Red Hat Process Automation Manager 7.6

Managing and monitoring Process Server

Last Updated: 2021-06-08

Red Hat Process Automation Manager 7.6 Managing and monitoring Process Server

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

Legal Notice

Abstract

This document explains how install, configure, and performance tune Red Hat Process Automation Manager 7.6.

Table of Contents

PREFACE

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

Prerequisites

- Red Hat JBoss Enterprise Application Platform 7.2 is installed. For more information, see *Red Hat JBoss Enterprise Application Platform 7.2 Installation Guide*.
- Red Hat Process Automation Manager is installed. For more information, see *Planning a Red Hat Process Automation Manager installation*.
- Red Hat Process Automation Manager is running and you can log in to Business Central with the **admin** role. For more information, see *Planning a Red Hat Process Automation Manager installation*.

CHAPTER 1. RED HAT PROCESS AUTOMATION MANAGER COMPONENTS

Red Hat Process Automation Manager is made up of Business Central and Process 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.

- Process 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 Process 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 Process Server to run in managed or unmanaged mode. If Process Server is unmanaged, you must manually create and maintain KIE containers (deployment units). A KIE container is a specific version of a project. If Process Server is managed, the Process Automation Manager controller manages the Process Server configuration and you interact with the Process Automation Manager controller to create and maintain KIE containers.

CHAPTER 2. SYSTEM INTEGRATION WITH MAVEN

Red Hat Process Automation 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 Process Automation 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

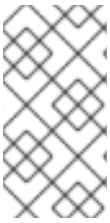
To disable this feature, set the **org.guvnor.project.gav.check.disabled** system property to **true** for Business Central at system startup:

```
$ ~/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 PROCESS AUTOMATION MANAGER

Automated update tools are often provided with both patch updates and new minor versions of Red Hat Process Automation Manager to facilitate updating certain components of Red Hat Process Automation Manager, such as Business Central, Process Server, and the headless Process Automation Manager controller. Other Red Hat Process Automation 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 Process Automation Manager 7.6. Patch updates of Red Hat Process Automation Manager, such as an update from version 7.6 to 7.6.1, include the latest security updates and bug fixes. Minor release upgrades of Red Hat Process Automation Manager, such as an upgrade from version 7.5.x to 7.6, include enhancements, security updates, and bug fixes.



NOTE

Only updates for Red Hat Process Automation Manager are included in Red Hat Process Automation 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 Process Automation Manager and Process Server instances are not running. Do not apply updates while you are running an instance of Red Hat Process Automation Manager or Process 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.
   Example:

   - **Product:** Process Automation Manager
   - **Version:** 7.6.1

   If you are upgrading to a new minor release of Red Hat Process Automation Manager, such as an upgrade from version 7.5.x to 7.6, first apply the latest patch update to your current version of Red Hat Process Automation Manager and then follow this procedure again to upgrade to the new minor release.

2. Click **Patches**, download the **Red Hat Process Automation Manager [VERSION] Update Tool** and extract the downloaded **rhpam-$VERSION-update.zip** file to a temporary directory.
   This update tool automates the update of certain components of Red Hat Process Automation Manager, such as Business Central, Process Server, and the headless Process Automation 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 Process Automation Manager distribution.

3. If you want to preserve any files from being updated by the update tool, navigate to the extracted **rhpam-$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:

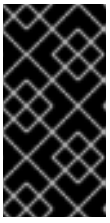
```
WEB-INF/web.xml  // Custom file
styles/base.css  // Obsolete custom file kept for record
```

   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 **rhpam-$VERSION-update.zip** file and run the **apply-updates** script in the following format:



IMPORTANT

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

   On Linux or Unix-based systems:

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

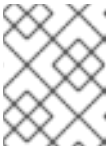
   On Windows:

```
$ .\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 Process Automation Manager update tool:

   - **rhpam-business-central-eap7-deployable**: Updates Business Central (**business-central.war**)
   - **rhpam-kie-server-ee8**: Updates Process Server (**kie-server.war**)



NOTE

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

- **rhpam-controller-ee7**: Updates the headless Process Automation Manager controller (**controller.war**)

Example update to Business Central and Process Server for a full Red Hat Process Automation Manager distribution on Red Hat JBoss EAP:

```
./apply-updates.sh ~EAP_HOME/standalone/deployments/business-central.war rhpam-business-central-eap7-deployable

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

Example update to headless Process Automation Manager controller, if used:

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

The update script creates a **backup** folder in the extracted **rhpam-$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 Process Automation Manager distribution. For files that already exist in your Red Hat Process Automation 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 Process Automation Manager 7.6.0 Maven Repository** artifact (**rhpam-7.6.0-maven-repository.zip**), such as in air-gap environments, download **Red Hat Process Automation Manager 7.6.x Maven Repository** and extract the downloaded **rhpam-7.6.x-maven-repository.zip** file to your existing **~/maven-repository** directory to update the relevant contents.
Example Maven repository update:

```
$ unzip -o rhpam-7.6.1-maven-repository.zip 'rhba-7.6.1.GA-maven-repository/maven-repository/*' -d /tmp/rhbaMavenRepoUpdate

$ mv /tmp/rhbaMavenRepoUpdate/rhba-7.6.1.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 Process Automation Manager and Process 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 **rhpam-$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 Process Automation 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 PROCESS SERVER

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

Procedure

Navigate to the Red Hat Process Automation Manager 7.6 **bin** directory and start the new Process 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=process_server_username (3)
-Dorg.kie.server.pwd=process_server_password (4)
-Dorg.kie.server.controller=http://localhost:8080/business-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)
-Dorg.kie.server.persistence.dialect=org.hibernate.dialect.PostgreSQLDialect (9)
-Dorg.kie.server.persistence.ds=java:jboss/datasources/psjbpmDS (10)
```

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 Process Server from the Process Automation Manager controller
4. Password to connect with Process Server from the Process Automation Manager controller
5. Process Automation Manager controller location, Business Central URL with **/rest/controller** suffix
6. User name to connect to the Process Automation Manager controller REST API
7. Password to connect to the Process Automation Manager controller REST API
8. Process Server location (on the same instance as Business Central in this example)
9. Hibernate dialect to be used
10. JNDI name of the data source used for your previous Red Hat JBoss BPM Suite database

NOTE

If Business Central and Process Server are installed on separate application server instances (Red Hat JBoss EAP or other), use a separate port for the Process Server location to avoid port conflicts with Business Central. If a separate Process Server port has not already been configured, you can add a port offset and adjust the Process Server port value accordingly in the Process 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 Process Server port, with a defined offset of 150, is 8230.

Process 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 Process Server from Process 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 Process Automation 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.**
```

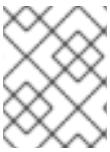
It is expected that you do not configure the classes that exists in KJAR using these system property. Ensure that only known classes are used in the system property to avoid any vulnerabilities.

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 JDBC DATA SOURCES FOR PROCESS SERVER

A data source is an object that enables a Java Database Connectivity (JDBC) client, such as an application server, to establish a connection with a database. Applications look up the data source on the Java Naming and Directory Interface (JNDI) tree or in the local application context and request a database connection to retrieve data. You must configure data sources for Process Server to ensure proper data exchange between the servers and the designated database.



NOTE

For production environments, specify an actual data source. Do not use the example data source in production environments.

Prerequisites

- The JDBC providers that you want to use to create database connections are configured on all servers on which you want to deploy Process Server, as described in the "Creating Datasources" and "JDBC Drivers" sections of the *Red Hat JBoss Enterprise Application Server Configuration Guide*.
- The Red Hat Process Automation Manager 7.6.0 Add Ons (rhpam-7.6.0-add-ons.zip) file is downloaded from the Software Downloads page in the Red Hat Customer Portal.

Procedure

1. Complete the following steps to prepare your database:
   a. Extract **rhpam-7.6.0-add-ons.zip** in a temporary directory, for example **TEMP_DIR**.
   b. Extract **TEMP_DIR/rhpam-7.6.0-migration-tool.zip**.
   c. Change your current directory to the **TEMP_DIR/rhpam-7.6.0-migration-tool/ddl-scripts** directory. This directory contains DDL scripts for several database types.
   d. Import the DDL script for your database type into the database that you want to use, for example:

```
psql jbpm < /ddl-scripts/postgresql/postgresql-jbpm-schema.sql
```

2. Open ***EAP_HOME*/standalone/configuration/standalone-full.xml** in a text editor and locate the **<system-properties>** tag.
3. Add the following properties to the **<system-properties>** tag where **<DATASOURCE>** is the JNDI name of your data source and **<HIBERNATE_DIALECT>** is the hibernate dialect for your database.

NOTE

The default value of the **org.kie.server.persistence.ds** property is **java:jboss/datasources/ExampleDS**. The default value of the **org.kie.server.persistence.dialect** property is **org.hibernate.dialect.H2Dialect**.

```
<property name="org.kie.server.persistence.ds" value="<DATASOURCE>"/>
<property name="org.kie.server.persistence.dialect" value="<HIBERNATE_DIALECT>"/>
```

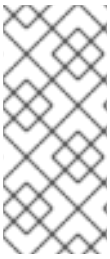
The following example shows how to configure a datasource for the PostgreSQL hibernate dialect:

```
<system-properties>
  <property name="org.kie.server.repo" value="${jboss.server.data.dir}"/>
  <property name="org.kie.example" value="true"/>
  <property name="org.jbpm.designer.perspective" value="full"/>
  <property name="designerdataobjects" value="false"/>
  <property name="org.kie.server.user" value="rhpamUser"/>
  <property name="org.kie.server.pwd" value="rhpam123!"/>
  <property name="org.kie.server.location" value="http://localhost:8080/kie-
server/services/rest/server"/>
  <property name="org.kie.server.controller" value="http://localhost:8080/business-
central/rest/controller"/>
  <property name="org.kie.server.controller.user" value="kieserver"/>
  <property name="org.kie.server.controller.pwd" value="kieserver1!"/>
  <property name="org.kie.server.id" value="local-server-123"/>

  <!-- Data source properties. -->
  <property name="org.kie.server.persistence.ds"
value="java:jboss/datasources/KieServerDS"/>
  <property name="org.kie.server.persistence.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/>
</system-properties>
```

The following dialects are supported:

- DB2: **org.hibernate.dialect.DB2Dialect**
- MSSQL: **org.hibernate.dialect.SQLServer2012Dialect**
- MySQL: **org.hibernate.dialect.MySQL5InnoDBDialect**
- MariaDB: **org.hibernate.dialect.MySQL5InnoDBDialect**
- Oracle: **org.hibernate.dialect.Oracle10gDialect**
- PostgreSQL: **org.hibernate.dialect.PostgreSQL82Dialect**
- PostgreSQL plus: **org.hibernate.dialect.PostgresPlusDialect**
- Sybase: **org.hibernate.dialect.SybaseASE157Dialect**

CHAPTER 6. CONFIGURING PROCESS SERVER WITH THE INTEGRATED PROCESS AUTOMATION MANAGER CONTROLLER



NOTE

Only make the changes described in this section if Process Server will be managed by Business Central and you installed Red Hat Process Automation Manager from the ZIP files. If you did not install Business Central, you can use the headless Process Automation Manager controller to manage Process Server, as described in Chapter 7, *Installing and running the headless Process Automation Manager controller*.

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

The Process Automation 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 Process Automation Manager controller.

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

Prerequisites

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

NOTE

You should install Business Central and Process Server on different servers in production environments. However, if you install Process 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 Process 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 Process Server installation, for example, **rhpam-7.6.0-process_server-1**.
- Replace **<HOST>** with the ID or name of the Process Server host, for example, **localhost** or **192.7.8.9**.
- Replace **<PORT>** with the port of the Process Server host, for example, **8080**.

NOTE

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

- Replace **<CONTROLLER_URL>** with the URL of Business Central. Process 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>/business-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 7. INSTALLING AND RUNNING THE HEADLESS PROCESS AUTOMATION MANAGER CONTROLLER

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

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

7.1. USING THE INSTALLER TO CONFIGURE PROCESS SERVER WITH THE PROCESS AUTOMATION MANAGER CONTROLLER

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

The Process Automation 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 Process Automation Manager controller.

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

NOTE

If you do not install Business Central, see Chapter 7, *Installing and running the headless Process Automation Manager controller* for information about using the headless Process Automation Manager controller.

