OpenJDK 11

Using OpenJDK 11 source-to-image for OpenShift
Abstract

OpenJDK 11 is a Red Hat offering on the Red Hat Enterprise Linux platform. The Using OpenJDK 11 source-to-image for OpenShift guide provides an overview of Red Hat Java S2I for OpenShift, and explains how to use OpenJDK 11 S2I for OpenShift.
# Table of Contents

**MAKING OPEN SOURCE MORE INCLUSIVE** ................................................. 3

**PROVIDING FEEDBACK ON RED HAT DOCUMENTATION** ............................ 4

**CHAPTER 1. INTRODUCTION TO RED HAT JAVA SOURCE-TO-IMAGE FOR OPENSHIFT** ............................................. 5

**CHAPTER 2. BEFORE YOU BEGIN** .......................................................... 6

  Initial setup ................................................................. 6

  Version compatibility and support ........................................ 6

**CHAPTER 3. USING JAVA SOURCE-TO-IMAGE FOR OPENSHIFT** ..................... 7

  3.1. BUILDING AND DEPLOYING JAVA APPLICATIONS WITH SOURCE-TO-IMAGE FOR OPENSHIFT ........................................... 7

  3.2. BUILDING AND DEPLOYING JAVA APPLICATIONS FROM BINARY ARTIFACTS ......................................................... 8

**CHAPTER 4. EXAMPLE WORKFLOWS FOR JAVA S2I ON OPENSHIFT** ............... 11

  4.1. REMOTE DEBUGGING JAVA APPLICATION FOR OPENSHIFT IMAGE .......................... 11

    Prepare for deployment .............................................. 11

    Deployment .................................................................. 11

    Enabling remote debugging for a new application ............ 11

    Enabling remote debugging for an existing application .... 11

    Post-deployment ......................................................... 12

    Connect local debugging port to a port on the pod ........ 12

    Attach debugger to an application .............................. 13

  4.2. RUNNING FLAT CLASSPATH JAR ON JAVA SOURCE-TO-IMAGE FOR OPENSHIFT 14

    Prepare for Deployment ............................................... 14

    Deployment ................................................................... 14

    Post-deployment ......................................................... 14

**CHAPTER 5. REFERENCE** ........................................................................ 15

  5.1. VERSION DETAILS ............................................................. 15

  5.2. INFORMATION ENVIRONMENT VARIABLES .............................................. 15

  5.3. CONFIGURATION ENVIRONMENT VARIABLES .................................... 15

    5.3.1. Configuration environment variables with default values ........ 27

  5.4. EXPOSED PORTS ............................................................... 28

  5.5. MAVEN SETTINGS ............................................................... 28

    Default Maven settings with Maven arguments .................. 28

    Provide custom Maven settings .................................... 29
MAKING OPEN SOURCE MORE INCLUSIVE

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

We appreciate your feedback on our documentation. To provide feedback, you can highlight the text in a document and add comments.

This section explains how to submit feedback.

Prerequisites

- You are logged in to the Red Hat Customer Portal.
- In the Red Hat Customer Portal, view the document in Multi-page HTML format.

Procedure

To provide your feedback, perform the following steps:

1. Click the Feedback button in the top-right corner of the document to see existing feedback.

   **NOTE**
   
   The feedback feature is enabled only in the Multi-page HTML format.

2. Highlight the section of the document where you want to provide feedback.

3. Click the Add Feedback pop-up that appears near the highlighted text. A text box appears in the feedback section on the right side of the page.

4. Enter your feedback in the text box and click Submit. A documentation issue is created.

5. To view the issue, click the issue tracker link in the feedback view.
OpenShift Container Platform provides an source-to-image (S2I) process to build and run applications. You can attach the application’s source code on top of a builder image (a technology image such as JBoss EAP). S2I process builds your application and layers it on top of the builder image to create an application image. After your application image is built, you can push it to an integrated registry inside OpenShift or to a standalone registry.

With Red Hat Java S2I for OpenShift you can build and run basic Java applications, for example, jat-jar or flat classpath within a containerized image on OpenShift.
CHAPTER 2. BEFORE YOU BEGIN

Initial setup
Create an OpenShift instance. For more details on how to create an OpenShift instance, see OpenShift container platform installation overview.

Version compatibility and support
OpenShift Container Platform versions 3.11 and 4.7 support the Red Hat Java S2I for OpenShift image.

For details about the current support levels for OpenShift Container Platform, see Red Hat OpenShift Container Platform Life Cycle Policy and Red Hat OpenShift Container Platform Life Cycle Policy (non-current versions).
CHAPTER 3. USING JAVA SOURCE-TO-IMAGE FOR OPENSHIFT

You can use the Java source-to-image (S2I) for OpenShift image to run your custom Java applications on OpenShift.

3.1. BUILDING AND DEPLOYING JAVA APPLICATIONS WITH SOURCE-TO-IMAGE FOR OPENSHIFT

To build and deploy a Java application from source on OpenShift using the Java source-to-image (S2I) for OpenShift image, use the OpenShift S2I process.

