Red Hat JBoss BRMS 6.4

Red Hat JBoss BRMS Realtime Decision Server for OpenShift

Using Red Hat JBoss BRMS Realtime Decision Server for OpenShift

Last Updated: 2019-05-13

Red Hat JBoss BRMS 6.4 Red Hat JBoss BRMS Realtime Decision Server for OpenShift

Using Red Hat JBoss BRMS Realtime Decision Server for OpenShift

Abstract

Guide to using Red Hat JBoss BRMS Realtime Decision Server for OpenShift

Table of Contents

PART I. INTRODUCTION

CHAPTER 1. WHAT IS THE RED HAT JBOSS BRMS REALTIME DECISION SERVER?

Red Hat JBoss BRMS Realtime Decision Server for OpenShift provides a platform for executing business rules on Red Hat JBoss BRMS Realtime Decision Server. Developers can quickly build, scale, and test applications deployed across hybrid environments.

PART II. BEFORE YOU BEGIN

CHAPTER 2. COMPARISON: RED HAT JBOSS BRMS AND REALTIME DECISION SERVER FOR OPENSHIFT

This topic details the differences between Realtime Decision Server for OpenShift and the full, non-PaaS release of Red Hat JBoss BRMS, and provides instructions specific to running and configuring Realtime Decision Server for OpenShift. Documentation for other Red Hat JBoss BRMS functionality not specific to Realtime Decision Server for OpenShift can be found in the Red Hat JBoss BRMS documentation on the Red Hat Customer Portal.

EAP_HOME in this documentation, as in the Red Hat JBoss BRMS documentation , is used to refer to the JBoss EAP installation directory where the decision server is deployed. The location of ***EAP_HOME*** inside Realtime Decision Server for OpenShift is ***/opt/eap/***, which the **JBOSS_HOME** environment variable is also set to by default.

2.1. FUNCTIONALITY DIFFERENCES FOR REALTIME DECISION SERVER FOR OPENSHIFT

There are several major functionality differences in Realtime Decision Server for OpenShift:

- Realtime Decision Server for OpenShift extends EAP for OpenShift, and any capabilities or limitations it has are also found in Realtime Decision Server for OpenShift.
- Only stateless scenarios are supported.
- To connect to the Decision Server web console, click the **Connect** button in the Decision Server pod of the OpenShift web console, or the **Open Java Console** button in OpenShift 3.2.
- There is no support for authoring any content through the BRMS Console or API.

2.2. VERSION COMPATIBILITY AND SUPPORT

For more information on OpenShift image version compatibility, see the xPaaS part of the OpenShift and Atomic Platform Tested Integrations page.

2.3. DEPRECATED IMAGE STREAMS AND APPLICATION TEMPLATES FOR REALTIME DECISION SERVER FOR OPENSHIFT

IMPORTANT

The Realtime Decision Server for OpenShift image version number 6.2 is deprecated and it will no longer receive updates of image and application templates.

The Realtime Decision Server for OpenShift image version number 6.3 is deprecated and it will no longer receive updates of image and application templates.

It is recommended to use the version 6.4 of Realtime Decision Server for OpenShift image and application templates to deploy new applications.

2.4. MANAGING REALTIME DECISION SERVER FOR OPENSHIFT

As Realtime Decision Server for OpenShift is built off EAP for OpenShift, the JBoss EAP Management CLI is accessible from within the container for troubleshooting purposes.

1. First open a remote shell session to the running pod:

```
$ oc rsh <pod_name>
```

2. Then run the following from the remote shell session to launch the JBoss EAP Management CLI:

```
$ /opt/eap/bin/jboss-cli.sh
```

WARNING

Any configuration changes made using the JBoss EAP Management CLI on a running container will be lost when the container restarts.

Making configuration changes to the JBoss EAP instance inside EAP for OpenShift is different from the process you may be used to for a regular release of JBoss EAP.

2.5. SECURITY IN REALTIME DECISION SERVER FOR OPENSHIFT

Access is limited to users with the ***kie-server*** authorization role. A user with this role can be specified via the ***KIE_SERVER_USER*** and ***KIE_SERVER_PASSWORD*** environment variables.

NOTE

The HTTP/REST endpoint is configured to only allow the execution of KIE containers and querying of KIE Server resources. Administrative functions like creating or disposing Containers, updating ReleaseIds or Scanners, etc. are restricted. The JMS endpoint currently does not support these restrictions. In the future, more fine-grained security configuration should be available for both endpoints.

2.6. INITIAL SETUP

The Tutorials in this guide follow on from and assume an OpenShift instance similar to that created in the OpenShift Primer.

PART III. GET STARTED

CHAPTER 3. DEPLOYMENT CONSIDERATIONS FOR REALTIME DECISION SERVER FOR OPENSHIFT

3.1. CONFIGURING KEYSTORES

Realtime Decision Server for OpenShift requires two keystores:

- An SSL keystore to provide private and public keys for https traffic encryption
- A JGroups keystore to provide private and public keys for network traffic encryption between nodes in the cluster

These keystores are expected by Realtime Decision Server for OpenShift, even if the application uses only http on a single-node OpenShift instance. Note that self-signed certificates do not provide secure communication and are intended for internal testing purposes.

WARNING

For production environments Red Hat recommends that you use your own SSL certificate purchased from a verified Certificate Authority (CA) for SSL-encrypted connections (HTTPS).

See Generate a SSL Encryption Key and Certificate for more information on how to create a keystore with self-signed or purchased SSL certificates.

3.2. GENERATING THE SECRET

OpenShift uses objects called **Secrets** to hold sensitive information, such as passwords or keystores. See the Secrets chapter in the OpenShift documentation for more information.

Realtime Decision Server for OpenShift requires a secret that holds the two keystores described earlier. This provides the necessary authorization to applications in the project.

Use the Java and JGroup keystore files to create a secret for the project:

```
$ oc create secret generic <rds-secret-name> --from-file=<jgroups.jceks> --from-file=<keystore.jks>
```

After the secret has been generated, it can be associated with a service account.

3.3. CREATING THE SERVICE ACCOUNT

The service account allows users to associate certain secrets and roles with applications in a project namespace. This provides the application with the necessary authorization to run with all required privileges.

1. Create a service account to be used for Realtime Decision Server for OpenShift deployment:

```
$ oc create serviceaccount <service-account-name>
```

2. Add the **view** role to the service account. This enables the service account to view all the resources in the application namespace in OpenShift, which is necessary for managing the cluster.