Prerequisites

- Two computers with backed-up Red Hat JBoss EAP 7.2 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 Process Automation Manager on Red Hat JBoss EAP 7.2* for more information.
2. On the **Component Selection** page, clear the **Process 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 Process Server properties** and click **Next**.
8. Enter the controller URL for Business Central and configure additional properties for Process 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 Process Automation Manager controller is now integrated with Business Central, go to the **Execution Servers** page in Business Central and confirm that the Process Server that you configured appears under **REMOTE SERVERS**.

7.2. INSTALLING THE HEADLESS PROCESS AUTOMATION MANAGER CONTROLLER

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

Prerequisites

- A backed-up Red Hat JBoss EAP installation version 7.2 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:** Process Automation Manager
   - **Version:** 7.6
2. Download **Red Hat Process Automation Manager 7.6.0 Add Ons**(the **rhpam-7.6.0-add-ons.zip** file).
3. Unzip the **rhpam-7.6.0-add-ons.zip** file. The **rhpam-7.6.0-controller-ee7.zip** file is in the unzipped directory.
4. Extract the **rhpam-7.6.0-controller-ee7** archive to a temporary directory. In the following examples this directory is called ***TEMP_DIR***.
5. Copy the ***TEMP_DIR*/rhpam-7.6.0-controller-ee7/controller.war** directory to ***EAP_HOME*/standalone/deployments/**.

WARNING

Ensure that the names of the headless Process Automation 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*/rhpam-7.6.0-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 Process Automation Manager controller is automatically deployed when the server starts.

7.2.1. Creating a headless Process Automation Manager controller user

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

Prerequisites

- The headless Process Automation 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 <USER_NAME> --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.

7.2.2. Configuring Process Server and the headless Process Automation Manager controller

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

Prerequisites

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

NOTE

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

- On Process 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 Process 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 Process Server installation, for example, **rhpam-7.6.0-process_server-1**.
- Replace **<HOST>** with the ID or name of the Process Server host, for example, **localhost** or **192.7.8.9**.
- Replace **<PORT>** with the port of the Process Server host, for example, **8080**.

NOTE

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

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

    1. Process Server connects to this URL during startup.

7.3. RUNNING THE HEADLESS PROCESS AUTOMATION MANAGER CONTROLLER

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

Prerequisites

- The headless Process Automation 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 Process Automation Manager controller on the same Red Hat JBoss EAP instance as the Red Hat JBoss EAP instance where you installed the Process 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 Process Automation Manager controller on a separate Red Hat JBoss EAP instance from the Red Hat JBoss EAP instance where you installed the Process Server, you can start the headless Process Automation 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 Process Automation 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 Process 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 Process Server Java API Client to access the headless Process Automation Manager controller.

7.4. CLUSTERING PROCESS SERVERS WITH THE HEADLESS PROCESS AUTOMATION MANAGER CONTROLLER

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

Prerequisites

- A backed-up Red Hat JBoss EAP installation version 7.2 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 Process Automation 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: Process Automation Manager**
   - **Version: 7.6**
2. Download **Red Hat Process Automation Manager 7.6.0 Add Ons**(the **rhpam-7.6.0-add-ons.zip** file).
3. Unzip the **rhpam-7.6.0-add-ons.zip** file. The **rhpam-7.6.0-controller-ee7.zip** file is in the unzipped directory.
4. Extract the **rhpam-7.6.0-controller-ee7** archive to a temporary directory. In the following examples this directory is called ***TEMP_DIR***.
5. Copy the ***TEMP_DIR*/rhpam-7.6.0-controller-ee7/controller.war** directory to ***EAP_HOME*/standalone/deployments/**.

WARNING

Ensure that the names of the headless Process Automation 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*/rhpam-7.6.0-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 Process Automation 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 Process Automation 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 Process Automation Manager in a Red Hat JBoss Enterprise Application Platform clustered environment, see *Installing and configuring Red Hat Process Automation Manager in a Red Hat JBoss EAP clustered environment*.

CHAPTER 8. CONFIGURING A PROCESS SERVER TO CONNECT TO BUSINESS CENTRAL

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

NOTE

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

Prerequisites

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

Procedure

1. In your Red Hat Process Automation Manager installation directory, navigate to the **standalone-full.xml** file. For example, if you use a Red Hat JBoss EAP installation for Red Hat Process Automation 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>/business-central/rest/controller**, where **<centralhost>** and **<centralport>** are the host name and port for Business Central. If Business Central is deployed on OpenShift, remove **business-central/** from the URL.

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

CHAPTER 9. CONFIGURING THE ENVIRONMENT MODE IN PROCESS SERVER AND BUSINESS CENTRAL

You can set Process 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 Process Server without stopping any running instances (if applicable), or click **Redeploy** to deploy the built KJAR file and replace all instances. The next time you deploy or redeploy the built KJAR, the previous deployment unit (KIE container) is automatically updated in the same target Process Server.

In a production environment, the **Redeploy** option 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 Process Server.

Procedure

1. To configure the Process Server environment mode, set the **org.kie.server.mode** system property to **org.kie.server.mode=development** or **org.kie.server.mode=production**.

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, Process Server and all new projects in Business Central are in development mode.

You cannot deploy a project with **Development Mode** turned on or with a manually added **SNAPSHOT** version suffix to a Process Server that is in production mode.

CHAPTER 10. CONFIGURING PROCESS 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 Process Server instance.

Prerequisites

- Users with the following roles exist:
  - In Business Central, a user with the role **rest-all**
  - On the Process 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 Process 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 10.1. JVM Properties for Process Server Instance

Property	Value	Note
org.kie.server.id	**default-kie-server**	The Process Server ID.
org.kie.server.controller	**http://localhost:8080/business-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 Process Server.

Table 10.2. 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 Process 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.

10.1. CONFIGURING SMART ROUTER FOR TLS SUPPORT

You can now configure Smart Router (previously, KIE Server Router) for TLS support to allow HTTPS traffic.

Procedure

- Open a terminal and enter the following command to start the smart router with TLS support:

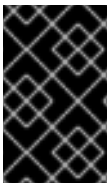
```
java -Dorg.kie.server.router.tls.keystore=PATH_TO_YOUR_KEYSTORE
    -Dorg.kie.server.router.tls.keystore.password=YOUR_KEYSTORE_PASSWD
    -Dorg.kie.server.router.tls.keystore.keyalias=YOUR_KEYSTORE_ALIAS
    -jar kie-server-router-proxy-YOUR_VERSION.jar
```

Replace **PATH_TO_YOUR_KEYSTORE**, **YOUR_KEYSTORE_PASSWD**, **YOUR_KEYSTORE_ALIAS**, and **YOUR_VERSION** with the relevant data.

CHAPTER 11. MANAGED PROCESS SERVER

A managed instance requires an available Process Automation Manager controller to start the Process Server.

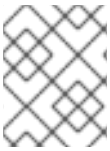
A Process Automation Manager controller manages the Process Server configuration in a centralized way. Each Process Automation Manager controller can manage multiple configurations at once, and there can be multiple Process Automation Manager controllers in the environment. Managed Process Server can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



IMPORTANT

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

When the Process Server is configured with a list of Process Automation 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 Process Server in standalone mode without connecting to Process Automation Manager controllers, see Chapter 12, *Unmanaged Process Server*.

CHAPTER 12. UNMANAGED PROCESS SERVER

An unmanaged Process Server is a standalone instance, and therefore must be configured individually using REST/JMS API from the Process 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 Process 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 13. ACTIVATING OR DEACTIVATING A KIE CONTAINER ON PROCESS SERVER

You can now stop the creation of new process instances from a given container by deactivating it but at the same time continue working on its existing process instances and tasks. In case the deactivation is temporary, you can activate the container again later. The activation or deactivation of KIE containers do not require restarting of KIE server.

Prerequisites

- A KIE container has been created and configured in Business Central.

Procedure

1. Log in to Business Central.
2. In the main menu, click **Menu → Deploy → Execution Servers**.
3. From the **Server Configurations** pane, which is on the left of the page, select your server.
4. From the **Deployment Units** pane, select the deployment unit you want to activate or deactivate.
5. Click **Activate** or **Deactivate** in the upper-right corner of the deployment unit pane.
   You cannot create a process instance from a KIE container once it is deactivated.

CHAPTER 14. 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 Business Central 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 Process 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 Process 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
```

14.1. 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 14.1. 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 Process 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.

14.2. 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.

14.3. 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 15. ACCESSING RUNTIME DATA FROM BUSINESS CENTRAL

The following pages in Business Central allow you to view the runtime data of the Process Server:

- **Process Reports**
- **Task Reports**
- **Process Definitions**
- **Process Instances**
- **Execution Errors**
- **Jobs**
- **Tasks**

These pages use the credentials of the currently logged in user to load data from the Process Server. Therefore, to be able to view the runtime data in Business Central, ensure that the following conditions are met:

- The user exists in the KIE container (deployment unit) running the Business Central application. This user must have **admin**, **analyst**, or **developer** roles assigned, in addition to the **kie-server** role, with full access to the runtime data. The **manager** and **process_admin** roles also allow access to runtime data pages in Business Central.
- The user exists in the KIE container (deployment unit) running the Process Server and has **kie-server** role assigned.
- Communication between Business Central and the Process Server is established. That is, the Process Server is registered in the Process Automation Manager controller, which is part of Business Central.
- The **deployment.business-central.war** login module is present in the **standalone.xml** configuration of the server running Business Central:

```
<login-module code="org.kie.security.jaas.KieLoginModule" flag="optional"
module="deployment.business-central.war"/>
```

CHAPTER 16. EXECUTION ERROR MANAGEMENT

When an execution error occurs for a business process, the process stops and reverts to the most recent stable state (the closest safe point) and continues its execution. If an error of any kind is not handled by the process the entire transaction rolls back, leaving the process instance in the previous wait state. Any trace of this is only visible in the logs, and usually displayed to the caller who sent the request to the process engine.

Users with process administrator (**process-admin**) or administrator (**admin**) roles are able to access error messages in Business Central. Execution error messaging provides the following primary benefits:

- Better traceability
- Visibility in case of critical processes
- Reporting and analytics based on error situations
- External system error handling and compensation

Configurable error handling is responsible for receiving any technical errors thrown throughout the process engine execution (including task service). The following technical exceptions apply:

- Anything that extends **java.lang.Throwable**
- Process level error handling and any other exceptions not previously handled

There are several components that make up the error handling mechanism and allow a pluggable approach to extend its capabilities.

The process engine entry point for error handling is the **ExecutionErrorManager**. This is integrated with **RuntimeManager**, which is then responsible for providing it to the underlying **KieSession** and **TaskService**.

From an API point of view, **ExecutionErrorManager** provides access to the following components:

- **ExecutionErrorHandler**: The primary mechanism for error handling
- **ExecutionErrorStorage**: Pluggable storage for execution error information

16.1. MANAGING EXECUTION ERRORS

By definition, every process error that is detected and stored is unacknowledged and must be handled by someone or something (in case of automatic error recovery). Errors are filtered on the basis of whether or not they have been acknowledged. Acknowledging an error saves the user information and time stamp for traceability. You can access the **Error Management** view at any time.

Procedure

1. In Business Central, go to **Menu → Manage → Execution Errors**.
2. Select an error from the list to open the **Details** tab. This displays information about the error or errors.
3. Click the **Acknowledge** button to acknowledge and clear the error. You can view the error later by selecting **Yes** on the **Acknowledged** filter in the **Manage Execution Errors** page.
   If the error was related to a task, a **Go to Task** button is displayed.

4. Click the **Go to Task** button, if applicable, to view the associated job information in the **Manage Tasks** page.
   In the **Manage Tasks** page, you can restart, reschedule, or retry the corresponding task.

16.2. THE EXECUTIONERRORHANDLER

The **ExecutionErrorHandler** is the primary mechanism for all process error handling. It is bound to the life cycle of RuntimeEngine; meaning it is created when a new runtime engine is created, and is destroyed when RuntimeEngine is disposed. A single instance of the **ExecutionErrorHandler** is used within a given execution context or transaction. Both **KieSession** and **TaskService** use that instance to inform the error handling about processed nodes/tasks. **ExecutionErrorHandler** is informed about:

- Starting of processing of a given node instance.
- Completion of processing of a given node instance.
- Starting of processing of a given task instance.
- Completion of processing of a given task instance.

This information is mainly used for errors that are of unknown type; that is, errors that do not provide information about the process context. For example, upon commit time, database exceptions do not carry any process information.

16.3. EXECUTION ERROR STORAGE

ExecutionErrorStorage is a pluggable strategy that permits various ways of persisting information about execution errors. Storage is used directly by the handler that gets an instance of the store when it is created (when RuntimeEngine is created). Default storage implementation is based on the database table, which stores every error and includes all of the available information. Some errors may not contain details, as this depends on the type of error and whether or not it is possible to extract specific information.

16.4. ERROR TYPES AND FILTERS

Error handling attempts to catch and handle any kind of error, therefore it needs a way to categorize errors. By doing this, it is able to properly extract information from the error and make it pluggable, as some users may require specific types of errors to be thrown and handled in different ways than what is provided by default.

Error categorization and filtering is based on **ExecutionErrorFilters**. This interface is solely responsible for building instances of **ExecutionError**, which are later stored by way of the **ExecutionErrorStorage** strategy. It has following methods:

- **accept**: indicates if given error can be handled by the filter.
- **filter**: where the actual filtering, handling, and so on happens.
- **getPriority**: indicates the priority that is used when calling filters.

Filters process one error at a time and use a priority system to avoid having multiple filters returning alternative “views” of the same error. Priority enables more specialized filters to see if the error can be accepted, or otherwise allow another filter to handle it.

ExecutionErrorFilter can be provided using the **ServiceLoader** mechanism, which enables the capability of error handling to be easily extended.

Red Hat Process Automation Manager ships with the following **ExecutionErrorFilters** :

Table 16.1. ExecutionErrorFilters

Class name	Type	Priority
org.jbpm.runtime.manager.impl.error.filters.ProcessExecutionErrorFilter	Process	100
org.jbpm.runtime.manager.impl.error.filters.TaskExecutionErrorFilter	Task	80
org.jbpm.runtime.manager.impl.error.filters.DBExecutionErrorFilter	DB	200
org.jbpm.executor.impl.error.JobExecutionErrorFilter	Job	100

Filters are given a higher execution order based on the lowest value of the priority. In the above table, filters are invoked in the following order:

1. Task
2. Process
3. Job
4. DB

16.5. AUTO ACKNOWLEDGING EXECUTION ERRORS

When executions errors occur they are unacknowledged by default, and require a manual acknowledgment to be performed otherwise they are always seen as information that requires attention. In case of larger volumes, manual actions can be time consuming and not suitable in some situations.

Auto acknowledgment resolves this issue. It is based on scheduled jobs by way of the **jbpm-executor**, with the following three types of jobs available:

org.jbpm.executor.commands.error.JobAutoAckErrorCommand

Responsible for finding jobs that previously failed but now are either canceled, completed, or rescheduled for another execution. This job only acknowledges execution errors of type **Job**.

org.jbpm.executor.commands.error.TaskAutoAckErrorCommand

Responsible for auto acknowledgment of user task execution errors for tasks that previously failed but now are in one of the exit states (completed, failed, exited, obsolete). This job only acknowledges execution errors of type **Task**.

org.jbpm.executor.commands.error.ProcessAutoAckErrorCommand

Responsible for auto acknowledgment of process instances that have errors attached. It acknowledges errors where the process instance is already finished (completed or aborted), or the task that the error originated from is already finished. This is based on **init_activity_id** value. This job acknowledges any type of execution error that matches the above criteria.

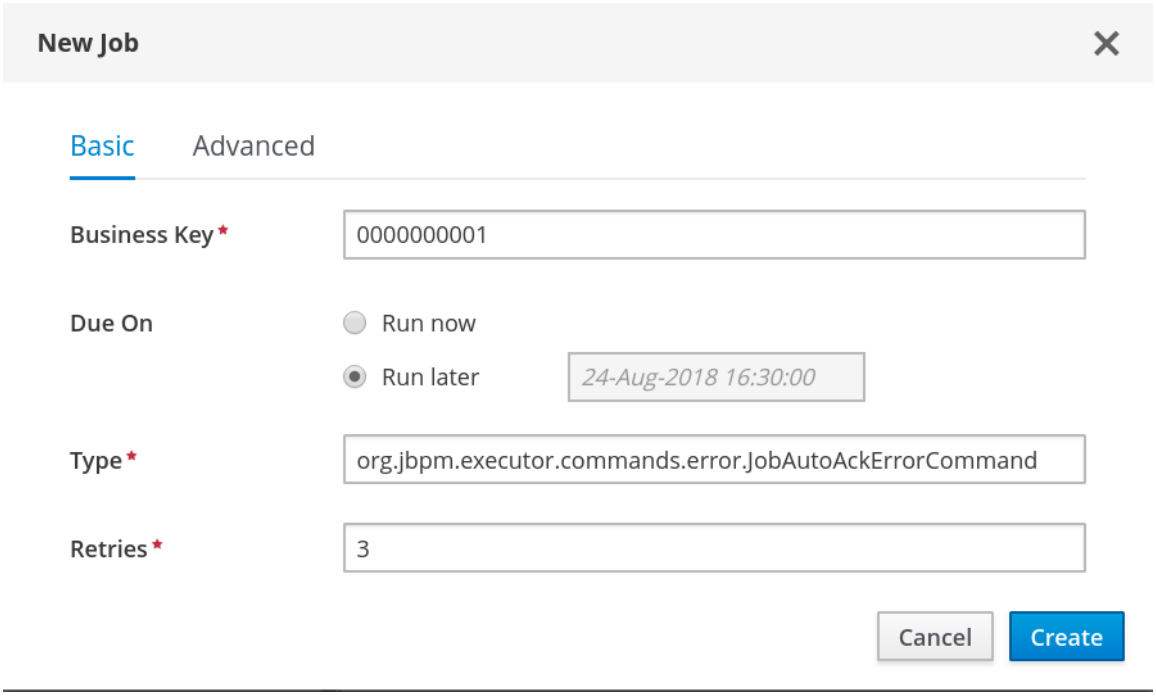
Jobs can be registered on the Process Server. In Business Central you can configure auto acknowledge jobs for errors:

Prerequisites

- Execution errors of one or more type have accumulated during processes execution but require no further attention.

Procedure

1. In Business Central, click **Menu → Manage → Jobs**.
2. In the top right of the screen, click **New Job**.
3. Type the process correlation key into the **Business Key** field.
4. In the **Type** field, add type of the auto acknowledge job type from the list above.
5. Select a **Due On** time for the job to be completed:
   a. To run the job immediately, select the **Run now** option.
   b. To run the job at a specific time, select **Run later**. A date and time field appears next to the **Run later** option. Click the field to open the calendar and schedule a specific time and date for the job.



6. Click **Create** to create the job and return to the **Manage Jobs** page.

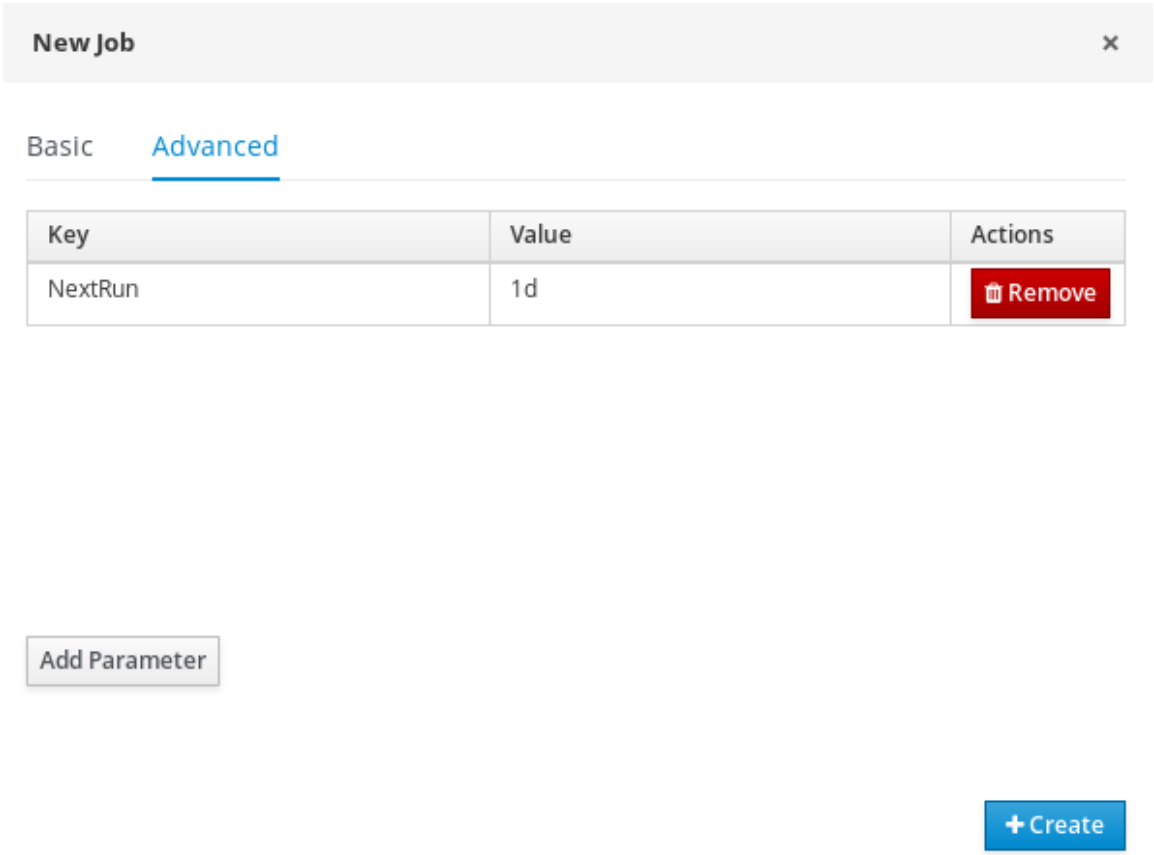
The following steps are optional, and allow you to configure auto acknowledge jobs to run either once (**SingleRun**), on specific time intervals (**NextRun**), or using the custom name of an entity manager factory to search for jobs to acknowledge (**EmfName**).

1. Click the **Advanced** tab.
2. Click the **Add Parameter** button.
3. Enter the configuration parameter you want to apply to the job:

a. **SingleRun**: **true** or **false**

b. **NextRun**: time expression, such as 2h, 5d, 1m, and so on.

c. **EmfName**: custom entity manager factory name.



16.6. CLEANING UP THE ERROR LIST

The **ExecutionErrorInfo** error list table can be cleaned up to remove redundant information. Depending on the life cycle of the process, errors may remain in the list for some time, and there is no direct API with which to clean up the list. Instead, the **ExecutionErrorCleanupCommand** command can be scheduled to periodically clean up errors.

The following parameters can be set for the clean up command. The command is restricted to deleting execution errors of already completed or aborted process instances:

- **DateFormat**
  - Date format for further date related parameters - if not given **yyyy-MM-dd** is used (pattern of **SimpleDateFormat** class).
- **EmfName**
  - Name of the entity manager factory to be used for queries (valid persistence unit name).
- **SingleRun**
  - Indicates if execution should be single run only (**true**|**false**).
- **NextRun**
  - Provides next execution time (valid time expression, for example: 1d, 5h, and so on)

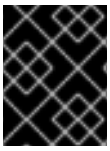
- **OlderThan**
  - Indicates what errors should be deleted - older than given date.
- **OlderThanPeriod**
  - Indicated what errors should be deleted older than given time expression (valid time expression e.g. 1d, 5h, and so on)
- **ForProcess**
  - Indicates errors to be deleted only for given process definition.
- **ForProcessInstance**
  - Indicates errors to be deleted only for given process instance.
- **ForDeployment**
  - Indicates errors to be deleted that are from given deployment ID.

CHAPTER 17. PROMETHEUS METRICS MONITORING IN RED HAT PROCESS AUTOMATION MANAGER

Prometheus is an open-source systems monitoring toolkit that you can use with Red Hat Process Automation Manager to collect and store metrics related to the execution of business rules, processes, Decision Model and Notation (DMN) models, and other Red Hat Process Automation Manager assets. You can access the stored metrics through a REST API call to the Process 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 Process Server instance, for Process Server on Spring Boot, or for a Process Server deployment on Red Hat OpenShift Container Platform.

For the list of available metrics that Process Server exposes with Prometheus, download the **Red Hat Process Automation Manager 7.6.0 Source Distribution** from the Red Hat Customer Portal and navigate to **~/rhpam-7.6.0-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.

17.1. CONFIGURING PROMETHEUS METRICS MONITORING FOR PROCESS SERVER

You can configure your Process Server instances to use Prometheus to collect and store metrics related to your business asset activity in Red Hat Process Automation Manager. For the list of available metrics that Process Server exposes with Prometheus, download the **Red Hat Process Automation Manager 7.6.0 Source Distribution** from the Red Hat Customer Portal and navigate to **~/rhpam-7.6.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus**.

Prerequisites

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

Procedure

1. In your Process 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 Process Server or in the **standalone.xml** or **standalone-full.xml** file of Red Hat Process Automation Manager distribution.
2. If you are running Red Hat Process Automation Manager on Spring Boot, configure the required key in the **application.properties** system property:

Spring Boot application.properties key for Red Hat Process Automation Manager

and Prometheus

```
kieserver.jbpm.enabled=true
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 Process 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 Process Server location and settings.

