Red Hat JBoss BPM Suite 6.4

Red Hat JBoss BPM Suite Intelligent Process Server for OpenShift

Develop with Red Hat JBoss BPM Suite Intelligent Process Server for OpenShift

Last Updated: 2019-05-13

Red Hat JBoss BPM Suite 6.4 Red Hat JBoss BPM Suite Intelligent Process Server for OpenShift

Develop with Red Hat JBoss BPM Suite Intelligent Process Server for OpenShift

Legal Notice

Abstract

Guide to using Red Hat JBoss BPM Suite Intelligent Process Server for OpenShift

Table of Contents

PART I. INTRODUCTION

CHAPTER 1. WHAT IS THE RED HAT JBOSS BPM SUITE INTELLIGENT PROCESS SERVER?

Red Hat JBoss BPM Suite intelligent process server (IPS) for OpenShift provides a platform for executing business processes on JBoss BPMS Intelligent Process Server 6.3, which is a modular, standalone server component that can be used to instantiate and execute rules and processes. It exposes this functionality through REST, JMS and Java interfaces to client application.

Red Hat offers nine Intelligent Process Server application templates:

Template	Description
processserver63-basic-s2i	template provides HTTP and JMS (via HornetQ) interfaces with a simple H2 database.
processserver63-mysql-s2i	template provides HTTP and JMS (via HornetQ) interfaces with a simple MySQL database.
processserver63-mysql-persistent-s2i	template provides HTTP and JMS (via HornetQ) interfaces with a MySQL persistence volume.
processserver63-postgresql-s2i	template provides HTTP and JMS (via HornetQ) interfaces with a simple PostgreSQL database.
processserver63-postgresql-persistent-s2i	template provides HTTP and JMS (via HornetQ) interfaces with a PostgresQL persistence volume.
processserver63-amq-mysql-s2i	template provides HTTP, HTTPS, and JMS (via ActiveMQ) interfaces with a simple MySQL database.
processserver63-amq-mysql-persistent-s2i	template provides HTTP, HTTPS, and JMS (via ActiveMQ) interfaces with a MySQL persistence volume.
processserver63-amq-postgresql-s2i	template provides HTTP, HTTPS, and JMS (via ActiveMQ) interfaces with a simple PostgreSQL database.
processserver63-amq-postgresql-persistent-s2i	template provides HTTP, HTTPS, and JMS (via ActiveMQ) interfaces with a PostgresQL persistence volume.

PART II. BEFORE YOU BEGIN

CHAPTER 2. COMPARISON: INTELLIGENT PROCESS SERVER FOR OPENSHIFT AND INTELLIGENT PROCESS SERVER

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

2.1. FUNCTIONALITY DIFFERENCES FOR INTELLIGENT PROCESS SERVER FOR OPENSHIFT IMAGES

There are several major functionality differences between the regular release of Intelligent Process Server and the Intelligent Process Server for OpenShift:

- Intelligent Process Server for OpenShift extends the OpenShift EAP image, and any capabilities or limitations it has are also found in the Intelligent Process Server for OpenShift.
- Business Central is not included in Intelligent Process Server for OpenShift. To connect to the Intelligent Process Server web console, click the **Open Java Console** button in OpenShift 3.2.
- There is no support for authoring any content through the BPMS Console or API.
- There is no support for the **Singleton** strategy for maintaining a single instance of the **RuntimeEngine**.

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 INTELLIGENT PROCESS SERVER FOR OPENSHIFT

IMPORTANT

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

The Intelligent Process 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 Intelligent Process Server for OpenShift image and application templates to deploy new applications.

2.4. MANAGING INTELLIGENT PROCESS SERVER FOR OPENSHIFT

As Intelligent Process 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.

2.5. SECURITY IN INTELLIGENT PROCESS 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. USING INTELLIGENT PROCESS SERVER FOR OPENSHIFT IMAGE STREAMS AND APPLICATION TEMPLATES

The Red Hat xPaaS middleware images were automatically created during the installation of OpenShift along with the other default image streams and templates.