   ```
   $ oc policy add-role-to-user view system:serviceaccount:<project-name>:<service-account-name>
   ```

3. Add the secret created for the project to the service account:

   ```
   $ oc secret add sa/<service-account-name> secret/<secret-name>
   ```

CHAPTER 4. PREPARING A RED HAT JBOSS BRMS PROJECT REPOSITORY FOR OPENSHIFT

4.1. STATELESS SESSIONS

NOTE

The Red Hat JBoss BRMS project must be configured to be stateless. OpenShift does not support stateful sessions on KIE servers.

A Red Hat JBoss BRMS project running in Red Hat JBoss BRMS could be configured to be stateless of stateful. A project that has already been deployed on a Decision Server xPaaS image will be configured to be stateless.

Ensuring a Red Hat JBoss BRMS project is stateless:
The **Knowledge Session** (in the web console, **Open Project Editor → Project Settings → Knowledge bases and sessions**) displays whether the session is configured to be **Stateless**.

4.2. CONFIGURING THE PROJECT REMOTE REPOSITORY

The project must be configured to use a remote repository so that Red Hat JBoss BRMS can push changes and OpenShift can pull the repository to build the application.

In the application repository files:

1. The **pom.xml** must be configured to use a remote repository so that OpenShift can access it.

```
...
<distributionManagement>
  <repository>
    <id>deployment</id>
    <name>OpenShift Maven repo</name>
    <url>http://maven.example/deployment/filepath/</url>
  </repository>

  <snapshotRepository>
    <id>deployment</id>
    <name>OpenShift Maven repo</name>
    <url>http://maven.example/snapshots/filepath/</url>
  </snapshotRepository>
</distributionManagement>
...
```

For more information, see the Red Hat JBoss BRMS Administration and Configuration Guide .

2. The **configuration/settings.xml** file must have the remote repository defined so that OpenShift can download the application artifacts.

```
...
<profiles>
  <profile>
    <id>openshift-mirror-repositories</id>
    <repositories>
```

```
      <repository>
        <id>openshift-mirror</id>
        <url>http://maven.example/public/filepath/</url>
      </repository>
    </repositories>

    <pluginRepositories>
      <pluginRepository>
        <id>openshift-mirror</id>
        <url>http://maven.example/public/filepath/</url>
      </pluginRepository>
    </pluginRepositories>
  </profile>
</profiles>
...
```

For more information, see the Red Hat JBoss BRMS Installation Guide .

3. The hidden *.s2i/environment* file defines the KIE container deployment, including which KIE jars to use and the location from which to retrieve them. When OpenShift deploys the built image, the pod name is derived from the deployment alias defined in this file:

```
KIE_CONTAINER_DEPLOYMENT=<alias>=<group_id>:<artifact_id>:<version>
```

For example:

```
KIE_CONTAINER_DEPLOYMENT=RulesTest=com.example.openshift:example_workflow:1.0
```

NOTE

Defining the container name here is necessary because the default behavior of the KIE server is to search for the default stateful session and fail if it does not find one.

CHAPTER 5. UPDATING RULES

Each image is built from a snapshot of a specific Maven repository. When a new rule is added, or an existing rule modified, a new image must be created and deployed for the rule modifications to take effect.

Updating the Application
The **KIE_CONTAINER_DEVELOPMENT_OVERRIDE** variable can be used to explicitly override the **KIE_CONTAINER_DEPLOYMENT** variable set in the original deployment.
When an application has been modified and is ready to be deployed, include the updated version details for the **KIE_CONTAINER_DEPLOYMENT_OVERRIDE** variable in the **.s2i/environment** file. This can then be pushed to your repository to be built as an image.
Alternatively, start a binary build from the local repo:

```
$ oc start-build <RulesTest> --from-repo=</repository/filepath>
```

This sends the contents of the Git repository directly to OpenShift. If Incremental Builds has been configured, the new build pulls the image previously used, extracts the Maven repository for the new pod, and downloads the missing content.

5.1. RECREATE UPDATE STRATEGY

Use the Recreate Update Strategy for Realtime Decision Server for OpenShift deployment. This update strategy automatically scales down the old deployment to 0 and deploys the new version. After the new version is validated, the new deployment is automatically scaled up to the replica size of the old deployment.

The Recreate update strategy supports Lifecycle Hooks and is set as the default update strategy in Realtime Decision Server for OpenShift application templates.

NOTE

Realtime Decision Server for OpenShift will be inactive during the Recreate update process, until the new deployment has been validated and scaled. During this period, REST clients may return **503 service unavailable** errors and A-MQ clients may experience **timeouts**.

IMPORTANT

The Rolling Update Strategy is not supported for Realtime Decision Server for OpenShift. Although multiple concurrent versions of an application are supported in a deployment, a cluster can only support valid routing to pods of the same version.

5.2. MULTIPLE CONCURRENT VERSIONS

An application may contain multiple concurrent KIE containers of different versions. Each container has a classloader environment and a unique identifier. The unique identifier is one of either a container ID or a deployment ID, which are synonymous.

Multiple versions are deployed using the **KIE_CONTAINER_DEPLOYMENT** variable, specifying the *<alias>=<group_id>:<artifact_id>:<version>* for each version of the application, separated by a pipe (**|**) in the **.s2i/environment** file. For example:

```
KIE_CONTAINER_DEPLOYMENT=RulesTest=com.example.openshift:example_workflow:1.0|RulesTe
st=com.example.openshift:example_workflow:1.1
```

creates the following:

```
KIE_CONTAINER_DEPLOYMENT=RulesTest=com.example.openshift:example_workflow:1.0|RulesTe
st=com.example.openshift:example_workflow:1.1
KIE_CONTAINER_DEPLOYMENT_ORIGINAL:
KIE_CONTAINER_DEPLOYMENT_OVERRIDE:
RulesTest=com.example.openshift:example_workflow:1.0|RulesTest=com.example.openshift:example_
workflow:1.1
KIE_CONTAINER_DEPLOYMENT_COUNT: 2
KIE_CONTAINER_ID_0: df729302a0b7293c0729384710dd82a1
KIE_CONTAINER_KJAR_GROUP_ID_0: com.example.openshift
KIE_CONTAINER_KJAR_ARTIFACT_ID_0: example_workflow
KIE_CONTAINER_KJAR_VERSION_0: 1.0
KIE_CONTAINER_ID_1: 01932fc2931b02cb042ab29d9fc82a8a
KIE_CONTAINER_KJAR_GROUP_ID_1: com.example.openshift
KIE_CONTAINER_KJAR_ARTIFACT_ID_1: example_workflow
KIE_CONTAINER_KJAR_VERSION_1: 1.0
KIE_CONTAINER_REDIRECT_ENABLED: true
```

or, as represented in XML format:

```
<kie-server-state>
  <containers>
    <container>
      <containerId>df729302a0b7293c0729384710dd82a1</containerId>
      <releaseId>
        <groupId>com.example.openshift</groupId>
        <artifactId>example_workflow</artifactId>
        <version>1.0</version>
      </releaseId>
      <status>STARTED</status>
      <configItems/>
      <messages/>
    </container>
    <container>
      <containerId>01932fc2931b02cb042ab29d9fc82a8a</containerId>
      <releaseId>
        <groupId>com.example.openshift</groupId>
        <artifactId>example_workflow</artifactId>
        <version>1.1</version>
      </releaseId>
      <status>STARTED</status>
      <configItems/>
      <messages/>
    </container>
  </containers>
</kie-server-state>
```

