Red Hat build of OpenJDK 11

Red Hat ビルドの OpenJDK 11 での OpenShift の Source-to-Image の使用

Last Updated: 2024-05-10

Red Hat build of OpenJDK 11 Red Hat ビルドの OpenJDK 11 での OpenShift の Source-to-Image の使用

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

Red Hat build of OpenJDK 17 の Red Hat ビルドは、Red Hat Enterprise Linux プラットフォーム上の Red Hat 製品です。OpenJDK 11 での OpenShift での source-to-image の使用 ガイドでは、OpenShift 向けの S2I の概要と、Red Hat ビルドの OpenJDK 11 で OpenShift に S2I を使用する方法について説明します。

目次

RED HAT BUILD OF OPENJDK ドキュメントへのフィードバック

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

1. 次のリンクをクリックして **チケットを作成します**。
2. **Summary** に課題の簡単な説明を入力します。
3. **Description** に課題や機能拡張の詳細な説明を入力します。問題があるドキュメントのセクションへの URL を含めてください。
4. **Submit** をクリックすると、課題が作成され、適切なドキュメントチームに転送されます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ を参照してください。

第1章 OPENSHIFT 用 SOURCE-TO-IMAGE の概要

OpenShift Container Platform は、アプリケーションをビルドして実行する Source-to-Image (S2I) プロセスを提供します。ビルダーイメージ (JBoss EAP などのテクノロジーイメージ) にアプリケーションのソースコードを関連付けることができます。S2I プロセスは、アプリケーションをビルドし、ビルダーイメージの上に階層化してアプリケーションイメージを作成します。アプリケーションイメージのビルド後に、これを OpenShift 内の 統合レジストリー にプッシュするか、スタンドアロンレジストリー にプッシュできます。

S2I for OpenShift を使用すると、OpenShift のコンテナー化されたイメージ内に **jat-jar** や **flat classpath** などの基本的な Java アプリケーションをビルドし、実行できます。

1.1. イメージストリームの定義

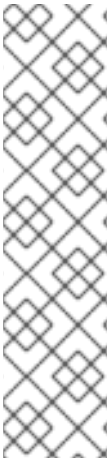
デフォルトでは、Red Hat OpenShift Container Platform には、Red Hat OpenJDK コンテナーイメージなどのイメージストリームが含まれています。

イメージストリーム定義を新しい namespace にインポートするか、再作成できます。これらのイメージストリームテンプレートには、**openjdk** GitHub ページからアクセスできます。

Red Hat OpenShift Container Platform にはイメージストリームとして **java** が含まれており、コンテナーイメージの最新バージョンを使用します。このイメージストリームには次のタグが含まれています。

- **:最新** の OpenJDK バージョンをサポートする最新の Red Hat ビルドを提供します。このタグは、このイメージストリームの更新を追跡します。
- **:11**。最新の JDK 11 イメージを提供します。
- **:8**。最新の JDK 8 イメージを提供します。

以前のイメージストリームとそのタグは、RHEL Universal Base Image (UBI) の最新バージョンに基づいています。



注記

特定の RHEL または Red Hat ビルドの OpenJDK バージョンを選択する場合は、**openjdk-X-ubiY** 形式のタグを選択します。**X** は OpenJDK バージョンの Red Hat ビルドを、**Y** は RHEL バージョンを指します。

次の例は、このの形式に従うタグを示しています。

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

最新のコンテナーイメージバージョンを正確に追跡するために、特定のイメージストリームが存在します。これらのイメージストリームは **ubiX-openjdk-Y** 形式に従います。ここで、**X** は RHEL UBI バージョンを、**Y** は OpenJDK バージョンを指定します。次の例は、この形式に従うイメージストリームを示しています。

- **ubi8-openjdk-8**

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

これらのイメージストリームのタグは、**1.11**、**1.12** などのイメージバージョンに直接マップされます。

関連情報

- イメージストリームの管理 (OpenShift Container Platform)
- テンプレート (GitHub)

第2章 作業を開始する前に

初期設定

OpenShift インスタンスを作成します。OpenShift インスタンスの作成方法についての詳細は、OpenShift container platform installation overview を参照してください。

バージョンの互換性とサポート

OpenShift Container Platform バージョン 3.11、4.7、および 4.7 以降は、S2I for OpenShift イメージをサポートします。

OpenShift Container Platform の現在のサポートレベルの詳細は、Red Hat OpenShift Container Platform Life Cycle Policy および Red Hat OpenShift Container Platform Life Cycle Policy (non-current versions) を参照してください。