4. Start the Process Server instance.

Example start command for Red Hat Process Automation Manager on Red Hat JBoss EAP

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

After you start the configured Process Server instance, Prometheus begins collecting metrics and Process 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 Process Server is publishing the metrics:
For REST client:

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

- **HTTP method**: Set to **GET**.
- **URL**: Enter the Process 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 Process Server user with the **kie-server** role.
- **-H**: Set the following header:
  - **accept**: **application/json**
- **-X**: Set to **GET**.
- **URL**: Enter the Process 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 Process Automation 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 Process Automation 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
# HELP kie_server_case_started_total Kie Server Started Cases
# TYPE kie_server_case_started_total counter
kie_server_case_started_total{case_definition_id="itorders.orderhardware",} 1.0
# HELP kie_server_case_running_total Kie Server Running Cases
# TYPE kie_server_case_running_total gauge
kie_server_case_running_total{case_definition_id="itorders.orderhardware",} 2.0
# HELP kie_server_data_set_registered_total Kie Server Data Set Registered
# TYPE kie_server_data_set_registered_total gauge
kie_server_data_set_registered_total{name="jbpmProcessInstanceLogs::CUSTOM",uuid="jbp
mProcessInstanceLogs",} 1.0
kie_server_data_set_registered_total{name="jbpmRequestList::CUSTOM",uuid="jbpmRequest
List",} 1.0
kie_server_data_set_registered_total{name="tasksMonitoring::CUSTOM",uuid="tasksMonitorin
g",} 1.0
kie_server_data_set_registered_total{name="jbpmHumanTasks::CUSTOM",uuid="jbpmHuman
Tasks",} 1.0
kie_server_data_set_registered_total{name="jbpmHumanTasksWithUser::FILTERED_PO_TA
SK",uuid="jbpmHumanTasksWithUser",} 1.0
kie_server_data_set_registered_total{name="jbpmHumanTasksWithVariables::CUSTOM",uuid
="jbpmHumanTasksWithVariables",} 1.0
kie_server_data_set_registered_total{name="jbpmProcessInstancesWithVariables::CUSTOM",
uuid="jbpmProcessInstancesWithVariables",} 1.0
kie_server_data_set_registered_total{name="jbpmProcessInstances::CUSTOM",uuid="jbpmPr
ocessInstances",} 1.0
kie_server_data_set_registered_total{name="jbpmExecutionErrorList::CUSTOM",uuid="jbpmEx
ecutionErrorList",} 1.0
kie_server_data_set_registered_total{name="processesMonitoring::CUSTOM",uuid="processe
sMonitoring",} 1.0
kie_server_data_set_registered_total{name="jbpmHumanTasksWithAdmin::FILTERED_BA_TA
SK",uuid="jbpmHumanTasksWithAdmin",} 1.0
# HELP kie_server_execution_error_total Kie Server Execution Errors
# TYPE kie_server_execution_error_total counter
# HELP kie_server_task_completed_total Kie Server Completed Tasks
# TYPE kie_server_task_completed_total counter
# HELP kie_server_container_running_total Kie Server Running Containers
# TYPE kie_server_container_running_total gauge
kie_server_container_running_total{container_id="itorders_1.0.0-SNAPSHOT",} 1.0
# HELP kie_server_job_cancelled_total Kie Server Cancelled Jobs
# TYPE kie_server_job_cancelled_total counter
# HELP kie_server_process_instance_started_total Kie Server Started Process Instances
# TYPE kie_server_process_instance_started_total counter
kie_server_process_instance_started_total{container_id="itorders_1.0.0-
SNAPSHOT",process_id="itorders.orderhardware",} 1.0
# HELP solver_duration_seconds Time in seconds it took solver to solve the constraint
problem
```

```
# TYPE solver_duration_seconds summary
# HELP kie_server_task_skipped_total Kie Server Skipped Tasks
# TYPE kie_server_task_skipped_total counter
# HELP kie_server_data_set_execution_time_seconds Kie Server Data Set Execution Time
# TYPE kie_server_data_set_execution_time_seconds summary
kie_server_data_set_execution_time_seconds_count{uuid="jbpmProcessInstances",} 8.0
kie_server_data_set_execution_time_seconds_sum{uuid="jbpmProcessInstances",}
0.05600000000000001
# HELP kie_server_job_scheduled_total Kie Server Started Jobs
# TYPE kie_server_job_scheduled_total counter
# HELP kie_server_data_set_execution_total Kie Server Data Set Execution
# TYPE kie_server_data_set_execution_total counter
kie_server_data_set_execution_total{uuid="jbpmProcessInstances",} 8.0
# HELP kie_server_process_instance_completed_total Kie Server Completed Process
Instances
# TYPE kie_server_process_instance_completed_total counter
# HELP kie_server_job_running_total Kie Server Running Jobs
# TYPE kie_server_job_running_total gauge
# HELP kie_server_task_failed_total Kie Server Failed Tasks
# TYPE kie_server_task_failed_total counter
# HELP kie_server_task_exited_total Kie Server Exited Tasks
# TYPE kie_server_task_exited_total counter
# HELP dmn_evaluate_decision_nanosecond DMN Evaluation Time
# TYPE dmn_evaluate_decision_nanosecond histogram
# HELP kie_server_data_set_lookups_total Kie Server Data Set Running Lookups
# TYPE kie_server_data_set_lookups_total gauge
kie_server_data_set_lookups_total{uuid="jbpmProcessInstances",} 0.0
# HELP kie_server_process_instance_duration_seconds Kie Server Process Instances
Duration
# TYPE kie_server_process_instance_duration_seconds summary
# HELP kie_server_case_duration_seconds Kie Server Case Duration
# TYPE kie_server_case_duration_seconds summary
# HELP dmn_evaluate_failed_count DMN Evaluation Failed
# TYPE dmn_evaluate_failed_count counter
# HELP kie_server_task_added_total Kie Server Added Tasks
# TYPE kie_server_task_added_total counter
kie_server_task_added_total{deployment_id="itorders_1.0.0-
SNAPSHOT",process_id="itorders.orderhardware",task_name="Prepare hardware spec",}
1.0
# HELP drl_match_fired_nanosecond Drools Firing Time
# TYPE drl_match_fired_nanosecond histogram
# HELP kie_server_container_started_total Kie Server Started Containers
# TYPE kie_server_container_started_total counter
kie_server_container_started_total{container_id="itorders_1.0.0-SNAPSHOT",} 1.0
# HELP kie_server_process_instance_sla_violated_total Kie Server Process Instances SLA
Violated
# TYPE kie_server_process_instance_sla_violated_total counter
# HELP kie_server_task_duration_seconds Kie Server Task Duration
# TYPE kie_server_task_duration_seconds summary
# HELP kie_server_job_executed_total Kie Server Executed Jobs
# TYPE kie_server_job_executed_total counter
# HELP kie_server_deployments_active_total Kie Server Active Deployments
# TYPE kie_server_deployments_active_total gauge
kie_server_deployments_active_total{deployment_id="itorders_1.0.0-SNAPSHOT",} 1.0
# HELP kie_server_process_instance_running_total Kie Server Running Process Instances
# TYPE kie_server_process_instance_running_total gauge
```