IMPORTANT

To deploy multiple concurrent versions, the **KIE_CONTAINER_REDIRECT_ENABLED** variable must be set to **true**. This variable defaults to **true** and only needs to be explicitly included in the **.s2i/environment** file if setting to **false**.

The **KIE_CONTAINER_REDIRECT_ENABLED** variable enables override of the container ID. When set to **true**, a unique md5 sum hash is generated from the *<alias>=<group_id>:<artifact_id>:<version>* for each version of the application. It also enables alias redirection so that client requests using the deployment alias are redirected to the container of the correct version.

If set to **false**, the deployment alias is used as the container ID and multiple concurrent versions are not possible. If multiple versions of an application are specified for **KIE_CONTAINER_DEPLOYMENT**, and **KIE_CONTAINER_REDIRECT_ENABLED** is set to **false**, only the latest version of the application will be deployed and alias redirection will be disabled.

Changing the **KIE_CONTAINER_REDIRECT_ENABLED** variable in the **.s2i/environment** file of a running application generates a new container ID for the running application, which may make it incompatible with any clients using the old container ID.

5.3. CONTAINER ID

The container ID is an md5 sum hash generated from the *<alias>=<group_id>:<artifact_id>:<version>* of the application, and is used for client communication. In the case of multiple versions, each version of the application will have a unique container ID, but share the deployment alias name.

5.4. ADDING, OVERRIDING, OR UPDATING MULTIPLE VERSIONS OF THE APPLICATION

If an application has already been deployed, use the **KIE_CONTAINER_DEPLOYMENT_OVERRIDE** variable in the **.s2i/environment** file, and specify the *<alias>=<group_id>:<artifact_id>:<version>* for each version of the application to override the **KIE_CONTAINER_DEPLOYMENT** variable in the json application template. This is useful for preserving older versions of an application that are still in use.

For example, The *RulesTest* application example:

```
KIE_CONTAINER_DEPLOYMENT=RulesTest=com.example.openshift:example_workflow:1.0
```

To maintain this version of the application, but to add an updated version, update the **.s2i/environment** file:

```
KIE_CONTAINER_DEPLOYMENT_OVERRIDE=RulesTest=com.example.openshift:example_workflow
:1.0|RulesTest=com.example.openshift:example_workflow:1.1
```

See Example Workflow: Deploying an Updated Version Concurrently with Original Application for an example on deploying an updated application alongside the older version.

5.5. REQUEST TARGETING FOR MULTIPLE VERSIONS

In most cases, clients must target a particular container by name to execute server-side functions. This can be done by specifying the full deployment name, the container ID hash, or the deployment alias.

For example:

- Full Deployment Name: *RulesTest=com.example.openshift:example_workflow:1.0*
- Container ID Hash: *df729302a0b7293c0729384710dd82a1*
- Deployment Alias: *RulesTest*

Specifying either the full deployment name or the container ID targets the appropriate container. Specifying the deployment alias, which is used by all the containers in the KIE server, requires a multi-stage resolution process to target the correct version container.

5.6. ALIAS REDIRECTION

In a multi-version deployment, all applications share the same deployment alias. Requests that use the deployment alias of the application require a resolution process in order to redirect the request to the container of the correct version.

Resolution Process Hierarchy

The multi-stage resolution process depends on the method invoked by the client, and the ID associated with the request:

Process Hierarchy (in descending order):

1. Conversation ID
2. Default Container ID

Clients

Multiple clients can be used to invoke the server, depending on the client interaction type:

Client	Interaction
KIE interaction	org.kie.server.client.KieServicesClient
Decision Server interaction	org.kie.server.client.RuleServicesClient

Conversation ID

A conversation represents interactions between KIE Services java clients and the server. When a client initiates a conversation, the response from the server includes an encoded multi-part heading. The client will then use this heading in subsequent requests to the server. This conversation header contains the conversation ID, which is used by the Servlet Filter in the REST interface, or the EJB Interceptor in the JMS interface, to determine the correct version of the application to invoke.

Default Container ID

If a specific container ID cannot be resolved, the default container ID is determined as the application with the latest version (based on *<alias>=<group_id>:<artifact_id>:<version>*).

CHAPTER 6. RUNNING AND CONFIGURING REALTIME DECISION SERVER FOR OPENSHIFT

You can make changes to Realtime Decision Server for OpenShift configuration in the image using either the S2I templates, or by using a modified Realtime Decision Server for OpenShift.

6.1. USING REALTIME DECISION SERVER FOR OPENSHIFT SOURCE-TO-IMAGE (S2I) PROCESS

The recommended method to run and configure Realtime Decision Server for OpenShift is to use the OpenShift S2I process together with the application template parameters and environment variables.

The S2I process for Realtime Decision Server for OpenShift works as follows:

1. If there is a ***pom.xml*** file in the source repository, a Maven build is triggered with the contents of **$MAVEN_ARGS** environment variable.
    - By default, the **package** goal is used with the **openshift** profile, including the system properties for skipping tests (**-DskipTests**) and enabling the Red Hat GA repository (**-Dcom.redhat.xpaas.repo.redhatga**).
2. The results of a successful Maven build are installed into the local Maven repository, ***/home/jboss/.m2/repository/***, along with all dependencies for offline usage. Realtime Decision Server for OpenShift will load the created kjars from this local repository.
    - In addition to kjars resulting from the Maven build, any kjars found in the deployments source directory will also be installed into the local Maven repository. Kjars do not end up in the ***EAP_HOME/standalone/deployments/*** directory.
3. Any JAR (that is not a kjar), WAR, and EAR in the ***deployments*** source repository directory will be copied to the ***EAP_HOME/standalone/deployments*** directory and subsequently deployed using the JBoss EAP deployment scanner.
4. All files in the ***configuration*** source repository directory are copied to ***EAP_HOME/standalone/configuration***.