CHAPTER 4. DEPLOYMENT CONSIDERATIONS FOR INTELLIGENT PROCESS SERVER FOR OPENSHIFT

4.1. CREATING THE SERVICE ACCOUNT

Intelligent Process Server for OpenShift requires a service account for deployments. For multiple node deployments, the service account must have the **view** role enabled so that it can manage the various pods in the cluster. In addition, you will need to configure SSL to enable connections to Intelligent Process Server from outside of the OpenShift instance.

1. Create the service account:

```
$ echo '{"kind": "ServiceAccount", "apiVersion": "v1", "metadata": {"name": "<service-account-name>"}}' | oc create -f -
```

   OpenShift 3.2 users can use the following command to create the service account:

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

2. Add the **view** role to the service account:

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

4.2. CONFIGURING KEYSTORES

Intelligent Process 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 Intelligent Process 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.

4.3. 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.

Intelligent Process 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 JGroups keystore files to create a secret for the project:

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

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

4.4. 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 the Intelligent Process Server 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/<ips-secret-name>
```

4.5. CONFIGURING THE PROJECT REMOTE REPOSITORY

The project must be configured to use a remote repository so that Business Central 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 BPM Suite 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 BPM Suite 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=ApplicationTest=com.example.openshift:example_workflow
:1.0
```

CHAPTER 5. UPDATING PROCESSES

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

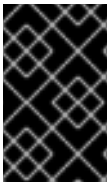
5.1. RECREATE UPDATE STRATEGY

Use the Recreate Update Strategy for the Intelligent Process Server 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 the Intelligent Process Server application templates.

NOTE

The Intelligent Process Server 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 Intelligent Process 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.

CHAPTER 6. 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=ApplicationTest=com.example.openshift:example_workflow:1.0|App
licationTest=com.example.openshift:example_workflow:1.1
```

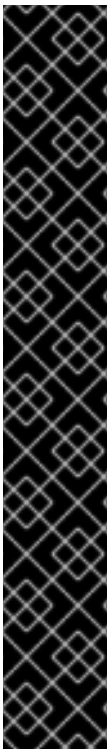
would create the following:

```
KIE_CONTAINER_DEPLOYMENT=ApplicationTest=com.example.openshift:example_workflow:1.0|App
licationTest=com.example.openshift:example_workflow:1.1
KIE_CONTAINER_DEPLOYMENT_ORIGINAL:
KIE_CONTAINER_DEPLOYMENT_OVERRIDE:
ApplicationTest=com.example.openshift:example_workflow:1.0|ApplicationTest=com.example.openshift
example_workflow:1.1
KIE_CONTAINER_DEPLOYMENT_COUNT: 2
KIE_CONTAINER_ID_0: be690712c7a5808a0696926088ff18b2
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: 72978ef7154f52df289ef01cbdb51c4d
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>be690712c7a5808a0696926088ff18b2</containerId>
      <releaseId>
        <groupId>com.example.openshift</groupId>
        <artifactId>example_workflow</artifactId>
        <version>1.0</version>
      </releaseId>
      <status>STARTED</status>
      <configItems/>
      <messages/>
    </container>
    <container>
      <containerId>72978ef7154f52df289ef01cbdb51c4d</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.

6.1. 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.