Procedure

1. Log in to the OpenShift instance by running the following command and by providing your credentials:
   
   ```bash
   $ oc login
   ```

2. Create a new project:
   
   ```bash
   $ oc new-project <project-name>
   ```

3. Create a new application using the Java S2I for OpenShift image:
   The `<source-location>` is the URL of git repository or path to a local folder.

   ```bash
   $ oc new-app java:11~<source-location>
   ```

4. Get the service name:
   
   ```bash
   $ oc get service
   ```

5. Expose the service as a route, so that you can use the server from your browser. In the following example, replace `<service-name>` with the value of the NAME field from previous command’s output:

   ```bash
   $ oc expose svc/<service-name> --port=8080
   ```

6. Get the route:
   
   ```bash
   $ oc get route
   ```

7. Access the application in your browser by using the URL. Use the value of HOST/PORT field from the previous command’s output.

Additional resources

- For more detailed example, see the Running flat classpath JAR on Java source-to-image for OpenShift.
3.2. BUILDING AND DEPLOYING JAVA APPLICATIONS FROM BINARY ARTIFACTS

You can deploy your existing Java applications on OpenShift by using the binary source capability.

The procedure uses undertow-servlet quickstart to build a Java application on your local machine. The quickstart copies the resulting binary Artifacts into OpenShift by using the S2I binary source capability.

Prerequisites

- Enable Red Hat JBoss Enterprise Maven Repository on your local machine.
- Get the JAR application archive and build the application locally.
  - Clone the undertow-servlet source code:
    
    ```sh
    $ git clone https://github.com/jboss-openshift/openshift-quickstarts.git
    ```
  - Build the application:
    
    ```sh
    $ cd openshift-quickstarts/undertow-servlet/
    $ mvn clean package
    [INFO] Scanning for projects...
    ...
    [INFO] Building Undertow Servlet Example 1.0.0.Final
    ...
    [INFO] BUILD SUCCESS
    [INFO] Total time: 1.986 s
    [INFO] Finished at: 2017-06-27T16:43:07+02:00
    [INFO] Final Memory: 19M/281M
    ```
  - Prepare the directory structure on the local file system.
    Copy the application archives in the deployments/ sub-directory (where the main binary build directory) to the standard deployments folder (where the image is build on OpenShift).
    Structure the directory hierarchy containing the web application data for the application to deploy.
    
    Create main directory for the binary build on the local file system and deployments/ subdirectory within it. Copy the built JAR archive to the deployments/ subdirectory:
    
    ```sh
    undertow-servlet]$ ls
    dependency-reduced-pom.xml pom.xml README src target
    $ mkdir -p ocp/deployments
    $ cp target/undertow-servlet.jar ocp/deployments/
    ```
Procedure

1. Log in to the OpenShift instance by running the following command and by providing your credentials:

   $ oc login

2. Create a new project:

   $ oc new-project jdk-bin-demo

3. Create new binary build, specify image stream, and application name:

   $ oc new-build --binary=true --name=jdk-us-app --image-stream=java:11
   --> Found image c1f5b31 (2 months old) in image stream "openshift/java:11" under tag "latest" for "java:11"

   Java Applications
   -----------------
   Platform for building and running plain Java applications (fat-jar and flat classpath)

   --> Creating resources with label build=jdk-us-app ...
   imagestream "jdk-us-app" created
   buildconfig "jdk-us-app" created
   --> Success
   Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
   'oc expose svc/jdk-us-app'

4. Start the binary build.
   Instruct oc executable to use main directory of the binary build you have created in previous step as the directory containing binary input for the OpenShift build:

   $ oc start-build jdk-us-app --from-dir=./ocp --follow
   Uploading directory "ocp" as binary input for the build ...
   build "jdk-us-app-1" started
   Receiving source from STDIN as archive ...
   ==============================================================
   Starting S2I Java Build ..... 
   S2I source build with plain binaries detected
   Copying binaries from /tmp/src/deployments to /deployments ...
   ... done
   Pushing image 172.30.197.203:5000/jdk-bin-demo/jdk-us-app:latest ...
   Pushed 0/6 layers, 2% complete
   Pushed 1/6 layers, 24% complete
   Pushed 2/6 layers, 36% complete
   Pushed 3/6 layers, 54% complete
   Pushed 4/6 layers, 71% complete
   Pushed 5/6 layers, 95% complete
   Pushed 6/6 layers, 100% complete
   Push successful
5. Create a new OpenShift application based on the build:

```
$ oc new-app jdk-us-app
---> Found image 66f4e0b (About a minute old) in image stream "jdk-bin-demo/jdk-us-app" under tag "latest" for "jdk-us-app"

jdk-bin-demo/jdk-us-app-1:c1dbfb7a
------------------
Platform for building and running plain Java applications (fat-jar and flat classpath)

Tags: builder, java

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

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

6. Expose the service as route.

