Red Hat build of OpenJDK 11

Using source-to-image for OpenShift with Red Hat build of OpenJDK 11

Last Updated: 2024-05-09

Legal Notice

Abstract

Red Hat build of OpenJDK 11 is a Red Hat offering on the Red Hat Enterprise Linux platform. The Using source-to-image for OpenShift with Red Hat build of OpenJDK 11 guide provides an overview of S2I for OpenShift and explains how to use S2I for OpenShift in Red Hat build of OpenJDK 11.

Table of Contents

PROVIDING FEEDBACK ON RED HAT BUILD OF OPENJDK DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

1. Click the following link to **create a ticket**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

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.

CHAPTER 1. INTRODUCTION TO SOURCE-TO-IMAGE FOR OPENSHIFT

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 S2I for OpenShift you can build and run basic Java applications, for example, **fat-jar** or **flat classpath** within a containerized image on OpenShift.

1.1. IMAGE STREAM DEFINITIONS

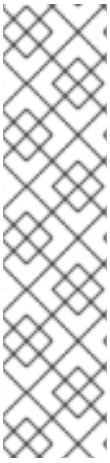
By default, the Red Hat OpenShift Container Platform includes image streams that contain the Red Hat build of OpenJDK container images.

You can import image stream definitions into a new namespace or recreate them. You can access these image stream templates on the **openjdk** GitHub page.

Red Hat OpenShift Container Platform includes **java** as an image stream, which follows the latest version of the container image. This image stream contains the following tags:

- **:latest**, which provides the latest supported Red Hat build of OpenJDK version. A tag tracks any updates for this image stream.
- **:11**, which provides latest JDK 11 images.
- **:8**, which provides the latest JDK 8 images.

The previous image stream and its tags are based on the latest version of the RHEL Universal Base Image (UBI).



NOTE

If you want to select a specific RHEL or Red Hat build of OpenJDK version, select a tag with the **openjdk-X-ubiY** format, where **X** refers to the Red Hat build of OpenJDK version and **Y** refers to the RHEL version.

The following examples demonstrate tags that follow this format:

- **openjdk-8-ubi8**
- **openjdk-11-ubi8**
- **openjdk-17-ubi8**

Specific image streams exist to accurately track the latest container image versions. These image streams follow the **ubiX-openjdk-Y** format, where **X** specifies the RHEL UBI version and **Y** specifies the Red Hat build of OpenJDK version. The following examples demonstrate image streams that follow this format:

- **ubi8-openjdk-8**
- **ubi8-openjdk-11**

- **ubi8-openjdk-17**

The tags for these image streams map directly to the image versions, such as **1.11**, **1.12**, and so on.

Additional resources

- Managing image streams (OpenShift Container Platform)
- templates (GitHub)

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, 4.7, and above 4.7 support the 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 SOURCE-TO-IMAGE FOR OPENSHIFT

You can use the 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 by using the 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:

```
$ oc login
```

2. Create a new project:

```
$ oc new-project <project-name>
```

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

```
$ oc new-app <source-location>
```

For example:

```
$ oc new-app --context-dir=getting-started --name=quarkus-quickstart \
 'registry.access.redhat.com/ubi8/openjdk-11~https://github.com/quarkusio/quarkus-quickstarts.git#2.12.1.Final'
```

4. Get the service name:

```
$ oc get svc
```

5. Expose the service as a route, so that you can use the server from your browser:

```
$ oc expose svc/ --port=8080
```

6. Get the route:

```
$ 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 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:

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

  - Build the application:

```
$ cd openshift-quickstarts/undertow-servlet/
```

```
$ mvn clean package
[INFO] Scanning for projects...
...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Undertow Servlet Example 1.0.0.Final
[INFO] ------------------------------------------------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.986 s
[INFO] Finished at: 2017-06-27T16:43:07+02:00
[INFO] Final Memory: 19M/281M
[INFO] ------------------------------------------------------------------------
```

- 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 a 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:

```
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 a new binary build, and specify the image stream and the application's 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 the **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).

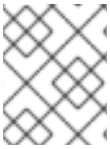
Additional resources

- 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.

CHAPTER 4. EXAMPLE WORKFLOWS FOR S2I ON OPENSHIFT

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 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 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 --context-dir=getting-started --name=quarkus-quickstart \
 'registry.access.redhat.com/ubi8/openjdk-11~https://github.com/quarkusio/quarkus-quickstarts.git#2.12.1.Final'
  -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-1-1uymm** as the pod name.

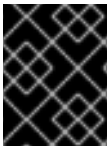
```
$ oc get pods
NAME                          READY     STATUS      RESTARTS   AGE
openshift-quickstarts-1-1uymm   1/1       Running     0          3m
openshift-quickstarts-1-build   0/1       Completed   0          6m
```

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 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 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 S2I for OpenShift image, see Configuring and Connecting the IDE Debugger.