第3章 OPENSHIFT での SOURCE-TO-IMAGE の使用

OpenShift イメージの Source-to-Image (S2I) を使用して、OpenShift でカスタムの Java アプリケーションを実行できます。

3.1. OPENSHIFT の SOURCE-TO-IMAGE を使用した JAVA アプリケーションのビルドおよびデプロイ

OpenShift イメージの Source-to-Image (S2I) を使用して OpenShift で ソースから Java アプリケーションをビルドおよびデプロイするには、OpenShift S2I プロセスを使用します。

手順

1. 以下のコマンドを実行して OpenShift インスタンスにログインし、認証情報を指定します。

```
$ oc login
```

2. 新しいプロジェクトを作成します。

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

3. S2I for OpenShift イメージを使用して新規アプリケーションを作成します。
<source-location> は、GitHub リポジトリーの URL、またはローカルフォルダーへのパスです。

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

以下に例を示します。

```
$ 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. サービス名を取得します。

```
$ oc get svc
```

5. ブラウザーからサーバーを使用できるように、サービスをルートとして公開します。

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

6. ルートを取得します。

```
$ oc get route
```

7. URL を使用してブラウザーでアプリケーションにアクセスします。前のコマンド出力にある **HOST/PORT** フィールドの値を使用します。

関連情報

- 詳細な例は、Running flat classpath JAR on source-to-image for OpenShift を参照してください。

3.2. バイナリーアーティファクトからの JAVA アプリケーションのビルドおよびデプロイ

バイナリーソース機能を使用して、既存の Java アプリケーションを OpenShift にデプロイできます。

この手順では、undertow-servlet クイックスタートを使用してローカルマシンで Java アプリケーションを構築します。クイックスタートは、S2I バイナリーソース機能を使用して、作成されたバイナリーアーティファクトを OpenShift にコピーします。

前提条件

- ローカルマシンで Red Hat JBoss Enterprise Maven リポジトリーを有効にします。
- JAR アプリケーションアーカイブを取得し、アプリケーションをローカルにビルドします。
  - **undertow-servlet** ソースコードのクローンを作成します。

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

  - アプリケーションをビルドします。

```
$ 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] ------------------------------------------------------------------------
```

- ローカルファイルシステムでディレクトリー構造を準備します。
deployments/ サブディレクトリー (メインのバイナリービルドディレクトリー) のアプリケーションアーカイブを標準のデプロイメントフォルダー (イメージが OpenShift 上にビルドされる場所) にコピーします。デプロイするアプリケーションの Web アプリケーションデータを含むディレクトリー階層を作成します。

  ローカルファイルシステム上にバイナリービルド用のメインディレクトリーと、そのディレクトリー内に **deployments/** サブディレクトリーを作成します。ビルド JAR アーカイブを **deployments/** サブディレクトリーにコピーします。

```
undertow-servlet]$ ls
dependency-reduced-pom.xml  pom.xml  README  src  target
```

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

```
$ cp target/undertow-servlet.jar ocp/deployments/
```

手順

1. 以下のコマンドを実行して OpenShift インスタンスにログインし、認証情報を指定します。

```
$ oc login
```

2. 新しいプロジェクトを作成します。

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

3. 新しいバイナリービルドを作成し、イメージストリームとアプリケーションの名前を指定します。

```
$ 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. バイナリービルドを開始します。
直前の手順で作成したバイナリービルドのメインディレクトリーを OpenShift ビルドのバイナリー入力が含まれるディレクトリーとして使用するよう **oc** 実行ファイルに指示します。

```
$ 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. ビルドに基づいて新規の OpenShift アプリケーションを作成します。

```
$ 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. サービスをルートとして公開します。

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

7. ルートを取得します。

```
$ oc get route
```

8. URL(直前のコマンド出力にある **HOST/PORT** フィールドの値) を使用して、ブラウザーでアプリケーションにアクセスします。

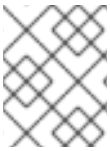
関連情報

- バイナリーソース 機能を使用して、既存の Java アプリケーションを OpenShift にデプロイします。
- Maven リポジトリーの設定方法に関する詳細は、Use the Maven Repository を参照してください。

第4章 OPENSHIFT 上の S2I のワークフロー例

4.1. OPENSHIFT イメージのリモートデバッグ JAVA アプリケーション

この手順の例では、S2I for OpenShift イメージを使用して、OpenShift にデプロイされた Java アプリケーションのリモートデバッグを示しています。環境変数 **JAVA_DEBUG** の値を **true** に設定し、**JAVA_DEBUG_PORT** を **9009** に設定すると、この機能を有効にできます。