    NOTE

    If you want to use a custom JBoss EAP configuration file, it should be named ***standalone-openshift.xml***.
5. All files in the ***modules*** source repository directory are copied to ***EAP_HOME/modules***.

Refer to the Artifact Repository Mirrors section for additional guidance on how to instruct the S2I process to utilize the custom Maven artifacts repository mirror.

6.2. BINARY BUILDS

To deploy existing applications on OpenShift, you can use the binary source capability.

Prerequisite:

A. **Get the application archive or build the application locally.**
The following example uses both the hellorules and hellorules-client quickstarts.

- Clone the source code.

```
$ git clone https://github.com/jboss-openshift/openshift-quickstarts.git
```

- Configure the Red Hat JBoss Middleware Maven repository .
- Build the application – both the **hellorules** and **hellorules-client** quickstarts.

> **NOTE**
>
> The output of **mvn clean package** command below has been shortened to contain just selected information.

```
$ cd openshift-quickstarts/decisionserver/
```

```
$ mvn clean package
[INFO] Scanning for projects...
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] OpenShift Quickstarts: Decision Server: Hello Rules
[INFO] OpenShift Quickstarts: Decision Server: Hello Rules - Client
[INFO] OpenShift Quickstarts: Decision Server: Parent
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building OpenShift Quickstarts: Decision Server: Hello Rules 1.4.0.Final
[INFO] ------------------------------------------------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] Building OpenShift Quickstarts: Decision Server: Hello Rules - Client 1.4.0.Final
[INFO] ------------------------------------------------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] OpenShift Quickstarts: Decision Server: Hello Rules  SUCCESS [  0.844 s]
[INFO] OpenShift Quickstarts: Decision Server: Hello Rules - Client SUCCESS [  7.446 s]
[INFO] OpenShift Quickstarts: Decision Server: Parent ..... SUCCESS [  0.002 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.286 s
[INFO] Finished at: 2017-06-27T16:49:25+02:00
[INFO] Final Memory: 49M/502M
[INFO] ------------------------------------------------------------------------
```

B. **Prepare the directory structure on the local file system.**
Application archives in the **deployments/** subdirectory of the main binary build directory are copied directly to the standard deployments folder of the image being built on OpenShift. For the application to deploy, the directory hierarchy containing the web application data must be correctly structured.

Create main directory for the binary build on the local file system and **deployments/** subdirectory within it. Copy both the previously built JAR archive for the **hellorules** quickstart, and WAR archive for the **hellorules-client** quickstart to the **deployments/** subdirectory:

```
decisionserver]$ ls
hellorules  hellorules-client  pom.xml
```

```
$ mkdir -p ocp/deployments
```

```
$ cp hellorules/target/decisionserver-hellorules-1.4.0.Final.jar ocp/deployments/
```

```
$ cp hellorules-client/target/decisionserver-hellorules-client-1.4.0.Final.war ocp/deployments/
```

NOTE

Location of the standard deployments directory depends on the underlying base image, that was used to deploy the application. See the following table:

Table 6.1. Standard Location of the Deployments Directory

Name of the Underlying Base Image(s)	Standard Location of the Deployments Directory
EAP for OpenShift 6.4 and 7.0	***$JBOSS_HOME/standalone/deployments***
Java S2I for OpenShift	***/deployments***
JWS for OpenShift	***$JWS_HOME/webapps***

Perform the following steps to run application consisting of binary input on OpenShift:

1. Login into OpenShift instance.

```
$ oc login
```

2. Create a new project.

```
$ oc new-project ds-bin-demo
```

3. (Optional) Identify the image stream for the particular image.

```
$ oc get is -n openshift | grep ^jboss-decisionserver | cut -f1 -d ' '
jboss-decisionserver62-openshift
jboss-decisionserver63-openshift
```

NOTE

Since the images from **jboss-decisionserver62-openshift** image stream are obsolete, we will use **jboss-decisionserver63-openshift** below.

4. Create new binary build, specifying image stream and application name.

   NOTE

   You can change the default user name and password to access the REST interface of the KIE server by providing custom values for ***KIE_SERVER_USER*** and ***KIE_SERVER_PASSWORD*** environment variables.

```
$ oc new-build --binary=true \
--name=ds-hr-app \
--image-stream=jboss-decisionserver63-openshift \
-e KIE_SERVER_USER=kieserveruser \
-e KIE_SERVER_PASSWORD=kieserverPwd1!
--> Found image 4a6c0ce (5 weeks old) in image stream "jboss-decisionserver63-openshift"
in project "openshift" under tag "latest" for "jboss-decisionserver63-openshift"

    JBoss BRMS Realtime Decision Server 6.3
    ---------------------------------------
    Platform for executing business rules on JBoss BRMS Realtime Decision Server 6.3.

    Tags: builder, decisionserver, decisionserver6

    * A source build using binary input will be created
      * The resulting image will be pushed to image stream "ds-hr-app:latest"
      * Use 'start-build --from-dir=DIR|--from-repo=DIR|--from-file=FILE' to trigger a new build
      * WARNING: a binary build was created, you must specify one of --from-dir|--from-file|--
from-repo when starting builds

--> Creating resources with label build=ds-hr-app ...
    imagestream "ds-hr-app" created
    buildconfig "ds-hr-app" created
--> Success
```

5. Start the binary build. Instruct **oc** executable to use main directory of the binary build we created in previous step as the directory containing binary input for the OpenShift build.

   NOTE

   The output of the next command has been shortened for brevity.

```
$ oc start-build ds-hr-app --from-dir=./ocp/ --follow
Uploading directory "ocp" as binary input for the build ...
build "ds-hr-app-1" started
Receiving source from STDIN as archive ...

Copying all war artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all ear artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all rar artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all jar artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all war artifacts from /home/jboss/source/deployments directory into
```

```
/opt/eap/standalone/deployments for later deployment...
'/home/jboss/source/deployments/decisionserver-hellorules-client-1.4.0.Final.war' ->
'/opt/eap/standalone/deployments/decisionserver-hellorules-client-1.4.0.Final.war'
Copying all ear artifacts from /home/jboss/source/deployments directory into
/opt/eap/standalone/deployments for later deployment...
Copying all rar artifacts from /home/jboss/source/deployments directory into
/opt/eap/standalone/deployments for later deployment...
Copying all jar artifacts from /home/jboss/source/deployments directory into
/opt/eap/standalone/deployments for later deployment...
'/home/jboss/source/deployments/decisionserver-hellorules-1.4.0.Final.jar' ->
'/opt/eap/standalone/deployments/decisionserver-hellorules-1.4.0.Final.jar'
/opt/eap/standalone/deployments/decisionserver-hellorules-1.4.0.Final.jar is a kjar
...
INFO: org.openshift.quickstarts:decisionserver-hellorules:1.4.0.Final verified.


Pushing image 172.30.202.111:5000/ds-bin-demo/ds-hr-app:latest ...
Pushed 6/9 layers, 67% complete
Pushed 7/9 layers, 78% complete
Pushed 8/9 layers, 89% complete
Pushed 9/9 layers, 100% complete
Push successful
```