```
$ oc expose svc/jdk-us-app
route "jdk-us-app" exposed
```

7. Get the route:

```
$ oc get route
```

8. Access the application in your browser by using the URL (value of HOST/PORT field from the previous command output).

**Additional resources**

- Use the **binary source** capability to deploy existing Java applications on OpenShift.
- For more information on how to configure maven repository, see **Use the Maven Repository**.
4.1. REMOTE DEBUGGING JAVA APPLICATION FOR OPENSHIFT IMAGE

The example in the procedure shows the remote debugging of a Java application deployed on OpenShift by using the Java S2I for OpenShift image. You can enable the capability by setting the value of the environment variables `JAVA_DEBUG` to `true` and `JAVA_DEBUG_PORT` to `9009`, respectively.

**NOTE**

If the `JAVA_DEBUG` variable is set to `true` and no value is provided for the `JAVA_DEBUG_PORT` variable, `JAVA_DEBUG_PORT` is set to `5005` by default.

Prepare for deployment

**Procedure**

1. Log in to the OpenShift instance by running following command and by providing your credentials:

   ```
   $ oc login
   ```

2. Create a new project:

   ```
   $ oc new-project js2i-remote-debug-demo
   ```

Deployment

You can enable remote debugging for your new and existing applications.

Enabling remote debugging for a new application

**Procedure**

- Create a new application by using the Java S2I for OpenShift image and example Java source code. Ensure that you set the `JAVA_DEBUG` and the `JAVA_DEBUG_PORT` environment variables before creating your application:

  ```
  $ oc new-app java:11~https://github.com/jboss-openshift/openshift-quickstarts.git \
  --context-dir=undertow-servlet \
  -e JAVA_DEBUG=true \
  -e JAVA_DEBUG_PORT=9009
  ```

  Proceed to Connect local debugging port to a port on the pod.

Enabling remote debugging for an existing application

**Procedure**

1. Switch to the appropriate OpenShift project:
$ oc project js2i-remote-debug-demo

2. Retrieve the name of the deploymentconfig:

$ oc get dc -o name
deploymentconfig/openshift-quickstarts

3. Edit the deploymentconfig and add the JAVA_DEBUG=true and JAVA_DEBUG_PORT=9009 environment variables.

4. Specify object to edit at the path .spec.template.spec.containers and type of Container:

$ oc edit dc/openshift-quickstarts

NOTE
Launch an editor to run oc edit command in your terminal. You can change the editor that is launched by defining your environment’s EDITOR variable.

Proceed to Connect local debugging port to a port on the pod.

Post-deployment
Connect local debugging port to a port on the pod

Procedure

1. Get the name of the pod running the application (Status Running):
   Example showing openshift-quickstarts-1-1uyym as the pod name.

   $ oc get pods
   NAME               READY STATUS    RESTARTS AGE
   openshift-quickstarts-1-1uyym   1/1   Running   0      3m
   openshift-quickstarts-1-build   0/1   Completed 0      6m

2. Use the OpenShift or Kubernetes port forwarding feature to listen on a local port and forward to a port on the OpenShift pod. <running-pod> is the value of the NAME field for the pod with Status “running” from the previous command output:

   $ oc port-forward <running-pod> 5005:9009
   Forwarding from 127.0.0.1:5005 -> 9009
   Forwarding from ::1:5005 -> 9009

NOTE
In the previous example, 5005 is the port number on the local system, while 9009 is the remote port number of the OpenShift pod running the Java S2I for OpenShift image. Therefore, future debugging connections made to local port 5005 are forwarded to port 9009 of the OpenShift pod, running the Java Virtual Machine (JVM).
IMPORTANT

The command might prevent you from typing further in the terminal. In this case, launch a new terminal for performing the next steps.

Attach debugger to an application

Procedure

1. Attach the debugger on the local system to the remote JVM running on the Java S2I for OpenShift image:

   $ jdb -attach 5005
   Set uncaught java.lang.Throwable
   Set deferred uncaught java.lang.Throwable
   Initializing jdb ...
   >
   ...

   **NOTE**
   Once the local debugger to the remote OpenShift pod debugging connection is initiated, an entry similar to handling connection for 5005 is shown in the console where the previous oc port-forward command was issued.

2. Debug the application:

   $ jdb -attach 5005
   Set uncaught java.lang.Throwable
   Set deferred uncaught java.lang.Throwable
   Initializing jdb ...
   > threads
   Group system:
   (java.lang.ref.Reference$ReferenceHandler)0x79e Reference Handler  cond. waiting
   (java.lang.ref.Finalizer$FinalizerThread)0x79f Finalizer  cond. waiting
   (java.lang.Thread)0x7a0 Signal Dispatcher running
   Group main:
   (java.util.TimerThread)0x7a2 server-timer cond. waiting
   (org.jolokia.jvmagent.CleanupThread)0x7a3 Jolokia Agent Cleanup Thread cond. waiting
   (org.xnio.nio.WorkerThread)0x7a4 XNIO-1 I/O-1 running
   (org.xnio.nio.WorkerThread)0x7a5 XNIO-1 I/O-2 running
   (org.xnio.nio.WorkerThread)0x7a6 XNIO-1 I/O-3 running
   (org.xnio.nio.WorkerThread)0x7a7 XNIO-1 Accept running
   (java.lang.Thread)0x7a8 DestroyJavaVM running
   Group jolokia:
   (java.lang.Thread)0x7aa Thread-3 running
   >

Additional resources

- For more information on Openshift common object reference, see the OpenShift Common Object Reference, section Container.
- For more information on connecting the IDE debugger of the Red Hat JBoss Developer Studio to the OpenShift pod running the Java S2I for OpenShift image, see Configuring and Connecting the IDE Debugger.

4.2. RUNNING FLAT CLASSPATH JAR ON JAVA SOURCE-TO-IMAGE FOR OPENSHIFT

The example in the procedure describes the process of running flat classpath java applications on Java S2I for OpenShift.

Prepare for Deployment

Procedure

1. Log in to the OpenShift instance by providing your credentials:

   $ oc login

2. Create a new project:

   $ oc new-project js2i-flatclasspath-demo

Deployment

Procedure

1. Create a new application using the Java S2I for OpenShift image and Java source code:

   $ oc new-app java:11~https://github.com/jboss-openshift/openshift-quickstarts.git --context-dir=undertow-servlet

Post-deployment

Procedure

1. Get the service name:

   $ oc get service

2. Expose the service as a route to be able to use it from the browser:

   $ oc expose svc/openshift-quickstarts --port=8080

3. Get the route:

   $ oc get route

4. Access the application in your browser by using the URL (value of HOST/PORT field from previous command output).
CHAPTER 5. REFERENCE

5.1. VERSION DETAILS

The following table lists versions of technologies used in this image.

Table 5.1. Technology versions used in this image

<table>
<thead>
<tr>
<th>Technology</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJDK</td>
<td>11</td>
</tr>
<tr>
<td>Jolokia</td>
<td>1.6.2</td>
</tr>
<tr>
<td>Maven</td>
<td>3.6</td>
</tr>