```
kie_server_process_instance_running_total{container_id="itorders_1.0.0-
SNAPSHOT",process_id="itorders.orderhardware",} 2.0
# HELP solvers_running Number of solvers currently running
# TYPE solvers_running gauge
solvers_running 0.0
# HELP kie_server_work_item_duration_seconds Kie Server Work Items Duration
# TYPE kie_server_work_item_duration_seconds summary
# HELP kie_server_job_duration_seconds Kie Server Job Duration
# TYPE kie_server_job_duration_seconds summary
# HELP solver_score_calculation_speed Number of moves per second for a particular solver
solving the constraint problem
# TYPE solver_score_calculation_speed summary
# HELP kie_server_start_time Kie Server Start Time
# TYPE kie_server_start_time gauge
kie_server_start_time{name="sample-server",server_id="sample-
server",location="http://localhost:8080/kie-server/services/rest/server",version="7.22.0-
SNAPSHOT",} 1.557285486469E12
```

If the metrics are not available in Process Server, review and verify the Process 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 17.1. Prometheus expression browser with Process Server metrics

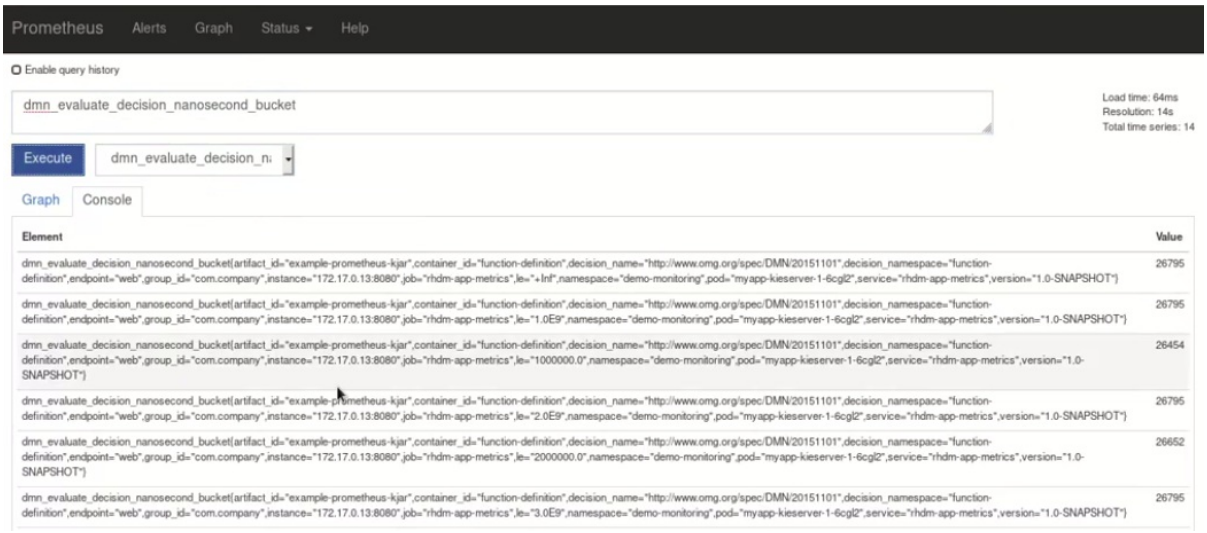


Figure 17.2. Prometheus expression browser with Process Server target

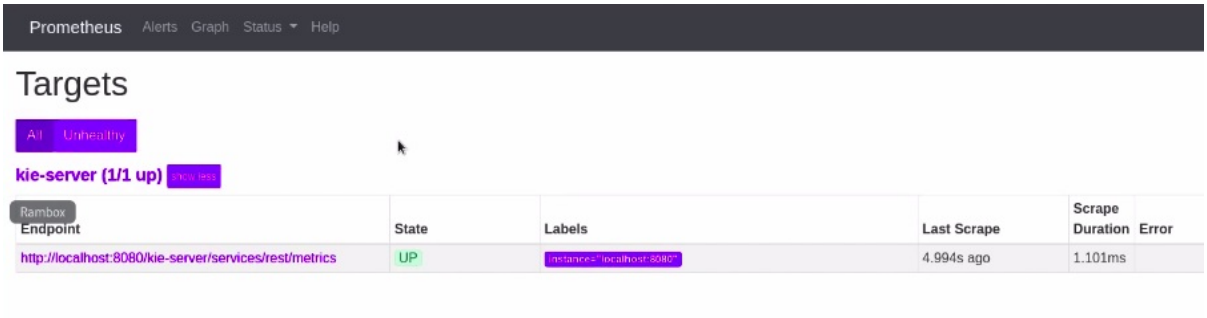


Figure 17.3. Grafana dashboard with Process Server metrics for DMN models

Figure 17.4. Grafana dashboard with Process Server metrics for solvers

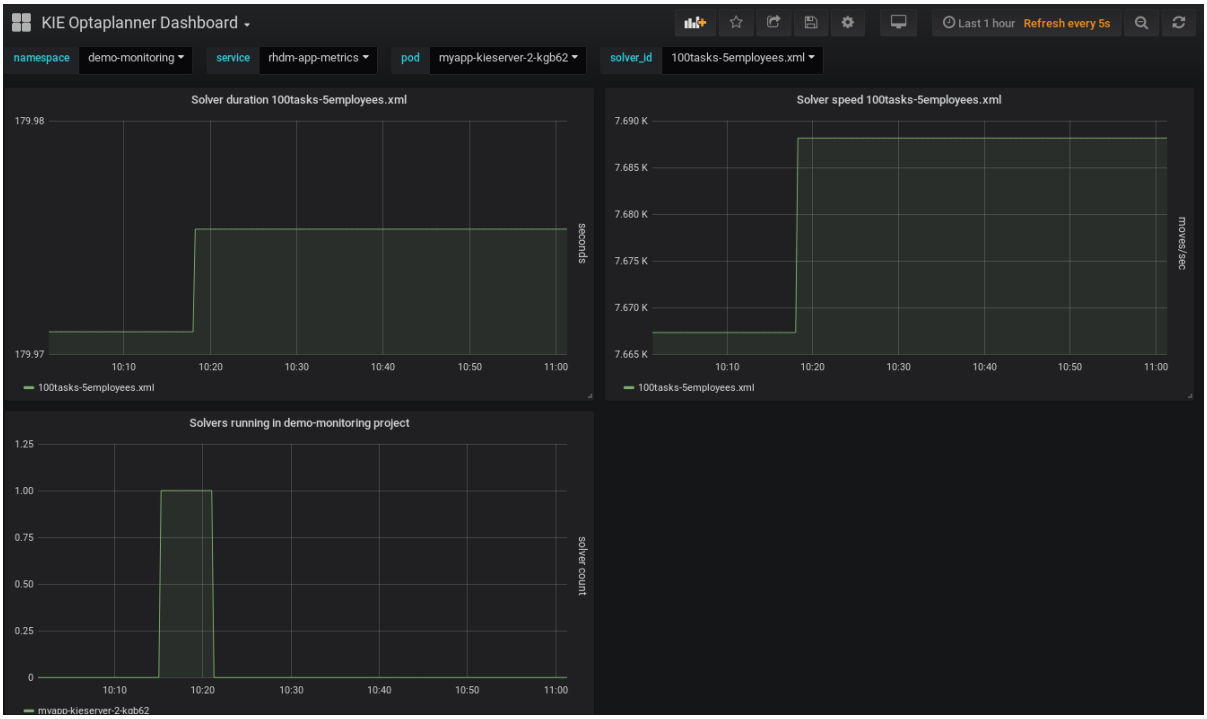
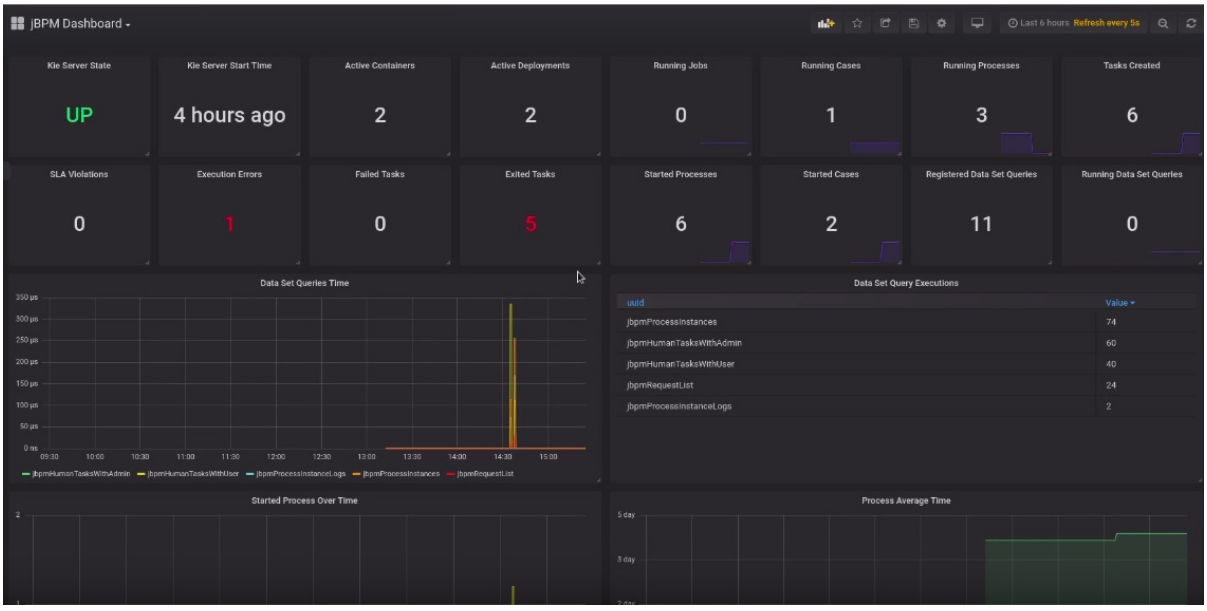


Figure 17.5. Grafana dashboard with Process Server metrics for processes, cases, and tasks



Additional resources

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

17.2. CONFIGURING PROMETHEUS METRICS MONITORING FOR PROCESS SERVER ON RED HAT OPENSHIFT CONTAINER PLATFORM

You can configure your Process 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 Process Automation Manager. For the list of available metrics that Process Server exposes with Prometheus, download the **Red Hat Process Automation Manager 7.6.0 Source Distribution** from the Red Hat Customer Portal and navigate to **~/rhpam-7.6.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus**.

Prerequisites

- Process Server is installed and deployed on Red Hat OpenShift Container Platform. For more information about Process Server on OpenShift, see the relevant OpenShift deployment option in the Product documentation for Red Hat Process Automation Manager 7.6 .
- You have **kie-server** user role access to Process 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 Process 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 Process Server on OpenShift, then in the OpenShift template that you plan to use for your OpenShift deployment (for example, **rhpam76-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 Process Server on OpenShift, then in your Process 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 Process Server to Prometheus:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    description: RHPAM Prometheus metrics exposed
  labels:
    app: myapp-kieserver
    application: myapp-kieserver
    template: myapp-kieserver
    metrics: rhpam
  name: rhpam-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 Process Server. The secret must contain the "username" and "password" elements with Process 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 Process Server metrics service.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: rhpam-service-monitor
  labels:
    team: frontend
spec:
  selector:
    matchLabels:
      metrics: rhpam
  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 Process 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 17.6. Prometheus expression browser with Process Server metrics

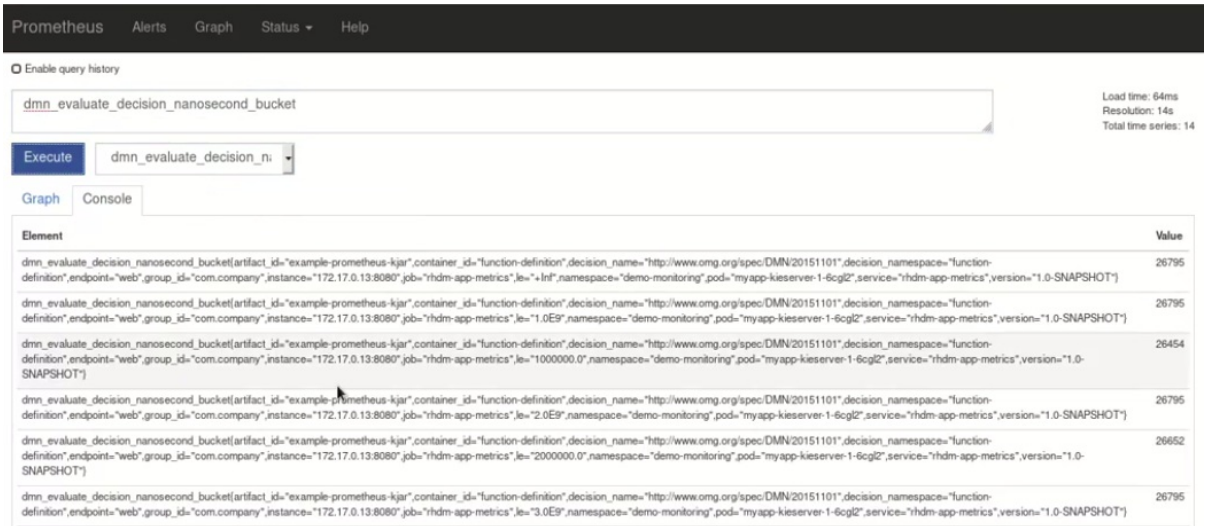


Figure 17.7. Prometheus expression browser with Process Server target

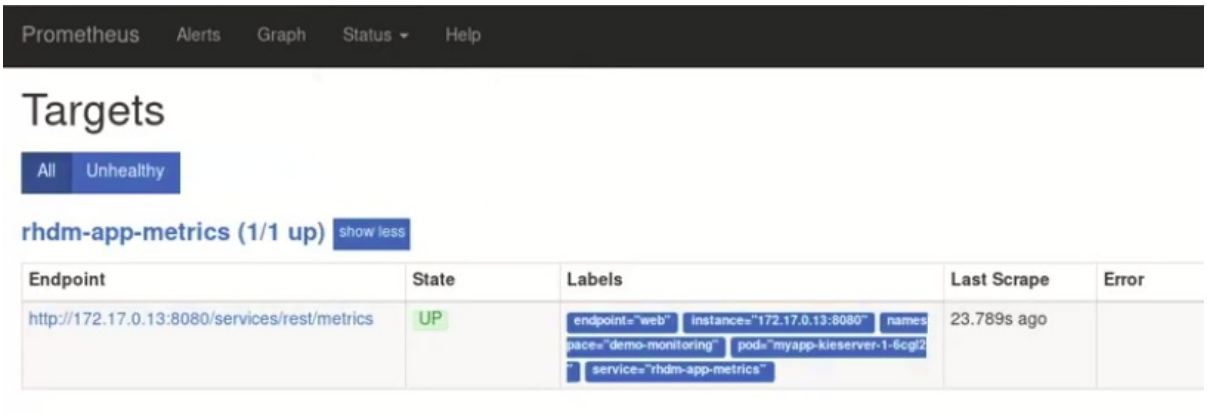


Figure 17.8. Grafana dashboard with Process Server metrics for DMN models

Figure 17.9. Grafana dashboard with Process Server metrics for solvers

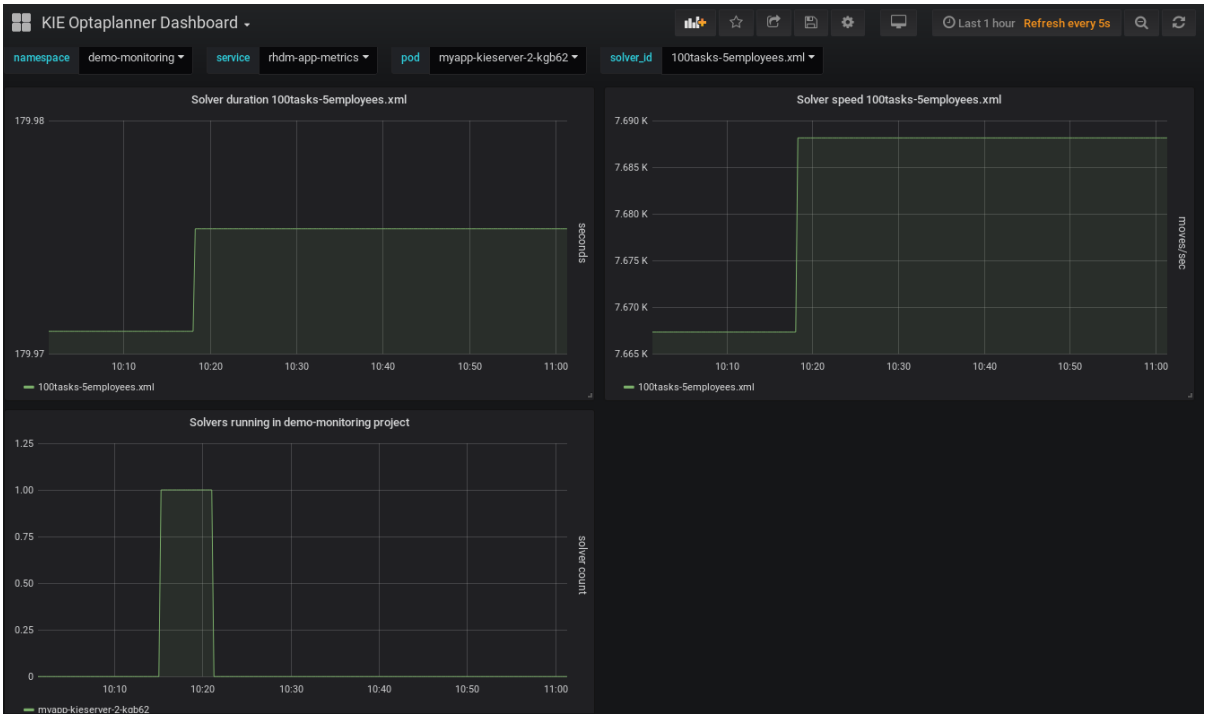
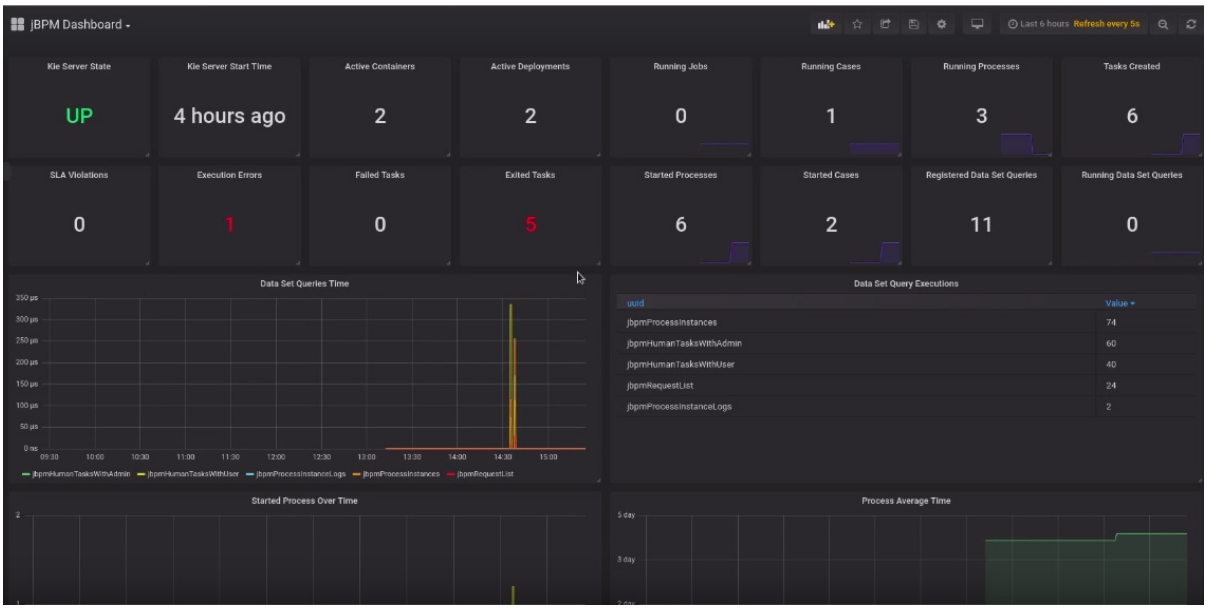


Figure 17.10. Grafana dashboard with Process Server metrics for processes, cases, and tasks



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 Process Automation Manager 7.6

17.3. EXTENDING PROMETHEUS METRICS MONITORING IN PROCESS SERVER WITH CUSTOM METRICS

After you configure your Process Server instance to use Prometheus metrics monitoring, you can extend the Prometheus functionality in Process Server to use custom metrics according to your business needs. Prometheus then collects and stores your custom metrics along with the default metrics that Process 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 Process Server instance. For information about Prometheus configuration with Process Server on-premise, see Section 17.1, “Configuring Prometheus metrics monitoring for Process Server”. For information about Prometheus configuration with Process Server on Red Hat OpenShift Container Platform, see Section 17.2, “Configuring Prometheus metrics monitoring for Process 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>7.30.0.Final-redhat-00003</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 Process 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 Process Server and deploy the built project to the running Process Server. You can deploy the project using the Business Central interface or the Process 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 Process Server, Prometheus begins collecting metrics and Process 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 18. 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: "$APPLICATION_NAME-rhpamcentr-http"
  metadata:
    name: "$APPLICATION_NAME-rhpamcentr"
    labels:
      application: "$APPLICATION_NAME"
    annotations:
      description: Route for Business Central's http service.
      haproxy.router.openshift.io/timeout: 60s
  spec:
    host: "$BUSINESS_CENTRAL_HOSTNAME_HTTP"
    to:
      name: "$APPLICATION_NAME-rhpamcentr"
```