6. Create a new OpenShift application based on the build.

```
$ oc new-app ds-hr-app
--> Found image c2c182e (48 seconds old) in image stream ds-hr-app under tag "latest" for
"ds-hr-app"

    ds-bin-demo/ds-hr-app-2:ea504dd7
    --------------------------------
    Platform for executing business rules on JBoss BRMS Realtime Decision Server 6.3.

    Tags: builder, decisionserver, decisionserver6

    * This image will be deployed in deployment config "ds-hr-app"
    * Ports 8080/tcp, 8443/tcp, 8778/tcp will be load balanced by service "ds-hr-app"
      * Other containers can access this service through the hostname "ds-hr-app"

--> Creating resources with label app=ds-hr-app ...
    deploymentconfig "ds-hr-app" created
    service "ds-hr-app" created
--> Success
    Run 'oc status' to view your app.
```

7. Expose the service as route.

```
$ oc get svc -o name
service/ds-hr-app
```

```
$ oc expose svc/ds-hr-app
route "ds-hr-app" exposed
```

8. Access the application.

You can get the list of available query string arguments of the **hellorules** application by accessing the URL **http://ds-hr-app-ds-bin-demo.openshift.example.com/hellorules/**.

Run the **hellorules-client** servlet using the URL **http://ds-hr-app-ds-bin-demo.openshift.example.com/hellorules?command=runLocal**.

NOTE

You may verify the current KIE server state by accessing dedicated **server/** page of the REST API: **http://ds-hr-app-ds-bin-demo.openshift.example.com/kie-server/services/rest/server/**. Use aforementioned user name and password to access this page (or any REST API method of the server in general).

6.3. USING A MODIFIED DECISION SERVER XPAAS IMAGE

An alternative method is to make changes to the image, and then use that modified image in OpenShift. The templates currently provided, along with the interfaces they support, are listed below:

Table 6.2. Provided Templates

Template Name	Supported Interfaces
decisionserver63-basic-s2i.json	http-rest, jms-hornetq
decisionserver63-https-s2i.json	http-rest, https-rest, jms-hornetq
decisionserver63-amq-s2i.json	http-rest, https-rest, jms-activemq

You can run Realtime Decision Server for OpenShift in Docker, make the required configuration changes using the JBoss EAP Management CLI (***EAP_HOME/bin/jboss-cli.sh***) included in Realtime Decision Server for OpenShift, and then commit the changed container as a new image. You can then use that modified image in OpenShift.

IMPORTANT

It is recommended that you do not replace the OpenShift placeholders in the JBoss EAP xPaaS configuration file, as they are used to automatically configure services (such as messaging, datastores, HTTPS) during a container's deployment. These configuration values are intended to be set using environment variables.

NOTE

Ensure that you follow the guidelines for creating images.

PART IV. TUTORIALS

CHAPTER 7. EXAMPLE WORKFLOW: DEPLOYING RED HAT JBOSS BRMS APPLICATION ON REALTIME DECISION SERVER FOR OPENSHIFT

This tutorial prepares a Red Hat JBoss BRMS application to be deployed as Realtime Decision Server for OpenShift. The Red Hat JBoss BRMS application may require modification to be deployed as an image.

Preparing the Application
The Red Hat JBoss BRMS project must be configured to have a stateless knowledge session, use a remote repository, and have defined KIE container deployment.
Refer to the Red Hat JBoss BRMS User Guide for more information on any of these tasks.

1. Log in to the Red Hat JBoss BRMS console and edit the project settings in the **Project Explorer**.

2. Click **Open Project Editor** and in the **Project Settings**:

   a. Under **Knowledge bases and sessions**, ensure that the **Knowledge Session** is set to **Stateless**. OpenShift does not support stateful sessions on KIE servers.

   b. Using **Repository View**, ensure the **pom.xml** is configured to use a remote repository by containing xml similar to the following:

   ```
   ...
   <distributionManagement>
     <repository>
       <id>deployment</id>
       <name>OpenShift Maven repo</name>
       <url>http://maven.example/content/repo/deployments/</url>
     </repository>

     <snapshotRepository>
       <id>deployment</id>
       <name>OpenShift Maven repo</name>
       <url>http://maven.example.xas/content/repo/snapshots/</url>
     </snapshotRepository>
   </distributionManagement>
   ...
   ```

   For more information, see the Red Hat JBoss BRMS Administration and Configuration Guide.

3. In the application’s repository, ensure the **settings.xml** and the **.s2i/environment** files define the Maven repository and the KIE container deployment respectively:

   a. The Maven repository should be defined in the **settings.xml** so that OpenShift can download the application artefacts. It should contain xml similar to the following:

   ```
   ...
   <profiles>
     <profile>
       <id>openshift-mirror-repositories</id>
       <repositories>
         <repository>
           <id>openshift-mirror</id>
   ```

```
      <url>http://maven.example/content/group/public/</url>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <id>openshift-mirror</id>
      <url>http://maven.example/content/group/public/</url>
    </pluginRepository>
  </pluginRepositories>
 </profile>
</profiles>
...
```

For more information, see the Red Hat JBoss BRMS Installation Guide .

b. The **.s2i/environment** file must define the KIE container deployment, including which KIE jars to use and the location from which to retrieve them. The pod name is derived from the deployment alias, which is defined as *DemoContainer* in this example:

```
KIE_CONTAINER_DEPLOYMENT_OVERRIDE=DemoContainer=com.example.openshift:
example_workflow:1.0
```

7.1. PREPARING DECISION SERVER DEPLOYMENT

1. Create a new project:

```
$ oc new-project rds-app-demo
```

2. Create a service account to be used for the deployment of the Decision Server application:

```
$ oc create serviceaccount rds-service-account
```

3. Add the view role to the service account. This enables the service account to view all the resources in the rds-app-demo namespace, which is necessary for managing the cluster.

```
$ oc policy add-role-to-user view system:serviceaccount:rds-app-demo:rds-service-account
```

4. The Decision Server template requires an SSL keystore and a JGroups keystore.
These keystores are expected even if the application will not use https.
This example uses ‘keytool’, a package included with the Java Development Kit, to generate self-signed certificates for these keystores. The following commands will prompt for passwords.