</tbody>
</table>

5.2. INFORMATION ENVIRONMENT VARIABLES

The following information environment variables are designed to convey information about the image. Do not modify these variables.

Table 5.2. Information environment variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>/home/jboss</td>
</tr>
<tr>
<td>JAVA_HOME</td>
<td>/usr/lib/jvm/java-11</td>
</tr>
<tr>
<td>JAVA_VENDOR</td>
<td>openjdk</td>
</tr>
<tr>
<td>JAVA_VERSION</td>
<td>11</td>
</tr>
<tr>
<td>JOLOKIA_VERSION</td>
<td>1.6.2</td>
</tr>
<tr>
<td>LD_PRELOAD</td>
<td>libnss_wrapper.so</td>
</tr>
<tr>
<td>MAVEN_VERSION</td>
<td>3.6</td>
</tr>
<tr>
<td>NSS_WRAPPER_GROUP</td>
<td>/etc/group</td>
</tr>
<tr>
<td>NSS_WRAPPER_PASSWD</td>
<td>/home/jboss/passwd</td>
</tr>
</tbody>
</table>

5.3. CONFIGURATION ENVIRONMENT VARIABLES

Configuration environment variables are designed to conveniently adjust the image without requiring a rebuild, and should be set by the user as desired.
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Default value</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB_JOLOKIA_CONFIG</td>
<td>If set uses this file (including path) as Jolokia JVM agent properties (as described in the Jolokia reference manual). If not set, the /opt/jolokia/etc/jolokia.properties will be created using the settings as defined in the manual. Otherwise the rest of the settings in this document are ignored.</td>
<td>-</td>
<td>/opt/jolokia/custom.properties</td>
</tr>
<tr>
<td>AB_JOLOKIA_DISCOVERY_ENABLED</td>
<td>Enable Jolokia discovery.</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>AB_JOLOKIA_HOST</td>
<td>Host address to bind to.</td>
<td>0.0.0.0</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>AB_JOLOKIA_ID</td>
<td>Agent ID to use, which is the container id.</td>
<td>$HOSTNAME</td>
<td>openjdk-app-1-xqlsj</td>
</tr>
<tr>
<td>AB_JOLOKIA_OFF</td>
<td>If set disables activation of Jolokia (that is, echos an empty value).</td>
<td>Jolokia is enabled</td>
<td>true</td>
</tr>
<tr>
<td>AB_JOLOKIA_OPTS</td>
<td>Additional options to be appended to the agent configuration. They should be specified in the format key=value,key=value, ....</td>
<td>-</td>
<td>backlog=20</td>
</tr>
<tr>
<td>AB_JOLOKIA_PASSWORD</td>
<td>Password for basic authentication. By default authentication is switched off.</td>
<td>-</td>
<td>mypassword</td>
</tr>
<tr>
<td>AB_JOLOKIA_PORT</td>
<td>Port to listen to.</td>
<td>8778</td>
<td>5432</td>
</tr>
<tr>
<td>AB_JOLOKIA_USER</td>
<td>User for basic authentication.</td>
<td>jolokia</td>
<td>myusername</td>
</tr>
<tr>
<td>AB_PROMETHEUS_ENABLE</td>
<td>Enable the use of the Prometheus agent.</td>
<td>-</td>
<td>True</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>AB_PROMETHEUS_JMX_EXPORTER_PORT</td>
<td>Port to use for the Prometheus JMX Exporter.</td>
<td>-</td>
<td>9799</td>
</tr>
<tr>
<td>CONTAINER_CORE_LIMIT</td>
<td>A calculated core limit as described in the CFS Bandwidth Control.</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>CONTAINER_MAX_MEMORY</td>
<td>Memory limit assigned to the container.</td>
<td>-</td>
<td>1024</td>
</tr>
<tr>
<td>GC_ADAPTIVE_SIZE_POLICY_WEIGHT</td>
<td>The weighting given to the current garbage collector time versus previous garbage collector times.</td>
<td>-</td>
<td>90</td>
</tr>
<tr>
<td>GC_CONTAINER_OPTIONS</td>
<td>Specify Java GC to use. The value of this variable should contain the necessary JRE command-line options to specify the required GC, which will override the default value.</td>
<td>-XX:+UseParallelOldGC</td>
<td>-XX:+UseG1GC</td>
</tr>
<tr>
<td>GC_MAX_HEAP_FREE_RATIO</td>
<td>Maximum percentage of heap free after GC to avoid shrinkage.</td>
<td>-</td>
<td>40</td>
</tr>
<tr>
<td>GC_MAX_METASPACE_SIZE</td>
<td>The maximum metaspace size.</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>GC_METASPACE_SIZE</td>
<td>The initial metaspace size.</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>GC_MIN_HEAP_FREE_RATIO</td>
<td>Minimum percentage of heap free after GC to avoid expansion.</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>GC_TIME_RATIO</td>
<td>Specifies the ratio of the time spent outside the garbage collection (for example, the time spent for application execution) to the time spent in the garbage collection.</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>HTTPS_PROXY</td>
<td>The location of the HTTPS proxy. This takes precedence over http_proxy and HTTP_PROXY, and will be used for both Maven builds and Java runtime.</td>
<td>-</td>
<td>myuser@127.0.0.1:8080</td>
</tr>
<tr>
<td>HTTP_PROXY</td>
<td>The location of the HTTP proxy. This will be used for both Maven builds and Java runtime.</td>
<td>-</td>
<td>127.0.0.1:8080</td>
</tr>
<tr>
<td>JAVA_APP_DIR</td>
<td>The directory where the application resides. All paths in your application are relative to this directory.</td>
<td>-</td>
<td>myapplication/</td>
</tr>
<tr>
<td>JAVA_ARGS</td>
<td>Arguments passed to the java application.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>JAVA_CLASSPATH</td>
<td>The classpath to use. If not given, the startup script checks for a file JAVA_APP_DIR/classpath and uses its content literally as classpath. If this file does not exists all jars in the app dir are added (classes:JAVA_APP_DIR/).</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>JAVA_DEBUG</td>