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

CHAPTER 19. PERSISTENCE

Binary persistence, or marshaling, converts the state of the process instance into a binary data set. Binary persistence is a mechanism used to store and retrieve information persistently. The same mechanism is also applied to the session state and work item states.

When you enable persistence of a process instance:

- Red Hat Process Automation Manager transforms the process instance information into binary data. Custom serialization is used instead of Java serialization for performance reasons.
- The binary data is stored together with other process instance metadata, such as process instance ID, process ID, and the process start date.

The session can also store other forms of state, such as the state of timer jobs, or data required for business rules evaluation. Session state is stored separately as a binary data set along with the ID of the session and metadata. You can restore the session state by reloading a session with given ID. Use **ksession.getId()** to get the session ID.

Red Hat Process Automation Manager will persist the following when persistence is configured:

- *Session state*: This includes the session ID, date of last modification, the session data that business rules would need for evaluation, state of timer jobs.
- *Process instance state*: This includes the process instance ID, process ID, date of last modification, date of last read access, process instance start date, runtime data (the execution status including the node being executed, variable values, and other process instance data) and the event types.
- *Work item runtime state* : This includes the work item ID, creation date, name, process instance ID, and the work item state itself.

Based on the persisted data, you can restore the state of execution of all running process instances in case of failure or to temporarily remove running instances from memory and restore them later.

19.1. CONFIGURING PROCESS SERVER PERSISTENCE

You can configure the Process Server persistence by passing Hibernate or JPA parameters as system properties.

The Process Server can acknowledge the system properties with the following prefixes and you can use every Hibernate or JPA parameters with these prefixes:

- **javax.persistence**
- **hibernate**

Procedure

1. To configure Process Server persistence, complete any of the following tasks:
   If you want to configure Process Server persistence using Red Hat JBoss EAP configuration file, complete the following tasks:
   i. In your Red Hat Process Automation Manager installation directory, navigate to the **standalone-full.xml** file. For example, if you use Red Hat JBoss EAP installation for Red Hat Process Automation Manager, go to

$EAP_HOME/standalone/configuration/standalone-full.xml file.

ii. Open the **standalone-full.xml** file and under the **<system-properties>** tag, set your Hibernate or JPA parameters as system properties.

Example of configuring Process Server persistence using Hibernate parameters

```
<system-properties>
    ...
        <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
    ...
<system-properties>
```

Example of configuring Process Server persistence using JPA parameters

```
<system-properties>
    ...
        <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://mysql.db.server:3306/my_database?
useSSL=false&serverTimezone=UTC"/>
    ...
<system-properties>
```

If you want to configure Process Server persistence using command line, complete the following tasks:

i. Pass the parameters directly from the command line using **-Dkey=value** as follows:

Example of configuring Process Server persistence using Hibernate parameters:

```
$EAP_HOME/bin/standalone.sh -Dhibernate.hbm2ddl.auto=create-drop
```

Example of configuring Process Server persistence using JPA parameters:

```
$EAP_HOME/bin/standalone.sh -
Djavax.persistence.jdbc.url=jdbc:mysql://mysql.db.server:3306/my_database?
useSSL=false&serverTimezone=UTC
```

19.2. CONFIGURING SAFE POINTS

To allow persistence, add the **jbpm-persistence** JAR files to the classpath of your application and configure the process engine to use persistence. The process engine automatically stores the runtime state in the storage when the process engine reaches a safe point.

Safe points are points where the process instance has paused. When a process instance invocation reaches a safe point in the process engine, the process engine stores any changes to the process instance as a snapshot of the process runtime data. However, when a process instance is completed, the persisted snapshot of process instance runtime data is automatically deleted.

BPMN2 safe point nodes ensure that the process engine saves the state of the process definition at the point where the execution stops and the transaction is committed. The following BPMN2 nodes are considered safe points:

- All intermediate CATCH events
  - Timer Intermediate event
  - Error Intermediate event
  - Conditional Intermediate event
  - Compensation Intermediate event
  - Signal Intermediate event
  - Escalation Intermediate event
  - Message Intermediate event
- User tasks
- Custom (defined by the user) service tasks that do not complete the task in the handler

If a failure occurs and you need to restore the process engine runtime from the storage, the process instances are automatically restored and their execution resumes so there is no need to reload and trigger the process instances manually.

Consider the runtime persistence data to be internal to the process engine. You should not access persisted runtime data or modify them directly because this might have unexpected side effects.

For more information about the current execution state, refer to the history log. Query the database for runtime data only if absolutely necessary.

19.3. SESSION PERSISTENCE ENTITIES

Sessions are persisted as **SessionInfo** entities. These persist the state of the runtime KIE session, and store the following data:

Table 19.1. SessionInfo

Field	Description	Nullable
id	The primary key.	NOT NULL
lastModificationDate	The last time that entity was saved to a database.	
rulesByteArray	The state of a session.	NOT NULL
startDate	The session start time.	
OPTLOCK	A version field containing a lock value.	

19.4. PROCESS INSTANCE PERSISTENCE ENTITIES

Process instances are persisted as **ProcessInstanceInfo** entities, which persist the state of a process instance on runtime and store the following data:

Table 19.2. ProcessInstanceInfo

Field	Description	Nullable
instanceId	The primary key.	NOT NULL
lastModificationDate	The last time that the entity was saved to a database.	
lastReadDate	The last time that the entity was retrieved from the database.	
processId	The ID of the process.	
processInstanceByteArray	The state of a process instance in form of a binary data set.	NOT NULL
startDate	The start time of the process.	
state	An integer representing the state of a process instance.	NOT NULL
OPTLOCK	A version field containing a lock value.	

ProcessInstanceInfo has a 1:N relationship to the **EventTypes** entity.

The **EventTypes** entity contains the following data:

Table 19.3. EventTypes

Field	Description	Nullable
instanceId	A reference to the **ProcessInstanceInfo** primary key and foreign key constraint on this column.	NOT NULL
element	A finished event in the process.	

19.5. WORK ITEM PERSISTENCE ENTITIES

Work items are persisted as **workiteminfo** entities, which persist the state of the particular work item instance on runtime and store the following data:

Table 19.4. WorkItemInfo

Field	Description	Nullable
workItemId	The primary key.	NOT NULL
name	The name of the work item.	
processInstanceId	The (primary key) ID of the process. There is no foreign key constraint on this field.	NOT NULL
state	The state of a work item.	NOT NULL
OPTLOCK	A version field containing a lock value.	
workitembytearay	The work item state in as a binary data set.	NOT NULL

19.6. CORRELATION KEY ENTITIES

The **CorrelationKeyInfo** entity contains information about the correlation key assigned to the given process instance. This table is optional. Use it only when you require correlation capabilities.

Table 19.5. CorrelationKeyInfo

Field	Description	Nullable
keyId	The primary key.	NOT NULL
name	The assigned name of the correlation key.	
processInstanceId	The ID of the process instance which is assigned to the correlation key.	NOT NULL
OPTLOCK	A version field containing a lock value.	

The **CorrelationPropertyInfo** entity contains information about correlation properties for a correlation key assigned the process instance.

Table 19.6. CorrelationPropertyInfo

Field	Description	Nullable
propertyId	The primary key.	NOT NULL

Field	Description	Nullable
name	The name of the property.	
value	The value of the property.	NOT NULL
OPTLOCK	A version field containing a lock value.	
correlationKey_keyId	A foreign key mapped to the correlation key.	NOT NULL

19.7. CONTEXT MAPPING ENTITY

The **ContextMappingInfo** entity contains information about the contextual information mapped to a **KieSession**. This is an internal part of **RuntimeManager** and can be considered optional when **RuntimeManager** is not used.