6.2. ADDING, OVERRIDING, OR UPDATING MULTIPLE VERSIONS

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 *ApplicationTest* application example:

```
KIE_CONTAINER_DEPLOYMENT=ApplicationTest=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=ApplicationTest=com.example.openshift:example_wor
kflow:1.0|ApplicationTest=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.

6.3. 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: *ApplicationTest=com.example.openshift:example_workflow:1.0*
- Container ID Hash: *be690712c7a5808a0696926088ff18b2*
- Deployment Alias: *ApplicationTest*

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.

6.4. 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. Process Instance ID (specific to IPS/BPM)
2. Correlation Key (specific to IPS/BPM)
3. Task Instance ID (specific to IPS/BPM)
4. Work Item ID (specific to IPS/BPM)
5. Job Request ID (specific to IPS/BPM)
6. Conversation ID
7. 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

Client	Interaction
Intelligent Process Server interaction	org.kie.server.client.ProcessServicesClient org.kie.server.client.JobServicesClient org.kie.server.client.QueryServicesClient org.kie.server.client.UserTaskServicesClient

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.

WARNING

Due to a bug in the KIE client, client classes do not share the conversation ID for all services and therefore, users will be unable to complete the conversation. This issue is fixed in BPM Suite 6.3.1 or higher. It is recommended that you use this version of BPM Suite in your BOM file, as shown here:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom.brms</groupId>
        <artifactId>jboss-brms-bpmsuite-platform-bom</artifactId>
          <version>6.3.1.GA-redhat-2</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Default Container ID

The final stage in the process hierarchy is the 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 7. RUNNING AND CONFIGURING THE INTELLIGENT PROCESS SERVER XPAAS IMAGE

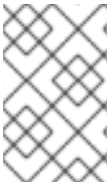
You can make changes to the Intelligent Process Server configuration in the xPaaS image using either the S2I templates, or by using a modified Intelligent Process Server image.

7.1. USING THE INTELLIGENT PROCESS SERVER XPAAS IMAGE SOURCE-TO-IMAGE (S2I) PROCESS

The recommended method to run and configure the OpenShift Intelligent Process Server xPaaS image is to use the OpenShift S2I process together with the application template parameters and environment variables.

The S2I process for the Intelligent Process Server xPaaS image 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. The Intelligent Process Server xPaaS Image 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***. . 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.

7.2. 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 library and library-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 **library** and **library-client** quickstarts.

NOTE

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

```
$ cd openshift-quickstarts/processserver/
```

```
$ mvn clean package
[INFO] Scanning for projects...
...
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ processserver-timerprocess ---
[INFO] Building jar: /tmp/openshift-
quickstarts/processserver/timerprocess/target/processserver-timerprocess-1.4.0.Final.jar
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building OpenShift Quickstarts: Intelligent Process Server: Parent 1.4.0.Final
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ processserver-parent ---
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] OpenShift Quickstarts: Intelligent Process Server: Library SUCCESS [  1.212 s]
[INFO] OpenShift Quickstarts: Intelligent Process Server: Library - Client SUCCESS [
7.827 s]
[INFO] OpenShift Quickstarts: Intelligent Process Server: Timer Process SUCCESS [
1.965 s]
[INFO] OpenShift Quickstarts: Intelligent Process Server: Parent SUCCESS [  0.002 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 11.759 s
[INFO] Finished at: 2017-06-05T10:26:06+02:00
[INFO] Final Memory: 57M/598M
[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 **library** quickstart (**processserver-library-1.4.0.Final.jar**), and WAR archive for the **library-client** quickstart

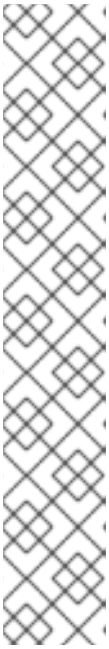
(**processserver-library-client-1.4.0.Final.war**) to the **deployments/** subdirectory:

```
$ ls
library  library-client  pom.xml  timerprocess
```

```
$ mkdir -p ps-bin-demo/deployments
```

```
$ cp library/target/processserver-library-1.4.0.Final.jar ps-bin-demo/deployments/
```

```
$ cp library-client/target/processserver-library-client-1.4.0.Final.war ps-bin-demo/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 7.1. 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
```

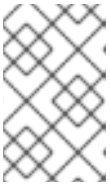
2. Create a new project.

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

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

```
$ oc get is -n openshift | grep ^jboss-process | cut -f1 -d ' '
jboss-processserver63-openshift
```

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=ps-l-app \
--image-stream=jboss-processserver63-openshift \
-e KIE_SERVER_USER=kieserveruser \
-e KIE_SERVER_PASSWORD=kieserverPwd1!
--> Found image 78c88f3 (2 months old) in image stream "openshift/jboss-processserver63-
openshift" under tag "latest" for "jboss-processserver63-openshift"

    JBoss BPMS Intelligent Process Server 6.3
    -----------------------------------------
    Platform for executing business rules on JBoss BPMS Intelligent Process Server 6.3.