a. Generate a secure key for the SSL keystore:

```
$ keytool -genkeypair -alias https -storetype JKS -keystore keystore.jks
```

b. Generate a secure key for the JGroups keystore:

```
$ keytool -genseckey -alias jgroups -storetype JCEKS -keystore jgroups.jceks
```

5. Use the SSL and JGroup keystore files to create the secret for the project:

```
$ oc create secret generic rds-app-secret --from-file=jgroups.jceks --from-file=keystore.jks
```

6. Add the secret to the service account created earlier:

```
$ oc secret add sa/rds-service-account secret/rds-app-secret
```

7.2. DEPLOYMENT

1. Log in to the OpenShift web console and select the *rds-app-demo* project space.
2. Click **Add to Project** to list all of the default image streams and templates.
3. Use the **Filter by keyword** search bar to limit the list to those that match **decisionserver**. You may need to click **See all** to show the desired application template.
4. Select and configure the desired template and click **Deploy**.

During the build, the Maven repository is downloaded and build into the container so that no additional packages or dependencies are downloaded at runtime.

The application is available once the pod is running. To connect to the Decision Server web console, navigate to the pod and click **Open Java Console** button.

CHAPTER 8. EXAMPLE WORKFLOW: DEPLOYING AN UPGRADED VERSION CONCURRENTLY WITH ORIGINAL APPLICATION

This example workflow follows on from Example Workflow: Deploying Red Hat JBoss BRMS Application on Decision Server xPaaS, in which the *1.0* version of the *example_workflow* artifact was deployed with a deployment alias of *DemoContainer*. This example deploys a *1.1* version of the of the *example_workflow* artifact alongside the *1.0* version so that both versions of the *example_workflow* artifact are running simultaneously, both with the *DemoContainer* deployment alias.

1. Update the repository with the new version of the server.
2. Edit the **.s2i/environment** file for the application:
   a. Change the **KIE_CONTAINER_DEPLOYMENT** variable to **KIE_CONTAINER_DEPLOYMENT_OVERRIDE**
   b. Add the new version to the end of the value string, separated from the older version with a pipe.

      ```
      KIE_CONTAINER_DEPLOYMENT_OVERRIDE=DemoContainer=com.example.openshift:
      example_workflow:1.0|DemoContainer=com.example.openshift:example_workflow:1.1
      ```
3. Save the changes.
4. If the project has GitHub Webhooks configured, the new version will be deployed automatically alongside the older running applicaiton. Otherwise it can be manually built:

   ```
   $ oc start-build rds-app-demo
   ```

Once the build has completed, the two different versions of the application will be running simultaneously using the same deployment alias. See Request Targeting for Multiple Versions for more information on how client requests are redirected to the correct version of the application.

CHAPTER 9. EXAMPLE WORKFLOW: DEPLOYING RED HAT JBOSS BRMS APPLICATION ON OPENSHIFT WITH WEBHOOKS ENABLED FOR AUTOMATIC APPLICATION UPDATES

This workflow details how to configure Red Hat JBoss BRMS, GitHub, and OpenShift to have your configuration changes automatically push to OpenShift. This example covers:

- Setting up a forked GitHub repository
- Cloning the repository
- Creating a hook in JBoss Decision Server to automatically update the GitHub repository
- Modifying the example JBoss Decision Server rules
- Creating a Decision Service on OpenShift
- Improving OpenShift build times using Maven
- Integrating the Maven Proxy
- Configuring the OpenShift webhook to automatically update the decision service OpenShift whenever a code change occurs in the GitHub repository
- Testing the configured service and hooks

NOTE

Make sure you are running Red Hat JBoss BRMS on your local machine.

9.1. FORKING THE REPOSITORY

1. Visit the Decision Server example page while you are logged in to GitHub.
2. Fork the repository.
   You are redirected to your new fork.
3. Copy the HTTPS clone URL for your fork.

This Decision Server example receives a name, and if it matches the user name specified as **master** in the rules file, then the user is recognized and greeted as the master user. If the name does not match, then the user is recognized as an intruder.

9.2. CLONING THE REPOSITORY

From the Red Hat JBoss BRMS workbench:

1. From the File Explorer, click **Authoring → Administration**.
2. Click **Repositories → Clone repository**.
3. Type the **Repository Name** **decision-services**.

4. Select an **Organizational Unit**.

5. Type in the HTTPS clone URL of your forked Git repository:
   https://github.com/*<Your_Github_Username>*/decisionserver.git

6. Click **Clone**.
   Once cloned, the repository displays the commit history.

9.3. CREATING A HOOK TO AUTOMATE GITHUB UPDATES

To make Red Hat JBoss BRMS automatically update your GitHub repository any time a file in this project changes:

NOTE

You must have SSH key access configured for GitHub before following these steps.

1. From the command line, navigate into the /**.niogit** directory in the project you forked earlier:

   ```
   $ cd EAPHOME/bin/.niogit/decision-services.git
   ```

   The path above is the default, which may differ depending on where the workbench has been configured to store its data. This location is set using the **org.uberfire.nio.git.dir** system property.

2. Set the remote URL for this project:

   ```
   $ git remote set-url origin git@github.com:/decisionserver
   ```

3. Navigate into the hooks directory:

   ```
   $ cd hooks
   ```

4. Create a simple post-commit file:

   ```
   $ touch post-commit
   ```

5. Edit the file and type the following:

   ```
   #!/bin/sh

   git push origin master
   ```

6. Save your changes and exit the file.

7. Change the permissions on the file to allow Red Hat JBoss BRMS the access it requires:

   ```
   $ chmod 777 post-commit
   ```

   The hook is now configured, meaning that any change to the files in this Red Hat JBoss BRMS project will automatically update your forked **decisionserver** GitHub repository.

9.4. MODIFYING THE EXAMPLE DECISION SERVER RULES

From the Red Hat JBoss BRMS workbench:

1. Click **Authoring → Project authoring**.
2. Under **DRL**, click to load the **HelloRules.drl** file:

```
package org.openshift.quickstarts.decisionserver.hellorules

query "get greeting"()
  greeting : Greeting()
end

rule "greet master"
  when
    person : Person( name == "john")
  then
    String salutation = "Hello " + person.getName() + "! What can I help you with today?";
    insert(new Greeting(salutation));
end
rule "greet strangers"
  when
    person : Person(name != "john")
  then
    String salutation = "Hey there " + person.getName() + ". I don't think I know you yet!";
    insert(new Greeting (salutation));
end
```

3. Modify the lines with **john** by replacing them with your user name.
4. Click **Save**, type a check in comment, and click **Save** again.
   The hook you created earlier will automatically update your forked GitHub repository with these saved changes.