<td>If set remote debugging will be switched on.</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>JAVA_DEBUG_PORT</td>
<td>Port used for remote debugging.</td>
<td>5005</td>
<td>8787</td>
</tr>
<tr>
<td>JAVA_DIAGNOSTICS</td>
<td>Set this to print some diagnostics information to standard output during the command is running.</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>JAVA_INITIAL_MEM_RATIO</td>
<td>It is used when no <code>-Xms</code> option is given in <code>JAVA_OPTS</code>. This is used to calculate a default initial heap memory based on the maximum heap memory. If used in a container without any memory constraints for the container then this option has no effect. If there is a memory constraint then <code>-Xms</code> is set to a ratio of the <code>-Xmx</code> memory as set here. The default is 25 which means 25% of the <code>-Xmx</code> is used as the initial heap size. You can skip this mechanism by setting this value to 0 in which case no <code>-Xms</code> option is added.</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>JAVA_LIB_DIR</td>
<td>Directory holding the Java jar files as well as an optional classpath file which holds the classpath. Either as a single-line classpath (colon separated) or with jar files listed line by line. If not set <code>JAVA_LIB_DIR</code> is set to the value of <code>JAVA_APP_DIR</code>.</td>
<td><code>JAVA_APP_DIR</code></td>
<td>-</td>
</tr>
<tr>
<td>JAVA_MAIN_CLASS</td>
<td>A main class to use as argument for <code>java</code>. When this environment variable is given, all jar files in <code>JAVA_APP_DIR</code> are added to the classpath as well as <code>JAVA_LIB_DIR</code>.</td>
<td>-</td>
<td>com.example.MainClass</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>JAVA_MAX_INITIAL_MEMORY</td>
<td>It is used when no <code>-Xms</code> option is given in <code>JAVA_OPTS</code>. This is used to calculate the maximum value of the initial heap memory. If used in a container without any memory constraints for the container then this option has no effect. If there is a memory constraint then <code>-Xms</code> is limited to the value set here. The default is 4096 which means the calculated value of <code>-Xms</code> will never be greater than 4096. The value of this variable is expressed in MB.</td>
<td>4096</td>
<td>4096</td>
</tr>
<tr>
<td>JAVA_MAX_MEM_RATIO</td>
<td>It is used when no <code>-Xmx</code> option is given in <code>JAVA_OPTS</code>. This is used to calculate a default maximum heap memory based on a container’s restriction. If used in a container without any memory constraints for the container then this option has no effect. If there is a memory constraint then <code>-Xmx</code> is set to a ratio of the container available memory as set here. The default is 50 which means 50% of the available memory is used as an upper boundary. You can skip this mechanism by setting this value to 0 in which case no <code>-Xmx</code> option is added.</td>
<td>50</td>
<td>-</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>JAVA_OPTS</td>
<td>JVM options passed to the <code>java</code> command.</td>
<td>-</td>
<td><code>-verbose:class</code></td>
</tr>
<tr>
<td>JAVA_OPTS_APPEND</td>
<td>User-specified Java options to be appended to generated options in JAVA_OPTS</td>
<td>-</td>
<td><code>-Dsomed.property=foo</code></td>
</tr>
<tr>
<td>LOGGING_SCRIPT_DEBUG</td>
<td>Set to true to enable script debugging. Deprecates <code>SCRIPT_DEBUG</code>.</td>
<td>true</td>
<td>True</td>
</tr>
<tr>
<td>MAVEN_ARGS</td>
<td>Arguments to use when calling Maven, replacing the default <code>package hawt-app:build -DskipTests -e</code>. Ensure that you run the <code>hawt-app:build</code> goal (when not already bound to the <code>package</code> execution phase), otherwise the startup scripts will not work.</td>
<td>package hawt-app:build -DskipTests -e</td>
<td><code>-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repos.redhatga package</code></td>
</tr>
<tr>
<td>MAVEN_ARGS_APPEND</td>
<td>Additional Maven arguments.</td>
<td>-</td>
<td><code>-X -am -pl</code></td>
</tr>
<tr>
<td>MAVEN_CLEAR_REPO</td>
<td>If set then the Maven repository is removed after the artifact is built. This is useful for for reducing the size of the created application image small, but prevents incremental builds. Will be overridden by <code>S2I_ENABLE_INCREMENTAL_BUILDS</code>.</td>
<td>false</td>
<td>-</td>
</tr>
<tr>
<td>MAVEN_LOCAL_REPO</td>
<td>Directory to use as the local Maven repository.</td>
<td>-</td>
<td><code>/home/jboss/.m2/repository</code></td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>MAVEN_MIRRORS</td>
<td>If set, multi-mirror support is enabled, and other MAVEN_MIRROR_* variables will be prefixed. For example, DEV_ONE_MAVEN_MIRROR_URL and QE_TWO_MAVEN_MIRROR_URL.</td>
<td>-</td>
<td>dev-one,qe-two</td>
</tr>
<tr>
<td>MAVEN_MIRROR_URL</td>
<td>The base URL of a mirror used for retrieving artifacts.</td>
<td>-</td>
<td><a href="http://10.0.0.1:8080/repository/internal/">http://10.0.0.1:8080/repository/internal/</a></td>
</tr>
<tr>
<td>MAVEN_REPOS</td>
<td>If set, multi-repo support is enabled, and other MAVEN_REPO_* variables will be prefixed. For example, DEV_ONE_MAVEN_REPO_URL and QE_TWO_MAVEN_REPO_URL.</td>
<td>-</td>
<td>dev-one,qe-two</td>
</tr>
<tr>
