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

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

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

CHAPTER 19. PERFORMANCE TUNING CONSIDERATIONS WITH KIE SERVER

The following key concepts or suggested practices can help you optimize KIE Server performance. These concepts are summarized in this section as a convenience and are explained in more detail in the cross-referenced documentation, where applicable. This section will expand or change as needed with new releases of Red Hat Decision Manager.

Ensure that development mode is enabled during development

You can set KIE Server or specific projects in Business Central to use **production** mode or **development** mode. By default, KIE Server and all new projects in Business Central are in development mode. This mode provides features that facilitate your development experience, such as flexible project deployment policies, and features that optimize KIE Server performance during development, such as disabled duplicate GAV detection. Use development mode until your Red Hat Decision Manager environment is established and completely ready for production mode.
For more information about configuring the environment mode or duplicate GAV detection, see the following resources:

- Chapter 8, *Configuring the environment mode in KIE Server and Business Central*
- *Packaging and deploying a Red Hat Decision Manager project*

Adapt KIE Server capabilities and extensions to your specific needs

The capabilities in KIE Server are determined by plug-in extensions that you can enable, disable, or further extend to meet your business needs. By default, KIE Server extensions are exposed through REST or JMS data transports and use predefined client APIs. You can extend existing KIE Server capabilities with additional REST endpoints, extend supported transport methods beyond REST or JMS, or extend functionality in the KIE Server client.
This flexibility in KIE Server functionality enables you to adapt your KIE Server instances to your business needs, instead of adapting your business needs to the default KIE Server capabilities.

For information about enabling, disabling, or extending KIE Server capabilities, see Chapter 18, *KIE Server capabilities and extensions*.

CHAPTER 20. ADDITIONAL RESOURCES

- *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.3*
- *Planning a Red Hat Decision Manager installation*
- *Installing and configuring Red Hat Decision Manager on Red Hat JBoss EAP 7.3*
- *Deploying a Red Hat Decision Manager immutable server environment on Red Hat OpenShift Container Platform*
- *Deploying a Red Hat Decision Manager authoring or managed server environment on Red Hat OpenShift Container Platform*
- *Deploying a Red Hat Decision Manager environment on Red Hat OpenShift Container Platform using Operators*

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Wednesday, August 12, 2020.