9.5. CREATING A DECISION SERVICE ON OPENSHIFT

From the OpenShift web console:

1. Log in using the username and password recommended to you by your administrator.
2. To create a new project, click **New Project**.
3. Type a unique name, display name, and description for the new project.
4. Click **Create**.
   The web console's welcome screen loads.
5. Click **Add to Project**.
6. In the **Filter by keyword** field, start typing **decision** to see the available xPaaS templates related to Decision Server.
7. Click the **decisionserver63-basic-s2i** template.

8. In the **Parameters** section, change the **KIE_SERVER_PASSWORD** to the password to access the KIE Server REST or JMS interface.

9. Change the **SOURCE_REPOSITORY_URL** to the Git source URI for your forked repository. For example:

   ```
   https://github.com/<your_github_username>/decisionserver.git
   ```

10. Change the **SOURCE_REPOSITORY_REF** to **master**.

11. Change the **CONTEXT_DIR** to **greeting**.

12. Scroll to the bottom of the page and click **Create**.

While your application builds, you can click **View Log** from the Overview page to see the build progress.

9.6. IMPROVING BUILD TIME USING MAVEN

Follow the details in this OpenShift blog post to configure the Maven proxy, which improves the build times of java builds on OpenShift.

9.7. INTEGRATING THE MAVEN PROXY

To change the build configuration so that it uses the Maven proxy, complete the following from the OpenShift web console:

1. Click **Browse** → **Builds** → ***<your_application>***

2. Click the three vertical dots next to **Start Build** and then click **Edit (Raw)**.

3. Add the **MAVEN_MIRROR_URL** environment variable below the **KIE_CONTAINER_DEPLOYMENT** variable:

   ```
   strategy
    sourceStrategy:
     env:
      -
       name: KIE_CONTAINER_DEPLOYMENT
       value: 'HelloRulesContainer=org.openshift.quickstarts:decisionserver-hellorules:1.2.0.Final'
      -
       name: MAVEN_MIRROR_URL
       value: 'http://nexus-ci.cloudapps.bos.openshift3roadshow.com/content/groups/public/'
   ```

   The value for **MAVEN_MIRROR_URL** can be found in Maven by viewing the repositories, then copying the path for the Public Repositories group.

4. Click **Save**.

5. Click the **Configuration** tab of your build to verify that **MAVEN_MIRROR_URL** is actively listed under Environment Variables.

Now that you have Maven configured for this OpenShift project, the build process will be shorter for all future builds. This is because subsequent builds only need to download updated files, which are then combined with the previously loaded files.

9.8. TEST THE SERVICE

After integrating the Maven proxy, you can test that service is working and see how quickly the build process completes compared to previous builds. From the OpenShift web console:

1. Click **Browse → Builds → *<your_application>***
2. Click **Start Build**.
3. In the list at the bottom of the screen, click the new build you just started.
4. Click the **Logs** tab, then click **Follow**.
5. Verify that the new build is using the new Maven proxy to download locally by finding the line in the log that references **Downloading**. For example:

   ```
   I0130 12:32:25.664594     1 sti.go:492] Downloading: http://nexus-
   ci.cloudapps.openshift.com/content/groups/public/org/kie/kie-maven-plugin/6.3.0.Final-
   redhat-5/kie-maven-plugin-6.3.0.Final-redhat-5.pom
   ```

6. When the build is complete, you can check the new build time against the previous build by clicking **Browse → Builds → *<your_application>*** and viewing the summary. The newest build will be considerably shorter with the Maven proxy in use.
7. Click **Overview** to see the status of the pod. It displays a **Not Ready** status while it is checked with readiness probes.
8. Click **Browse → Pods** to follow its progress. The status of the **Containers Ready** column will change to **1/1** when the pod has passed the readiness probes.

9.9. CONFIGURE THE OPENSHIFT WEBHOOK

From the OpenShift web console:

1. Click the **Browse** tab, then click **Builds**.
2. Click your build name, then click the **Configuration** tab.
3. Click the copy icon next to **GitHub webhook URL** to copy your webhook payload URL.
4. Navigate to your forked repository on GitHub, then click **Settings**.
5. Click **Webhooks & Services**.
6. Click **Add webhook**.
7. Paste your webhook URL into the **Payload URL** field.
8. Click **Disable SSL verification**, then confirm it in the pop-up window.
9. Click **Add webhook** to save.

Github pings the OpenShift server to ensure communication is successful. A green check mark next to the webhook URL signifies that it is configured correctly. Hover your cursor over the check mark to view the status of the last ping.

The next time you push a code change to your forked repository, your application will automatically rebuild.

9.10. TESTING THE CONFIGURED HOOKS

From the Red Hat JBoss BRMS workbench:

1. Load the **HelloRules.drl** file:

```
package org.openshift.quickstarts.decisionserver.hellorules

query "get greeting"()
   greeting : Greeting()
end

rule "greet master"
   when
      person : Person( name == "john")
   then
      String salutation = "Hello " + person.getName() + "! What can I help you with today?";
      insert(new Greeting(salutation));
end
rule "greet strangers"
   when
      person : Person(name != "john")
   then
      String salutation = "Hey there " + person.getName() + ". I don't think I know you yet!";
      insert(new Greeting (salutation));
end
```

2. Modify the String salutation line by changing **At your service my master** to something else.

3. Click **Save**, type a check-in comment, and click **Save** again.

The hook that you created earlier updates your forked GitHub repository, and then the GitHub webhook triggers a new build in OpenShift.

With this configuration, you need only save your configuration changes on the Red Hat JBoss BRMS workbench, and the rest of the process is completely automated.

PART V. REFERENCE

CHAPTER 10. ARTIFACT REPOSITORY MIRRORS

A repository in Maven holds build artifacts and dependencies of various types (all the project jars, library jar, plugins or any other project specific artifacts). It also specifies locations from where to download artifacts from, while performing the S2I build. Besides using central repositories, it is a common practice for organizations to deploy a local custom repository (mirror).

Benefits of using a mirror are:

- Availability of a synchronized mirror, which is geographically closer and faster.
- Ability to have greater control over the repository content.
- Possibility to share artifacts across different teams (developers, CI), without the need to rely on public servers and repositories.
- Improved build times.