Table 19.7. ContextMappingInfo

Field	Description	Nullable
mappingId	The primary key.	NOT NULL
CONTEXT_ID	The context identifier.	NOT NULL
KSESSION_ID	The **KieSession** identifier.	NOT NULL
OPTLOCK	A version field containing a lock value.	
OWNER_ID	Holds the identifier of the deployment unit that the given mapping is associated with	

19.8. PESSIMISTIC LOCKING SUPPORT

The default locking mechanism for persistence of processes is *optimistic*. With multi-thread high concurrency to the same process instance, this locking strategy can result in bad performance.

This can be changed at runtime to allow the user to set locking on a per process basis and to allow it to be *pessimistic* (the change can be made at a per KIE Session level or Runtime Manager level as well and not just at the process level).

To set a process to use pessimistic locking, use the following configuration in the runtime environment:

```
import org.kie.api.runtime.Environment;
import org.kie.api.runtime.EnvironmentName;
import org.kie.api.runtime.manager.RuntimeManager;
```

```
import org.kie.api.runtime.manager.RuntimeManagerFactory;

...

env.set(EnvironmentName.USE_PESSIMISTIC_LOCKING, true); (1)

RuntimeManager manager =
RuntimeManagerFactory.Factory.get().newPerRequestRuntimeManager(environment); (2)
```

1. **env** is an instance of **org.kie.api.runtime.Environment**.
2. Create your Runtime Manager by using this environment.

19.9. PERSISTING PROCESS VARIABLES IN A SEPARATE DATABASE SCHEMA IN RED HAT PROCESS AUTOMATION MANAGER

When you create process variables in Red Hat Process Automation Manager to use within the processes that you define, Red Hat Process Automation Manager stores those process variables as binary data in a default database schema. You can persist process variables in a separate database schema for greater flexibility in maintaining and implementing your process data.

For example, persisting your process variables in a separate database schema can help you perform the following tasks:

- Maintain process variables in human-readable format
- Make the variables available to services outside of Red Hat Process Automation Manager
- Clear the log of the default database tables in Red Hat Process Automation Manager without losing process variable data

NOTE

This procedure applies to process variables only. This procedure does not apply to case variables.

Prerequisites

- You have defined processes in Red Hat Process Automation Manager for which you want to implement variables.
- If you want to persist variables in a database schema outside of Red Hat Process Automation Manager, you have created a data source and the separate database schema that you want to use. For information about creating data sources, see *Configuring Business Central settings and properties*.

Procedure

1. In the data object file that you use as a process variable, add the following elements to configure variable persistence:

   Example Person.java object configured for variable persistence

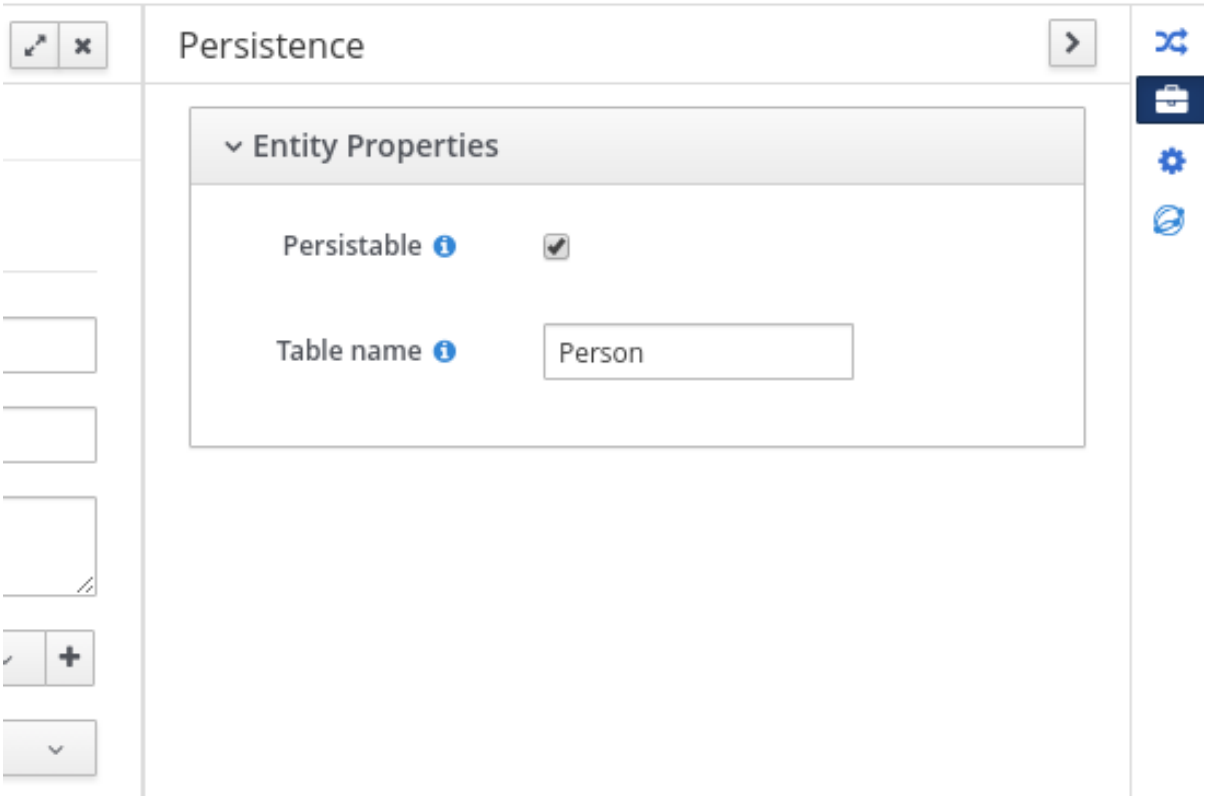
```
@javax.persistence.Entity (1)
@javax.persistence.Table(name = "Person") (2)
public class Person extends org.drools.persistence.jpa.marshaller.VariableEntity (3)
implements java.io.Serializable { (4)

 static final long serialVersionUID = 1L;

 @javax.persistence.GeneratedValue(strategy = javax.persistence.GenerationType.AUTO,
generator = "PERSON_ID_GENERATOR")
 @javax.persistence.Id (5)
 @javax.persistence.SequenceGenerator(name = "PERSON_ID_GENERATOR",
sequenceName = "PERSON_ID_SEQ")
 private java.lang.Long id;

 private java.lang.String name;

 private java.lang.Integer age;

 public Person() {
 }

 public java.lang.Long getId() {
  return this.id;
 }

 public void setId(java.lang.Long id) {
  this.id = id;
 }

 public java.lang.String getName() {
  return this.name;
 }

 public void setName(java.lang.String name) {
  this.name = name;
 }

 public java.lang.Integer getAge() {
  return this.age;
 }

 public void setAge(java.lang.Integer age) {
  this.age = age;
 }

 public Person(java.lang.Long id, java.lang.String name,
   java.lang.Integer age) {
  this.id = id;
  this.name = name;
  this.age = age;
 }

}
```

(1) Configures the data object as a persistence entity.

2. Defines the database table name used for the data object.
3. Creates a separate **MappedVariable** mapping table that maintains the relationship between this data object and the associated process instance. If you do not need this relationship maintained, you do not need to extend the **VariableEntity** class. Without this extension, the data object is still persisted, but contains no additional data.
4. Configures the data object as a serializable object.
5. Sets a persistence ID for the object.

To make the data object persistable using Business Central, navigate to the data object file in your project, click the **Persistence** icon in the upper-right corner of the window, and configure the persistence behavior:

Figure 19.1. Persistence configuration in Business Central



2. In the **pom.xml** file of your project, add the following dependency for persistence support. This dependency contains the **VariableEntity** class that you configured in your data object.

Project dependency for persistence

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-persistence-jpa</artifactId>
  <version>${rhpam.version}</version>
  <scope>provided</scope>
</dependency>
```

3. In the **~/META-INF/kie-deployment-descriptor.xml** file of your project, configure the JPA marshalling strategy and a persistence unit to be used with the marshaller. The JPA marshalling strategy and persistence unit are required for objects defined as entities.

JPA marshaller and persistence unit configured in the kie-deployment-descriptor.xml file

```
<marshalling-strategy>
  <resolver>mvel</resolver>
  <identifier>new
org.drools.persistence.jpa.marshaller.JPAPlaceholderResolverStrategy("myPersistenceUnit",
classLoader)</identifier>
  <parameters/>
</marshalling-strategy>
```

4. In the ~/**META-INF** directory of your project, create a **persistence.xml** file that specifies in which data source you want to persist the process variable:

Example persistence.xml file with data source configuration

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:orm="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd
http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_2_0.xsd">
   <persistence-unit name="myPersistenceUnit" transaction-type="JTA">
      <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
      <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source> (1)
      <class>org.space.example.Person</class>
      <exclude-unlisted-classes>true</exclude-unlisted-classes>
      <properties>
         <property name="hibernate.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/>
         <property name="hibernate.max_fetch_depth" value="3"/>
         <property name="hibernate.hbm2ddl.auto" value="update"/>
         <property name="hibernate.show_sql" value="true"/>
         <property name="hibernate.id.new_generator_mappings" value="false"/>
         <property name="hibernate.transaction.jta.platform"
value="org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform"/>
      </properties>
   </persistence-unit>
</persistence>
```

(1) Sets the data source in which the process variable is persisted

To configure the marshalling strategy, persistence unit, and data source using Business Central, navigate to project **Settings → Deployments → Marshalling Strategies** and to project **Settings → Persistence**:

Figure 19.2. JPA marshaller configuration in Business Central

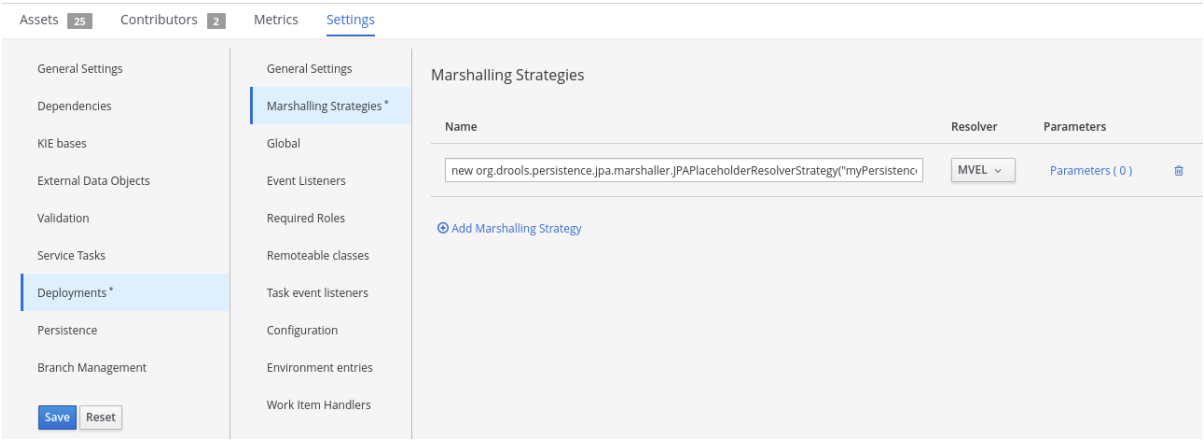
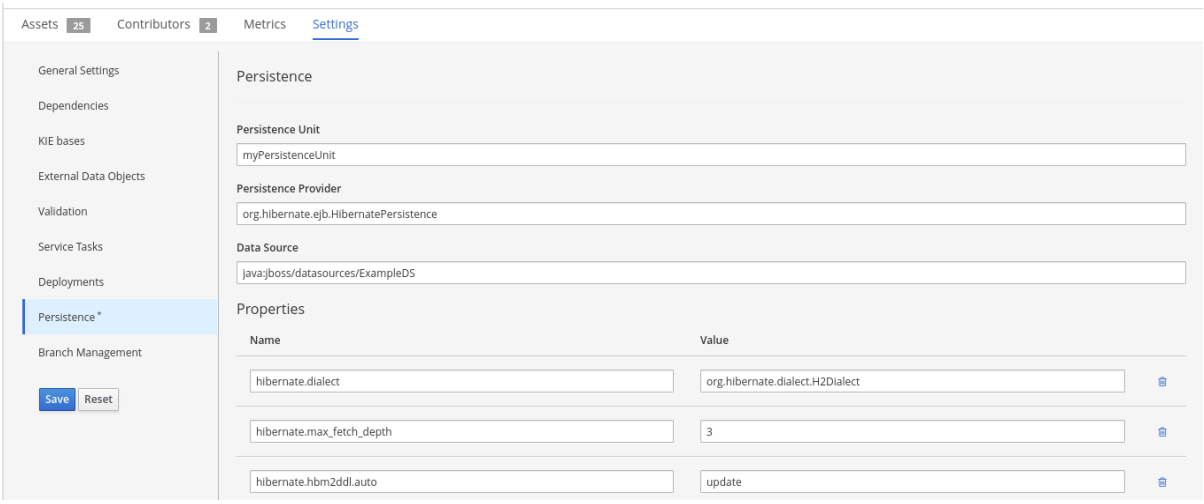