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

The example in the procedure describes the process of running flat classpath java applications on 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 S2I for OpenShift image and Java source code:

```
$ oc new-app --context-dir=getting-started --name=quarkus-quickstart \
 'registry.access.redhat.com/ubi8/openjdk-11~https://github.com/quarkusio/quarkus-quickstarts.git#2.12.1.Final'
```

Post-deployment

Procedure

1. Get the service name:

```
$ oc get svc
```

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

Technology	Version
Red Hat build of OpenJDK	11
Jolokia	1.6.2
Maven	3.6

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

Variable Name	Value
HOME	/home/jboss
JAVA_HOME	/usr/lib/jvm/java-11
JAVA_VENDOR	openjdk
JAVA_VERSION	11
JOLOKIA_VERSION	1.6.2
LD_PRELOAD	libnss_wrapper.so
MAVEN_VERSION	3.6
NSS_WRAPPER_GROUP	/etc/group
NSS_WRAPPER_PASSWD	/home/jboss/passwd

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 5.3. Configuration environment variables

Variable name	Description	Default value	Example value
AB_JOLOKIA_CONFIG	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.	-	/opt/jolokia/custom.properties
AB_JOLOKIA_DISCOVERY_ENABLED	Enable Jolokia discovery.	false	true
AB_JOLOKIA_HOST	Host address to bind to.	0.0.0.0	127.0.0.1
AB_JOLOKIA_ID	Agent ID to use, which is the container id.	**$HOSTNAME**	openjdk-app-1-xqlsj
AB_JOLOKIA_OFF	If set disables activation of Joloka (that is, echos an empty value).	Jolokia is enabled	true
AB_JOLOKIA_OPTS	Additional options to be appended to the agent configuration. They should be specified in the format **key=value,key=value,...**.	-	backlog=20
AB_JOLOKIA_PASSWORD	Password for basic authentication. By default authentication is switched off.	-	mypassword
AB_JOLOKIA_PORT	Port to listen to.	8778	5432
AB_JOLOKIA_USER	User for basic authentication.	jolokia	myusername
AB_PROMETHEUS_ENABLE	Enable the use of the Prometheus agent.	-	True

Variable name	Description	Default value	Example value
AB_PROMETHEUS_JMX_EXPORTER_PORT	Port to use for the Prometheus JMX Exporter.	-	9799
CONTAINER_CORE_LIMIT	A calculated core limit as described in the CFS Bandwidth Control.	-	2
CONTAINER_MAX_MEMORY	Memory limit assigned to the container.	-	1024
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	The weighting given to the current garbage collector time versus previous garbage collector times.	-	90
GC_CONTAINER_OPTIONS	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.	-XX:+UseParallelOldGC	-XX:+UseG1GC
GC_MAX_HEAP_FREE_RATIO	Maximum percentage of heap free after GC to avoid shrinkage.	-	40
GC_MAX_METASPACE_SIZE	The maximum metaspace size.	-	100
GC_METASPACE_SIZE	The initial metaspace size.	-	20
GC_MIN_HEAP_FREE_RATIO	Minimum percentage of heap free after GC to avoid expansion.	-	20
GC_TIME_RATIO	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.	-	4

Variable name	Description	Default value	Example value
HTTPS_PROXY	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.	-	myuser@127.0.0.1:8080
HTTP_PROXY	The location of the HTTP proxy. This will be used for both Maven builds and Java runtime.	-	127.0.0.1:8080
JAVA_APP_DIR	The directory where the application resides. All paths in your application are relative to this directory.	-	myapplication/
JAVA_ARGS	Arguments passed to the **java** application.	-	-
JAVA_CLASSPATH	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/**).	-	-
JAVA_DEBUG	If set remote debugging will be switched on.	false	true
JAVA_DEBUG_PORT	Port used for remote debugging.	5005	8787
JAVA_DIAGNOSTICS	Set this to print some diagnostics information to standard output during the command is running.	false	true

Variable name	Description	Default value	Example value
JAVA_INITIAL_MEM_RATIO	It is used when no **-Xms** option is given in **JAVA_OPTS**. 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 **-Xms** is set to a ratio of the **-Xmx** memory as set here. The default is **25** which means 25% of the **-Xmx** is used as the initial heap size. You can skip this mechanism by setting this value to **0** in which case no **-Xms** option is added.	25	25
JAVA_LIB_DIR	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 **JAVA_LIB_DIR** is set to the value of **JAVA_APP_DIR**.	JAVA_APP_DIR	-
JAVA_MAIN_CLASS	A main class to use as argument for **java**. When this environment variable is given, all jar files in **JAVA_APP_DIR** are added to the classpath as well as **JAVA_LIB_DIR**.	-	com.example.MainClass