<td>MAVEN_S2I_ARTIFACT_DIRS</td>
<td>Relative paths of source directories to scan for build output, which will be copied to $DEPLOY_DIR.</td>
<td>target</td>
<td>target</td>
</tr>
<tr>
<td>MAVEN_S2I_GOALS</td>
<td>Space-separated list of goals to be executed with Maven build. For example, mvn $MAVEN_S2I_GOALS.</td>
<td>package</td>
<td>package install</td>
</tr>
<tr>
<td>MAVEN_SETTINGS_XML</td>
<td>Location of custom Maven settings.xml file to use.</td>
<td>-</td>
<td>/home/jboss/.m2/settings.xml</td>
</tr>
<tr>
<td>NO_PROXY</td>
<td>A comma-separated lists of hosts, IP addresses or domains that can be accessed directly. This will be used for both Maven builds and Java runtime.</td>
<td>-</td>
<td>foo.example.com,bar.example.com</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>S2I_ARTIFACTS_DIR</td>
<td>Location mount for artifacts persisted with <code>save-artifacts</code> script, which are used with incremental builds. This should not be overridden by end users.</td>
<td>-</td>
<td><code>${S2I_DESTINATION_DIR}/artifacts</code></td>
</tr>
<tr>
<td>S2I_DESTINATION_DIR</td>
<td>Root directory for S2I mount, as specified by the <code>io.openshift.s2i.destination</code> label. This should not be overridden by end users.</td>
<td>-</td>
<td>/tmp</td>
</tr>
<tr>
<td>S2I_ENABLE_INCREMENTAL_BUILDS</td>
<td>Do not remove source and intermediate build files so they can be saved for use with future builds.</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>S2I_IMAGE_SOURCE_MOUNTS</td>
<td>Comma-separated list of relative paths in source directory that should be included in the image. List may include wildcards, which are expanded using find. By default, the contents of mounted directories are processed similarly to source folders, where the contents of <code>S2I_SOURCE_CONFIGURATION_DIR</code>, <code>S2I_SOURCE_DATA_DIR</code>, and <code>S2I_SOURCE_DEPLOYMENTS_DIR</code> are copied to their respective target directories. Alternatively, if an <code>install.sh</code> file is located in the root of the mount point, it is executed instead. Deprecates <code>CUSTOM_INSTALL_DIRECTORIES</code>.</td>
<td>-</td>
<td><code>extras/*</code></td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>S2I_SOURCE_CONFIGURATION_DIR</td>
<td>Relative path to directory containing application configuration files to be copied over to the product configuration directory, see S2I_TARGET_CONFIGURATION_DIR.</td>
<td>configuration</td>
<td>configuration</td>
</tr>
<tr>
<td>S2I_SOURCE_DATA_DIR</td>
<td>Relative path to directory containing application data files to be copied over to the product data directory, see S2I_TARGET_DATA_DIR.</td>
<td>data</td>
<td>data</td>
</tr>
<tr>
<td>S2I_SOURCE_DEPLOYMENTS_DIR</td>
<td>Relative path to directory containing binary files to be copied over to the product deployment directory, see S2I_TARGET_DEPLOYMENTS_DIR.</td>
<td>deployments</td>
<td>deployments</td>
</tr>
<tr>
<td>S2I_SOURCE_DIR</td>
<td>Location of mount for source code to be built. This should not be overridden by end users.</td>
<td>-</td>
<td>${S2I_DESTINATION_DIR}/src</td>
</tr>
<tr>
<td>S2I_TARGET_CONFIGURATION_DIR</td>
<td>Absolute path to which files located in S$S2I_SOURCE_DIR S$S2I_SOURCE_CONFIGURATION_DIR are copied.</td>
<td>-</td>
<td>/opt/eap/standalone/configuration</td>
</tr>
<tr>
<td>S2I_TARGET_DATA_DIR</td>
<td>Absolute path to which files located in S$S2I_SOURCE_DIR/$S2I_SOURCE_DATA_DIR are copied.</td>
<td>-</td>
<td>/opt/eap/standalone/data</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>S2I_TARGET_DEPLOYMENTS_DIR</td>
<td>Absolute path to which files located in $S2I_SOURCE_DIR/$S2I_SOURCE_DEPLOYMENTS_DIR are copied. Additionally, this is the directory to which build output is copied.</td>
<td>-</td>
<td>/deployments</td>
</tr>
<tr>
<td>http_proxy</td>
<td>The location of the HTTP proxy. This takes precedence over HTTP_PROXY and is use for both Maven builds and Java runtime.</td>
<td>-</td>
<td><a href="http://127.0.0.1:8080">http://127.0.0.1:8080</a></td>
</tr>
<tr>
<td>https_proxy</td>
<td>The location of the HTTPS proxy. This takes precedence over HTTPS_PROXY, http_proxy, and HTTP_PROXY, is use for both Maven builds and Java runtime.</td>
<td>-</td>
<td>myuser:mypass@127.0.0.1:8080</td>
</tr>
<tr>
<td>no_proxy</td>
<td>A comma-separated lists of hosts, IP addresses or domains that can be accessed directly. This takes precedence over NO_PROXY and is use for both Maven builds and Java runtime.</td>
<td>-</td>
<td>*.example.com</td>
</tr>
<tr>
<td>prefix_MAVEN_MIRROR_ID</td>
<td>ID to be used for the specified mirror. If omitted, a unique ID is generated.</td>
<td>-</td>
<td>internal-mirror</td>
</tr>
<tr>
<td>prefix_MAVEN_MIRROR_OF</td>
<td>Repository IDs mirrored by this entry.</td>
<td>external:*</td>
<td>-</td>
</tr>
<tr>
<td>prefix_MAVEN_MIRROR_URL</td>
<td>The URL of the mirror.</td>
<td>-</td>
<td><a href="http://10.0.0.1:8080/repository/internal">http://10.0.0.1:8080/repository/internal</a></td>
</tr>
<tr>
<td>prefix_MAVEN_REPOSITORY_PERMISSIONS</td>
<td>Maven repository directory permissions.</td>
<td>-</td>
<td>775</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_FILE_PERMISSIONS</td>
<td>Maven repository file permissions.</td>
<td>-</td>
<td>664</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_HOST</td>