Often, a repository manager can serve as local cache to a mirror. Assuming that the repository manager is already deployed and reachable externally at ***http://10.0.0.1:8080/repository/internal/***, the S2I build can then use this manager by supplying the **MAVEN_MIRROR_URL** environment variable to the build configuration of the application as follows:

1. Identify the name of the build configuration to apply **MAVEN_MIRROR_URL** variable against:

   ```
   oc get bc -o name
   buildconfig/ds
   ```

2. Update build configuration of **ds** with a **MAVEN_MIRROR_URL** environment variable

   ```
   oc env bc/ds MAVEN_MIRROR_URL="http://10.0.0.1:8080/repository/internal/"
   buildconfig "ds" updated
   ```

3. Verify the setting

   ```
   oc env bc/ds --list
   # buildconfigs ds
   MAVEN_MIRROR_URL=http://10.0.0.1:8080/repository/internal/
   ```

4. Schedule new build of the application

NOTE

During application build, you will notice that Maven dependencies are pulled from the repository manager, instead of the default public repositories. Also, after the build is finished, you will see that the mirror is filled with all the dependencies that were retrieved and used during the build.

CHAPTER 11. APPLICATION TEMPLATE PARAMETERS

Variable	Description
APPLICATION_NAME	The name for the application (required).
KIE_CONTAINER_DEPLOYMENT	The KIE Containers to deploy (required). Example: containerId=groupId:artifactId:version
MYSQL_LOWER_CASE_TABLE_NAMES	Sets how the table names are stored and compared.
AMQ_SECRET	The name of a secret containing SSL related files. If no value is specified, a random password is generated.
SOURCE_REPOSITORY_URL	Git source URI for application.
SOURCE_REPOSITORY_REF	Git branch/tag reference.
CONTEXT_DIR	Path within Git project to build; empty for root project directory.
KIE_SERVER_USER	The user name to access the KIE Server REST or JMS interface.
KIE_SERVER_PASSWORD	The password to access the KIE Server REST or JMS interface. Must be different than username; must not be root, admin, or administrator; must contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), and 1 non-alphanumeric symbol(s).

CHAPTER 12. ENDPOINTS

Clients can access Realtime Decision Server for OpenShift via multiple endpoints; by default the provided templates include support for REST, HornetQ, and ActiveMQ.

12.1. REST

Clients can use the REST API in various ways:

12.1.1. Browser

1. Current server state: http://host/kie-server/services/rest/server
2. List of containers: http://host/kie-server/services/rest/server/containers
3. Specific container state: http://host/kie-server/services/rest/server/containers/HelloRulesContainer

12.1.2. Java

```
// HelloRulesClient.java
KieServicesConfiguration config = KieServicesFactory.newRestConfiguration(
  "http://host/kie-server/services/rest/server", "kieserverUser", "kieserverPassword");
config.setMarshallingFormat(MarshallingFormat.XSTREAM);
RuleServicesClient client =
  KieServicesFactory.newKieServicesClient(config).getServicesClient(RuleServicesClient.class);
ServiceResponse<String> response = client.executeCommands("HelloRulesContainer",
myCommands);
```

12.1.3. Command Line

```
# request.sh
#!/bin/sh
curl -X POST \
  -d @request.xml \
  -H "Accept:application/xml" \
  -H "X-KIE-ContentType:XSTREAM" \
  -H "Content-Type:application/xml" \
  -H "Authorization:Basic a2llc2VydmVyOmtpZXNlcnZlcjEh" \
  -H "X-KIE-ClassType:org.drools.core.command.runtime.BatchExecutionCommandImpl" \
http://host/kie-server/services/rest/server/containers/instances/HelloRulesContainer
```

```
<!-- request.xml -->
<batch-execution lookup="HelloRulesSession">
  <insert>
    <org.openshift.quickstarts.decisionserver.hellorules.Person>
      <name>errantepiphany</name>
    </org.openshift.quickstarts.decisionserver.hellorules.Person>
  </insert>
  <fire-all-rules/>
  <query out-identifier="greetings" name="get greeting"/>
</batch-execution>
```

12.2. JMS

Client can also use the Java Messaging Service, as demonstrated below:

12.2.1. Java (HornetQ)

```
// HelloRulesClient.java
Properties props = new Properties();
props.setProperty(Context.INITIAL_CONTEXT_FACTORY,
  "org.jboss.naming.remote.client.InitialContextFactory");
props.setProperty(Context.PROVIDER_URL, "remote://host:4447");
props.setProperty(Context.SECURITY_PRINCIPAL, "kieserverUser");
props.setProperty(Context.SECURITY_CREDENTIALS, "kieserverPassword");
InitialContext context = new InitialContext(props);
KieServicesConfiguration config =
  KieServicesFactory.newJMSConfiguration(context, "hornetqUser", "hornetqPassword");
config.setMarshallingFormat(MarshallingFormat.XSTREAM);
RuleServicesClient client =
  KieServicesFactory.newKieServicesClient(config).getServicesClient(RuleServicesClient.class);
ServiceResponse<String> response = client.executeCommands("HelloRulesContainer",
myCommands);
```

12.2.2. Java (ActiveMQ)

```
// HelloRulesClient.java
props.setProperty(Context.INITIAL_CONTEXT_FACTORY,
  "org.apache.activemq.jndi.ActiveMQInitialContextFactory");
props.setProperty(Context.PROVIDER_URL, "tcp://host:61616");
props.setProperty(Context.SECURITY_PRINCIPAL, "kieserverUser");
props.setProperty(Context.SECURITY_CREDENTIALS, "kieserverPassword");
InitialContext context = new InitialContext(props);
ConnectionFactory connectionFactory = (ConnectionFactory)context.lookup("ConnectionFactory");
Queue requestQueue = (Queue)context.lookup("dynamicQueues/queue/KIE.SERVER.REQUEST");
Queue responseQueue =
(Queue)context.lookup("dynamicQueues/queue/KIE.SERVER.RESPONSE");
KieServicesConfiguration config = KieServicesFactory.newJMSConfiguration(
  connectionFactory, requestQueue, responseQueue, "activemqUser", "activemqPassword");
config.setMarshallingFormat(MarshallingFormat.XSTREAM);
RuleServicesClient client =
  KieServicesFactory.newKieServicesClient(config).getServicesClient(RuleServicesClient.class);
ServiceResponse<String> response = client.executeCommands("HelloRulesContainer",
myCommands);
```

CHAPTER 13. TROUBLESHOOTING

In addition to viewing the OpenShift logs, you can troubleshoot a running Decision Server xPaaS Image container by viewing its logs. These are outputted to the container's standard out, and are accessible with the following command:

```
$ oc logs -f <pod_name>
```

NOTE

By default, the OpenShift Decision Server xPaaS image does not have a file log handler configured. Logs are only sent to the container's standard out.

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on: Monday, May 13, 2019.