    Tags: builder, processserver, processserver6

    * A source build using binary input will be created
      * The resulting image will be pushed to image stream "ps-l-app:latest"
      * A binary build was created, use 'start-build --from-dir' to trigger a new build

--> Creating resources with label build=ps-l-app ...
    imagestream "ps-l-app" created
    buildconfig "ps-l-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 ps-l-app \
--from-dir=./ps-bin-demo/ \
--follow
Uploading directory "ps-bin-demo" as binary input for the build ...
build "ps-l-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/processserver-library-client-1.4.0.Final.war' ->
'/opt/eap/standalone/deployments/processserver-library-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/processserver-library-1.4.0.Final.jar' ->
'/opt/eap/standalone/deployments/processserver-library-1.4.0.Final.jar'
/opt/eap/standalone/deployments/processserver-library-1.4.0.Final.jar is a kjar
...
INFO: org.openshift.quickstarts:processserver-library:1.4.0.Final verified.
Pushing image 172.30.82.129:5000/ps-bin-demo/ps-l-app:latest ...
Pushed 0/9 layers, 0% complete
Pushed 1/9 layers, 12% complete
Pushed 2/9 layers, 33% complete
Pushed 3/9 layers, 45% complete
Pushed 4/9 layers, 64% complete
Pushed 5/9 layers, 71% complete
Pushed 6/9 layers, 80% complete
Pushed 7/9 layers, 88% complete
Pushed 8/9 layers, 100% complete
Pushed 9/9 layers, 100% complete
Push successful
```

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

```
$ oc new-app ps-l-app
--> Found image 65a8367 (About a minute old) in image stream "ps-bin-demo/ps-l-app"
under tag "latest" for "ps-l-app"

    ps-bin-demo/ps-l-app-1:0af8685b
    -------------------------------
    Platform for executing business rules on JBoss BPMS Intelligent Process Server 6.3.

    Tags: builder, processserver, processserver6

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

--> Creating resources ...
    deploymentconfig "ps-l-app" created
    service "ps-l-app" created
--> Success
    Run 'oc status' to view your app.
```

7. Expose the service as route.

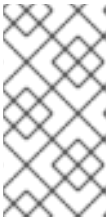
```
$ oc get svc -o name
service/ps-l-app
```

```
$ oc expose svc/ps-l-app
route "ps-l-app" exposed
```

```
$ oc get route
NAME      HOST/PORT                                  PATH      SERVICES   PORT
TERMINATION   WILDCARD
ps-l-app   ps-l-app-ps-bin-demo.openshift.example.com            ps-l-app   8080-tcp
None
```

8. Access the application.
   You can get the list of available query string arguments of the **library** application by accessing the URL **http://ps-l-app-ps-bin-demo.openshift.example.com/library**.

   Run the **library-client** servlet using the URL **http://ps-l-app-ps-bin-demo.openshift.example.com/library?command=runLocal**.



NOTE

You may verify the current KIE server state by accessing dedicated **server/** page of the REST API: **http://ps-l-app-ps-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).

PART IV. TUTORIALS

CHAPTER 8. EXAMPLE WORKFLOW: DEPLOYING A JBOSS BPMS PROJECT AS INTELLIGENT PROCESS SERVER FOR OPENSHIFT IMAGE

Business Central, the JBoss BPMS console that provides a unified web-based environment for managing projects, is not part of Intelligent Process Server for OpenShift. An external repository is required to integrate projects between Business Central and OpenShift. This tutorial presumes that a project has already been set up in Business Central.

8.1. PREPARING THE JBOSS BPMS PROJECT

Prepare a Git repository for the project configuration files. The repository is used to store the Maven repository and KIE container deployment files. The Git repository has the following file hierarchy:

- **../configuration/settings.xml** for the Maven repository
- **../.s2i/environment** for the KIE container deployment

Preparation of the BPMS project may require some prior knowledge with the Red Hat JBoss BPM Suite. Refer to the Red Hat JBoss BPM Suite User Guide for more information on any of these tasks.

1. Log in to Red Hat JBoss BPM Suite: Business Central web console.
2. Clone the existing repository to ensure it is up-to-date.
   a. **Authoring → Administration**
   b. **Repositories → Clone Repository**
   c. Provide the **Repository Name**, **Organizational Unit**, and the **Git URL** and click the **Clone** button.
3. 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/content/repo/snapshots/</url>
  </snapshotRepository>
</distributionManagement>
...
```