Variable name	Description	Default value	Example value
JAVA_MAX_INITIAL_MEM	It is used when no **-Xms** option is given in **JAVA_OPTS**. 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 **-Xms** is limited to the value set here. The default is 4096 which means the calculated value of **-Xms** will never be greater than 4096. The value of this variable is expressed in MB.	4096	4096
JAVA_MAX_MEM_RATIO	It is used when no **-Xmx** option is given in **JAVA_OPTS**. This is used to calculate a default maximum heap memory based on a containers 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 **-Xmx** 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 **-Xmx** option is added.	50	-

Variable name	Description	Default value	Example value
JAVA_OPTS	JVM options passed to the **java** command.	-	-verbose:class
JAVA_OPTS_APPEND	User-specified Java options to be appended to generated options in JAVA_OPTS.	-	-Dsome.property=foo
LOGGING_SCRIPT_DEBUG	Set to true to enable script debugging. Deprecates **SCRIPT_DEBUG**.	true	True
MAVEN_ARGS	Arguments to use when calling Maven, replacing the default **package hawt-app:build -DskipTests -e**. Ensure that you run the **hawt-app:build** goal (when not already bound to the **package** execution phase), otherwise the startup scripts will not work.	package hawt-app:build -DskipTests -e	-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package
MAVEN_ARGS_APPEND	Additional Maven arguments.	-	-X -am -pl
MAVEN_CLEAR_REPO	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 **S2I_ENABLE_INCREMENTAL_BUILDS**.	false	-
MAVEN_LOCAL_REPO	Directory to use as the local Maven repository.	-	/home/jboss/.m2/repository

Variable name	Description	Default value	Example value
MAVEN_MIRRORS	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**.	-	dev-one,qe-two
MAVEN_MIRROR_URL	The base URL of a mirror used for retrieving artifacts.	-	http://10.0.0.1:8080/repository/internal/
MAVEN_REPOS	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**.	-	dev-one,qe-two
MAVEN_S2I_ARTIFACT_DIRS	Relative paths of source directories to scan for build output, which will be copied to **$DEPLOY_DIR**.	target	target
MAVEN_S2I_GOALS	Space-separated list of goals to be executed with Maven build. For example, **mvn $MAVEN_S2I_GOALS**.	package	package install
MAVEN_SETTINGS_XML	Location of custom Maven **settings.xml** file to use.	-	/home/jboss/.m2/settings.xml
NO_PROXY	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.	-	foo.example.com,bar.example.com

Variable name	Description	Default value	Example value
S2I_ARTIFACTS_DIR	Location mount for artifacts persisted with *save-artifacts* script, which are used with incremental builds. This should not be overridden by end users.	-	**${S2I_DESTINATION_DIR}/artifacts}**
S2I_DESTINATION_DIR	Root directory for S2I mount, as specified by the **io.openshift.s2i.destination** label. This should not be overridden by end users.	-	**/tmp**
S2I_ENABLE_INCREMENTAL_BUILDS	Do not remove source and intermediate build files so they can be saved for use with future builds.	true	true
S2I_IMAGE_SOURCE_MOUNTS	Comma-separated list of relative paths in source directory that should be included in the image. List may include wildcards, which are expanded using find. By default, the contents of mounted directories are processed similarly to source folders, where the contents of **$S2I_SOURCE_CONFIGURATION_DIR**, **$S2I_SOURCE_DATA_DIR**, and **$S2I_SOURCE_DEPLOYMENTS_DIR** are copied to their respective target directories. Alternatively, if an **install.sh** file is located in the root of the mount point, it is executed instead. Deprecates **CUSTOM_INSTALL_DIRECTORIES**.	-	**extras/*`**

Variable name	Description	Default value	Example value
S2I_SOURCE_CONFIGURATION_DIR	Relative path to directory containing application configuration files to be copied over to the product configuration directory, see **S2I_TARGET_CONFIGURATION_DIR**.	configuration	configuration
S2I_SOURCE_DATA_DIR	Relative path to directory containing application data files to be copied over to the product data directory, see **S2I_TARGET_DATA_DIR**.	data	data
S2I_SOURCE_DEPLOYMENTS_DIR	Relative path to directory containing binary files to be copied over to the product deployment directory, see **S2I_TARGET_DEPLOYMENTS_DIR**.	deployments	deployments
S2I_SOURCE_DIR	Location of mount for source code to be built. This should not be overridden by end users.	-	${S2I_DESTINATION_DIR}/src}
S2I_TARGET_CONFIGURATION_DIR	Absolute path to which files located in **$S2I_SOURCE_DIR $S2I_SOURCE_CONFIGURATION_DIR** are copied.	-	/opt/eap/standalone/configuration
S2I_TARGET_DATA_DIR	Absolute path to which files located in **$S2I_SOURCE_DIR/$S2I_SOURCE_DATA_DIR** are copied.	-	/opt/eap/standalone/data