Figure 19.3. Persistence unit and data source configuration in Business Central



CHAPTER 20. DEFINE THE LDAP LOGIN DOMAIN

When you are setting up Red Hat Process Automation 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 21. AUTHENTICATING THIRD-PARTY CLIENTS THROUGH RH-SSO

To use the different remote services provided by Business Central or by Process 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 Process 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

21.1. BASIC AUTHENTICATION

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

- For web based remote repositories endpoint:

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

- For Process Server:

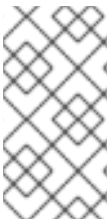
```
curl http://admin:password@localhost:8080/kie-execution-server/services/rest/server/
```

CHAPTER 22. PROCESS SERVER SYSTEM PROPERTIES

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

Table 22.1. System properties for disabling Process 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.jbpm.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the Red Hat Process Automation Manager support (for example, processes support).
org.jbpm.ui.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the Red Hat Process Automation Manager UI extension.
org.jbpm.case.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the Red Hat Process Automation Manager case management extension.
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.dmn.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the Process Server DMN support.
org.kie.swagger.server.ext.disabled	**true**, **false**	**false**	If set to **true**, disables the Process Server swagger documentation support



NOTE

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

Table 22.2. System properties required for Process Automation Manager controller

Property	Values	Default	Description

Property	Values	Default	Description
org.kie.server.id	String	N/A	An arbitrary ID to be assigned to the server. If a headless Process Automation Manager controller is configured outside of Business Central, this is the ID under which the server connects to the headless Process Automation Manager controller to fetch the KIE container configurations. If not provided, the ID is automatically generated.
org.kie.server.user	String	**kieserver**	The user name used to connect with the Process Server from the Process Automation Manager controller, required when running in managed mode. Set this property in Business Central system properties. Set this property when using a Process Automation Manager controller.
org.kie.server.pwd	String	**kieserver1!**	The password used to connect with the Process Server from the Process Automation Manager controller, required when running in managed mode. Set this property in Business Central system properties. Set this property when using a Process Automation Manager controller.
org.kie.server.token	String	N/A	A property that enables you to use token-based authentication between the Process Automation Manager controller and the Process Server instead of the basic user name and password authentication. The Process Automation 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 Process Server instance used by the Process Automation 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 Process Automation Manager controller.
org.kie.server.controller	Comma-separated list	N/A	A comma-separated list of URLs to the Process Automation Manager controller REST endpoints, for example, **http://localhost:8080/business-central/rest/controller**. Setting this property is required when using a Process Automation Manager controller.

Property	Values	Default	Description
org.kie.server.controller.user	String	**kieserver**	The user name to connect to the Process Automation Manager controller REST API. Setting this property is required when using a Process Automation Manager controller.
org.kie.server.controller.pwd	String	**kieserver1!**	The password to connect to the Process Automation Manager controller REST API. Setting this property is required when using a Process Automation Manager controller.
org.kie.server.controller.token	String	N/A	A property that enables you to use token-based authentication between the Process Server and the Process Automation 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 Process Server to the Process Automation Manager controller when the server starts.

Table 22.3. Persistence system properties

Property	Values	Default	Description
org.kie.server.persistence.ds	String	N/A	A data source JNDI name. Set this property when enabling the BPM support.
org.kie.server.persistence.tm	String	N/A	A transaction manager platform for Hibernate properties. Set this property when enabling the BPM support.
org.kie.server.persistence.dialect	String	N/A	The Hibernate dialect to be used. Set this property when enabling the BPM support.

Property	Values	Default	Description
org.kie.server.persistence.schema	String	N/A	The database schema to be used.

Table 22.4. Executor system properties

Property	Values	Default	Description
org.kie.executor.interval	Integer	**0**	The time between the moment the Red Hat Process Automation Manager executor finishes a job and the moment it starts a new one, in a time unit specified in the **org.kie.executor.timeunit** property.
org.kie.executor.timeunit	**java.util.concurrent.TimeUnit** constant	**SECONDS**	The time unit in which the **org.kie.executor.interval** property is specified.
org.kie.executor.pool.size	Integer	**1**	The number of threads used by the Red Hat Process Automation Manager executor.
org.kie.executor.retry.count	Integer	**3**	The number of retries the Red Hat Process Automation Manager executor attempts on a failed job.
org.kie.executor.jms.queue	String	**queue/KIE.SERVER.EXECUTOR**	Job executor JMS queue for Process Server.
org.kie.executor.disabled	**true**, **false**	**false**	If set to **true**, disables the Process Server executor.

Table 22.5. Human task system properties

Property	Values	Default	Description

Property	Values	Default	Description
org.jbpm.ht.callback	**mvel** **ldap** **db** **jaas** **props** **custom**	**jaas**	A property that specifies the implementation of user group callback to be used: ● **mvel**: Default; mostly used for testing. ● **ldap**: LDAP; requires additional configuration in the **jbpm.usergroup.callback.properties** file. ● **db**: Database; requires additional configuration in the **jbpm.usergroup.callback.properties** file. ● **jaas**: JAAS; delegates to the container to fetch information about user data. ● **props**: A simple property file; requires additional file that keeps all information (users and groups). ● **custom**: A custom implementation; specify the fully qualified name of the class in the **org.jbpm.ht.custom.callback** property.
org.jbpm.ht.custom.callback	Fully qualified name	N/A	A custom implementation of the **UserGroupCallback** interface in case the **org.jbpm.ht.callback** property is set to **custom**.
org.jbpm.task.cleanup.enabled	**true**, **false**	**true**	Enables task cleanup job listener to remove tasks once the process instance is completed.
org.jbpm.task.bam.enabled	**true**, **false**	**true**	Enables task BAM module to store task related information.
org.jbpm.ht.admin.user	String	**Administrator**	User who can access all the tasks from Process Server.
org.jbpm.ht.admin.group	String	**Administrators**	The group that users must belong to in order to view all the tasks from Process Server.

Table 22.6. 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 Process Automation Manager controller.
kie.keystore.key.ctrl.pwd	String	N/A	The password of an alias for default REST Process Automation Manager controller.

Table 22.7. 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.
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 Process Server extension accepts custom classes annotated by the **XmlRootElement** or **Remotable** annotations only.
org.kie.server.bypass.auth.user	**true**, **false**	**false**	A property that enables you to bypass the authenticated user for task-related operations, for example queries.

Property	Values	Default	Description
org.jbpm.rule.task.firelimit	Integer	**10000**	This property specifies the maximum number of executed rules to avoid situations where rules run into an infinite loop and make the server completely unresponsive.
org.jbpm.ejb.timer.local.cache	**true**, **false**	**true**	This property turns off the EJB Timers local cache.
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 Process Server state files are stored.

Property	Values	Default	Description
org.kie.server.sync.deploy	**true**, **false**	**false**	A property that instructs the Process Server to hold the deployment until the Process Automation 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 Process Automation Manager controller is asynchronous. The application starts, connects to the Process Automation 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 Process Automation 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.
org.kie.server.startup.strategy	**ControllerBasedStartupStrategy**, **LocalContainersStartupStrategy**	**ControllerBasedStartupStrategy**	The Startup strategy of Process 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 Process Server management API.

Property	Values	Default	Description
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 23. PROCESS SERVER CAPABILITIES AND EXTENSIONS

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

Table 23.1. Process Server capabilities and extensions

Capability name	Extension name	Description
KieServer	**KieServer**	Provides the core capabilities of Process 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
BPM	**jBPM**	Provides the Business Process Management (BPM) capabilities, such as managing user tasks and executing business processes
BPM-UI	**jBPM-UI**	Provides additional user-interface capabilities related to business processes, such as rendering XML forms and SVG images in process diagrams
CaseMgmt	**Case-Mgmt**	Provides the case management capabilities for business processes, such as managing case definitions and milestones
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 Process Server REST API

To view the supported extensions of a running Process 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 Process Server information

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

Example JSON response with Process Server information

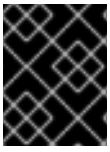
```
{
  "type": "SUCCESS",
  "msg": "Kie Server info",
  "result": {
    "kie-server-info": {
```

```
    "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",
      "BPM",
      "CaseMgmt",
      "BPM-UI",
      "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, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger]', messages=null',
mode=DEVELOPMENT}started successfully at Sun Aug 18 23:11:05 UTC 2019"
        ]
      }
    ],
    "mode": "DEVELOPMENT"
  }
 }
}
```


To enable or disable Process Server extensions, configure the related ***.server.ext.disabled** Process Server system property. For example, to disable the **BRM** capability, set the system property **org.drools.server.ext.disabled=true**. For all Process Server system properties, see Chapter 22, *Process Server system properties*.

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

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



IMPORTANT

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

23.1. EXTENDING AN EXISTING PROCESS SERVER CAPABILITY WITH A CUSTOM REST API ENDPOINT

The Process Server REST API enables you to interact with your KIE containers and business assets

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

As an example, this procedure extends the **Drools** Process 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>7.30.0.Final-redhat-00003</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 Process 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 Process Server instance calls the **getAppComponents** method, so the **if (!OWNER_EXTENSION.equals(extension))** call returns an empty collection for any extensions other than the specified **OWNER_EXTENSION** extension.
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 Process 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 Process 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 Process 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 Process Server and deploy the built project to the running Process Server. You can deploy the project using either the Business Central interface or the Process 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 Process 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
```

23.2. EXTENDING PROCESS SERVER TO USE A CUSTOM DATA TRANSPORT

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

As an example, this procedure adds a custom data transport to Process 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>7.30.0.Final-redhat-00003</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 Process 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 Process 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 Process Server session. To end a Process Server session in stream mode, send either an **exit** or **quit** command to the server.

4. To make the new data transport discoverable for Process 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 Process Server and deploy the built project to the running Process Server. You can deploy the project using either the Business Central interface or the Process 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 Process Server, you can view the status of the new data transport in your Process 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 Process Server:

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

```
telnet 127.0.0.1 9123
```

Example interactions with Process 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
```

23.3. EXTENDING THE PROCESS SERVER CLIENT WITH A CUSTOM CLIENT API

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

As an example, this procedure adds a custom client API to Process 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>7.30.0.Final-redhat-00003</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 Process 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 Process 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 Process 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 Process Server client infrastructure

(2) Defines the Process 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 Process 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 Process Server and deploy the built project to the running Process Server. You can

deploy the project using either the Business Central interface or the Process 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 Process Server, you can start interacting with your new Process Server client. You use your new client in the same way as the standard Process 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 Process 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 Process 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 24. PERFORMANCE TUNING CONSIDERATIONS WITH PROCESS SERVER

The following key concepts or suggested practices can help you optimize Process 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 Process Automation Manager.

Ensure that development mode is enabled during development

You can set Process Server or specific projects in Business Central to use **production** mode or **development** mode. By default, Process 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 Process Server performance during development, such as disabled duplicate GAV detection. Use development mode until your Red Hat Process Automation 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 9, *Configuring the environment mode in Process Server and Business Central*
- *Packaging and deploying a Red Hat Process Automation Manager project*

Adapt Process Server capabilities and extensions to your specific needs

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

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

For information about enabling, disabling, or extending Process Server capabilities, see Chapter 23, *Process Server capabilities and extensions* .

CHAPTER 25. ADDITIONAL RESOURCES

- *Installing and configuring Red Hat Process Automation Manager on Red Hat JBoss EAP 7.2*
- *Planning a Red Hat Process Automation Manager installation*
- *Installing and configuring Red Hat Process Automation Manager on Red Hat JBoss EAP 7.2*
- *Deploying a Red Hat Process Automation Manager immutable server environment on Red Hat OpenShift Container Platform*
- *Deploying a Red Hat Process Automation Manager authoring environment on Red Hat OpenShift Container Platform*
- *Deploying a Red Hat Process Automation Manager freeform managed server environment on Red Hat OpenShift Container Platform*
- *Deploying a Red Hat Process Automation Manager environment on Red Hat OpenShift Container Platform using Operators*

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Monday, June 07, 2021.