   For more information, see the Red Hat JBoss BRMS Administration and Configuration Guide .
4. In the application repository, ensure the **configuration/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 **configuration/settings.xml** so that OpenShift can download the application artifacts. 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 *ipsDemo* in this example:

```
KIE_CONTAINER_DEPLOYMENT=ipsDemo=com.example.openshift:example_workflow:1
.0
```

5. Click **Save** if any changes have been made to the project.

6. Click **Open Project Editor → Build → Build and Deploy**. This will push the project artifacts into the Maven repository so that it is ready to be deployed on OpenShift.

8.2. PREPARING INTELLIGENT PROCESS SERVER DEPLOYMENT

1. Create a new project:

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

2. Create a service account to be used for the IPS deployment:

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

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

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

4. The Intelligent Process 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 ips-app-secret --from-file=jgroups.jceks --from-file=keystore.jks
   ```

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

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

8.3. DEPLOYMENT

1. Log in to the OpenShift web console and select the *ips-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 **processserver**. You may need to click **See all** to show the desired application template.
4. Select and configure the desired template.
   The **SOURCE_REPOSITORY_URL** must be set to the Git repository for the deployment, so that the application can pull the **configuration/settings.xml** and **.s2i/environment** files.
5. 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.

CHAPTER 9. EXAMPLE WORKFLOW: DEPLOYING AN UPDATED VERSION CONCURRENTLY WITH ORIGINAL APPLICATION

This example workflow follows on from Example Workflow: Deploying a JBoss BPMS Project as an xPaaS Intelligent Process Server xPaaS Image, in which the *1.0* version of the *example_workflow* artifact was deployed with a deployment alias of *ipsDemo*. 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 *ipsDemo* 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=ipsDemo=com.example.openshift:example_workflow:1.0|ipsDemo=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 ips-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.

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/ips
```

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

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

3. Verify the setting

```
oc env bc/ips --list
# buildconfigs ips
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.
KIE_SERVER_PROTOCOL	The protocol to access the KIE Server REST interface.
KIE_SERVER_PORT	The port to access the KIE Server REST interface.
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).
KIE_SERVER_DOMAIN	JAAS LoginContext domain that shall be used to authenticate users when using JMS.
KIE_SERVER_JMS_QUEUES_REQUEST	JNDI name of request queue for JMS.
KIE_SERVER_JMS_QUEUES_RESPONSE	JNDI name of response queue for JMS.
KIE_SERVER_EXECUTOR_JMS_QUEUE	JNDI name of executor queue for JMS.
KIE_SERVER_PERSISTENCE_DIALECT	Hibernate persistence dialect.
HOSTNAME_HTTP	Custom hostname for http service route. Leave blank for default hostname, e.g., *<application-name>-<project>.<default-domain-suffix>*
HOSTNAME_HTTPS	Custom hostname for https service route. Leave blank for default hostname, e.g., secure-*<application-name>-<project>.<default-domain-suffix>*
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.
DB_JNDI	Database JNDI name used by application to resolve the datasource, e.g. java:/jboss/datasources/ExampleDS

Variable	Description
DB_DATABASE	Database name.
VOLUME_CAPACITY	Size of persistent storage for database volume.
MQ_JNDI	JNDI name for connection factory used by applications to connect to the broker, e.g. java:/JmsXA
MQ_PROTOCOL	Broker protocols to configure, separated by commas. Allowed values are: **openwire**, **amqp**, **stomp** and **mqtt**. Only **openwire** is supported by EAP.
MQ_QUEUES	Queue names, separated by commas. These queues will be automatically created when the broker starts. Also, they will be made accessible as JNDI resources in EAP.
MQ_TOPICS	Topic names, separated by commas. These topics will be automatically created when the broker starts. Also, they will be made accessible as JNDI resources in EAP.
HTTPS_SECRET	The name of the secret containing the keystore file.
HTTPS_KEYSTORE	The name of the keystore file within the secret.
HTTPS_NAME	The name associated with the server certificate.
HTTPS_PASSWORD	The password for the keystore and certificate.
DB_USERNAME	Database user name.
DB_PASSWORD	Database user password
DB_MIN_POOL_SIZE	Sets xa-pool/min-pool-size for the configured datasource.
DB_MAX_POOL_SIZE	Sets xa-pool/max-pool-size for the configured datasource.
DB_TX_ISOLATION	Sets transaction-isolation for the configured datasource.
POSTGRESQL_MAX_CONNECTIONS	The maximum number of client connections allowed. This also sets the maximum number of prepared transactions.

Variable	Description
POSTGRESQL_SHARED_BUFFERS	Configures how much memory is dedicated to PostgreSQL for caching data.
MQ_USERNAME	User name for standard broker user. It is required for connecting to the broker. If left empty, it will be generated.
MQ_PASSWORD	Password for standard broker user. It is required for connecting to the broker. If left empty, it will be generated.
AMQ_ADMIN_USERNAME	User name for broker admin. If left empty, it will be generated.
AMQ_ADMIN_PASSWORD	Password for broker admin. If left empty, it will be generated.
GITHUB_WEBHOOK_SECRET	GitHub trigger secret.
GENERIC_WEBHOOK_SECRET	Generic build trigger secret.
IMAGE_STREAM_NAMESPACE	Namespace in which the ImageStreams for Red Hat Middleware images are installed. These ImageStreams are normally installed in the openshift namespace. You should only need to modify this if you've installed the ImageStreams in a different namespace/project.

CHAPTER 12. LOGGING

In addition to viewing the OpenShift logs, you can troubleshoot a running Intelligent Process Server for OpenShift by viewing the logs that are outputted to the container's console:

```
$ oc logs -f <pod-name>
```

CHAPTER 13. ENDPOINTS

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

13.1. REST

Clients can use the REST API in various ways:

13.1.1. 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/processserver-library

13.1.2. Java

```
// LibraryClient.java
KieServicesConfiguration config = KieServicesFactory.newRestConfiguration(
  "http://host/kie-server/services/rest/server", "kieserverUser", "kieserverPassword");
config.setMarshallingFormat(MarshallingFormat.XSTREAM);
ProcessServicesClient client =
  KieServicesFactory.newKieServicesClient(config).getServicesClient(ProcessServicesClient.class);
Map<String,Object> params = new HashMap<String,Object>();
LoanRequest loanRequest = new LoanRequest();
loanRequest.setIsbn("978-0-307-35193-7");
params.put("loanRequest", loanRequest);
Long processInstanceId = client.startProcess("processserver-library", "LibraryProcess", params);
```

13.2. JMS

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

13.2.1. Java (HornetQ)

```
// LibraryClient.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);
ProcessServicesClient client =
  KieServicesFactory.newKieServicesClient(config).getServicesClient(ProcessServicesClient.class);
Map<String,Object> params = new HashMap<String,Object>();
LoanRequest loanRequest = new LoanRequest();
```

```
loanRequest.setIsbn("978-0-307-35193-7");
params.put("loanRequest", loanRequest);
Long processInstanceId = client.startProcess("processserver-library", "LibraryProcess", params);
```

13.2.2. Java (ActiveMQ)

```
// LibraryClient.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);
ProcessServicesClient client =
  KieServicesFactory.newKieServicesClient(config).getServicesClient(ProcessServicesClient.class);
Map<String,Object> params = new HashMap<String,Object>();
LoanRequest loanRequest = new LoanRequest();
loanRequest.setIsbn("978-0-307-35193-7");
params.put("loanRequest", loanRequest);
Long processInstanceId = client.startProcess("processserver-library", "LibraryProcess", params);
```

CHAPTER 14. TROUBLESHOOTING

In addition to viewing the OpenShift logs, you can troubleshoot a running Intelligent Process Server for OpenShift 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, Intelligent Process Server for OpenShift 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.