<td>Maven repository host (if not using fully defined URL, it will fall back to service).</td>
<td>-</td>
<td>repo.example.com</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_ID</td>
<td>Maven repository id.</td>
<td>-</td>
<td>my-repo-id</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_LAYOUT</td>
<td>Maven repository layout.</td>
<td>-</td>
<td>default</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_NAME</td>
<td>Maven repository name.</td>
<td>-</td>
<td>my-repo-name</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_PASSPHRASE</td>
<td>Maven repository passphrase.</td>
<td>-</td>
<td>maven1!</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_PASSWORD</td>
<td>Maven repository password.</td>
<td>-</td>
<td>maven1!</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_PATH</td>
<td>Maven repository path (if not using fully defined URL, it will fall back to service).</td>
<td>-</td>
<td>/maven2/</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_PORT</td>
<td>Maven repository port (if not using fully defined URL, it will fall back to service).</td>
<td>-</td>
<td>8080</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_PRIVATE_KEY</td>
<td>Maven repository private key.</td>
<td>-</td>
<td>${user.home}/.ssh/id_rsa</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_PROTOCOL</td>
<td>Maven repository protocol (if not using fully defined URL, it will fall back to service).</td>
<td>-</td>
<td>http</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_RELEASES_CHECKSUM_POLICY</td>
<td>Maven repository releases checksum policy.</td>
<td>-</td>
<td>warn</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_RELEASES_ENABLED</td>
<td>Maven repository releases enabled.</td>
<td>-</td>
<td>true</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default value</td>
<td>Example value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_RELEASES_UPDATE_POLICY</td>
<td>Maven repository releases update policy.</td>
<td>-</td>
<td>always</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_SERVICE</td>
<td>Maven repository service to look up if prefix_MAVEN_REPO_URL not specified.</td>
<td>-</td>
<td>buscentr-myapp</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_SNAPSHOTS.Checksum_Policy</td>
<td>Maven repository snapshots checksum policy.</td>
<td>-</td>
<td>warn</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_SNAPSHOTS_ENABLED</td>
<td>Maven repository snapshots enabled.</td>
<td>-</td>
<td>true</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_SNAPSHOTS_UPDATE_POLICY</td>
<td>Maven repository snapshots update policy.</td>
<td>-</td>
<td>always</td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_URL</td>
<td>Maven repository URL (fully defined).</td>
<td>-</td>
<td><a href="http://repo.example.com:8080/maven2/">http://repo.example.com:8080/maven2/</a></td>
</tr>
<tr>
<td>prefix_MAVEN_REPO_USERNAME</td>
<td>Maven repository username.</td>
<td>-</td>
<td>mavenUser</td>
</tr>
</tbody>
</table>

### 5.3.1. Configuration environment variables with default values

The following configuration Environment variables have default values specified that can be overridden.

#### Table 5.4. Configuration environment variables with default values

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB_JOLOKIA_AUTH_OPENSHIFT</td>
<td>Switch on client authentication for OpenShift TLS communication. The value of this parameter can be a relative distinguished name which must be contained in a presented client’s certificate. Enabling this parameter will automatically switch Jolokia into HTTPS communication mode. The default CA cert is set to /var/run/secrets/kubernetes.io/serviceaccount/ca.crt.</td>
<td>true</td>
</tr>
</tbody>
</table>
### 5.4. EXPOSED PORTS

The following table lists the exposed ports.

<table>
<thead>
<tr>
<th>Port Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080</td>
<td>HTTP</td>
</tr>
<tr>
<td>8443</td>
<td>HTTPS</td>
</tr>
<tr>
<td>8778</td>
<td>Jolokia Monitoring</td>
</tr>
</tbody>
</table>

### 5.5. MAVEN SETTINGS

**Default Maven settings with Maven arguments**

The default value of `MAVEN_ARGS` environment variable contains the `-Dcom.redhat.xpaas.repo.redhatga` property. This property activates a profile with the [https://maven.repository.redhat.com/ga/](https://maven.repository.redhat.com/ga/) repository within the default `jboss-settings.xml` file, which resides in the Java S2I for OpenShift image.

When specifying a custom value for the `MAVEN_ARGS` environment variable, if a custom `source_dir/configuration/settings.xml` file is not specified, the default `jboss-settings.xml` in the image is used.
To specify which Maven repository will be used within the default jboss-settings.xml, there are two properties:

- The `-Dcom.redhat.xpaas.repo.redhatga` property, to use the https://maven.repository.redhat.com/ga/ repository.

- The `-Dcom.redhat.xpaas.repo.jbossorg` property to use the https://repository.jboss.org/nexus/content/groups/public/ repository.

Provide custom Maven settings
To specify a custom settings.xml file along with Maven arguments, create the `source_dir/configuration` directory and place the settings`.xml` file inside.

Sample path should be similar to: `source_dir/configuration/settings.xml`.

Revised on 2021-10-20 13:50:30 UTC