Variable name	Description	Default value	Example value
S2I_TARGET_DEPLOYMENTS_DIR	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.	-	/deployments
http_proxy	The location of the HTTP proxy. This takes precedence over *HTTP_PROXY* and is use for both Maven builds and Java runtime.	-	http://127.0.0.1:8080
https_proxy	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.	-	myuser:mypass@127.0.0.1:8080
no_proxy	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.	-	*.example.com
prefix_MAVEN_MIRROR_ID	ID to be used for the specified mirror. If omitted, a unique ID is generated.	-	internal-mirror
prefix_MAVEN_MIRROR_OF	Repository IDs mirrored by this entry.	external:*	-
prefix_MAVEN_MIRROR_URL	The URL of the mirror.	-	http://10.0.0.1:8080/repository/internal
prefix_MAVEN_REPO_DIRECTORY_PERMISSIONS	Maven repository directory permissions.	-	775

Variable name	Description	Default value	Example value
prefix_MAVEN_REPO_FILE_PERMISSIONS	Maven repository file permissions.	-	664
prefix_MAVEN_REPO_HOST	Maven repository host (if not using fully defined URL, it will fall back to service).	-	repo.example.com
prefix_MAVEN_REPO_ID	Maven repository id.	-	my-repo-id
prefix_MAVEN_REPO_LAYOUT	Maven repository layout.	-	default
prefix_MAVEN_REPO_NAME	Maven repository name.	-	my-repo-name
prefix_MAVEN_REPO_PASSPHRASE	Maven repository passphrase.	-	maven1!
prefix_MAVEN_REPO_PASSWORD	Maven repository password.	-	maven1!
prefix_MAVEN_REPO_PATH	Maven repository path (if not using fully defined URL, it will fall back to service).	-	/maven2/
prefix_MAVEN_REPO_PORT	Maven repository port (if not using fully defined URL, it will fall back to service).	-	8080
prefix_MAVEN_REPO_PRIVATE_KEY	Maven repository private key.	-	${user.home}/.ssh/id_dsa
prefix_MAVEN_REPO_PROTOCOL	Maven repository protocol (if not using fully defined URL, it will fall back to service).	-	http
prefix_MAVEN_REPO_RELEASES_CHECKSUM_POLICY	Maven repository releases checksum policy.	-	warn
prefix_MAVEN_REPO_RELEASES_ENABLED	Maven repository releases enabled.	-	true

Variable name	Description	Default value	Example value
prefix_MAVEN_REPO_RELEASES_UPDATE_POLICY	Maven repository releases update policy.	-	always
prefix_MAVEN_REPO_SERVICE	Maven repository service to look up if prefix_MAVEN_REPO_URL not specified.	-	buscentr-myapp
prefix_MAVEN_REPO_SNAPSHOTS_CHECKSUM_POLICY	Maven repository snapshots checksum policy.	-	warn
prefix_MAVEN_REPO_SNAPSHOTS_ENABLED	Maven repository snapshots enabled.	-	true
prefix_MAVEN_REPO_SNAPSHOTS_UPDATE_POLICY	Maven repository snapshots update policy.	-	always
prefix_MAVEN_REPO_URL	Maven repository URL (fully defined).	-	http://repo.example.com:8080/maven2/
prefix_MAVEN_REPO_USERNAME	Maven repository username.	-	mavenUser

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

Variable name	Description	Defaul value
AB_JOLOKIA_AUTH_OPENSHIFT	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**.	true

Variable name	Description	Defaul value
AB_JOLOKIA_HTTPS	Switch on secure communication with HTTPS. By default self-signed server certificates are generated if no **serverCert** configuration is given in *AB_JOLOKIA_OPTS*.	true
AB_JOLOKIA_PASSWORD_RANDOM	Determines if a random **AB_JOLOKIA_PASSWORD** should be generated. Set to *true* to generate random password. Generated value will be written to **/opt/jolokia/etc/jolokia.pw**.	true
AB_PROMETHEUS_JMX_EXPORTER_CONFIG	Path to configuration to use for the Prometheus JMX exporter.	/opt/jboss/container/prometheus/etc/jmx-exporter-config.yaml
S2I_SOURCE_DEPLOYMENTS_FILTER	Space-separated list of filters to be applied when copying deployments. Defaults to *.	*

5.4. EXPOSED PORTS

The following table lists the exposed ports.

Port Number	Description
8080	HTTP
8443	HTTPS
8778	Jolokia Monitoring

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/** repository within the default **jboss-settings.xml** file, which resides in the 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 2024-05-09 16:48:40 UTC