注記

JAVA_DEBUG 変数を true に設定され、**JAVA_DEBUG_PORT** 変数に値が提供されていない場合、**JAVA_DEBUG_PORT** 変数はデフォルトで **5005** に設定されます。

デプロイメントの準備

手順

1. 以下のコマンドを実行して OpenShift インスタンスにログインし、認証情報を指定します。

```
$ oc login
```

2. 新しいプロジェクトを作成します。

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

Deployment

新規および既存のアプリケーションのリモートデバッグを有効にできます。

新規アプリケーションのリモートデバッグの有効化

手順

- S2I for OpenShift イメージと Java ソースコードのサンプルを使用して、新規アプリケーションを作成します。アプリケーションを作成する前に、**JAVA_DEBUG** および **JAVA_DEBUG_PORT** 環境変数を設定してください。

```
$ 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
```

ローカルデバッグポートを Pod のポートに接続 に進みます。

既存アプリケーションのリモートデバッグの有効化

手順

1. 適切な OpenShift プロジェクトに切り替えます。

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

2. **deploymentconfig** の名前を取得します。

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

3. **deploymentconfig** を編集し、**JAVA_DEBUG=true** および **JAVA_DEBUG_PORT=9009** 環境変数を追加します。

4. **.spec.template.spec.containers** パスと **Container** のタイプで編集するオブジェクトを指定します。

```
$ oc edit dc/openshift-quickstarts
```

注記

エディターを起動し、ターミナルで **oc edit** コマンドを実行します。環境の **EDITOR** 変数を定義することで、起動するエディターを変更できます。

ローカルデバッグポートを Pod のポートに接続 に進みます。

デプロイメント後
Pod のポートへのローカルデバッグポートの接続

手順

1. アプリケーションを実行している Pod の名前を取得します (**Running**の状況)。
Pod 名 として表示される **openshift-quickstarts-1-1uymm** の例

```
$ 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. OpenShift または Kubernetes ポート転送機能を使用してローカルポートでリッスンし、OpenShift Pod のポートに転送します。**<running-pod>** は、直前のコマンド出力から Status "running" が設定された Pod の **NAME** フィールドの値です。

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

注記

上記の例では、**5005** はローカルシステムのポート番号ですが **9009** は OpenShift イメージの S2I を実行する OpenShift Pod のリモートポート番号です。そのため、ローカルポート **5005** に送信される今後のデバッグ接続は、Java 仮想マシン (JVM) を実行する OpenShift Pod のポート **9009** に転送されます。

重要

このコマンドにより、ターミナルでそれ以上入力できなくなる場合があります。この場合は、以下の手順を実行するために新しいターミナルを開きます。

アプリケーションへのデバッガーの割り当て

手順

1. ローカルシステムのデバッガーを、S2I for OpenShift イメージで実行しているリモート JVM に割り当てます。

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

注記

リモート OpenShift Pod へのローカルデバッガーが開始すると、以前の oc port-forward コマンドが発行されたコンソールに **5005** 接続を処理するのと同様のエントリーが表示されます。

2. アプリケーションをデバッグします。

```
$ 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
>
```

関連情報

- Openshift の共通オブジェクト参照に関する詳細は、OpenShift Common Object Reference, section Container セクションを参照してください。
- Red Hat JBoss Developer Studio の IDE デバッガーを S2I for OpenShift イメージを実行する OpenShift Pod に接続する方法は、Configuring and Connecting the IDE Debugger を参照してください。

4.2. OPENSHIFT 用の SOURCE-TO-IMAGE でのフラットクラスパス JAR の実行

この手順の例では、S2I for OpenShift でフラットなクラスパスの java アプリケーションを実行するプロセスを説明します。

デプロイメントの準備

手順

1. 認証情報を指定して OpenShift インスタンスにログインします。

```
$ oc login
```

2. 新しいプロジェクトを作成します。

```
$ oc new-project js2i-flatclasspath-demo
```

Deployment

手順

1. S2I for OpenShift イメージおよび Java ソースコードを使用して新規アプリケーションを作成します。

```
$ 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'
```

デプロイメント後

手順

1. サービス名を取得します。

```
$ oc get svc
```

2. サービスをブラウザーから使用できるようにルートとして公開します。

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

3. ルートを取得します。

```
$ oc get route
```

4. URL(前のコマンド出力にある **HOST/PORT** フィールドの値) を使用して、ブラウザーでアプリケーションにアクセスします。

第5章 参照資料

5.1. バージョンの詳細

以下の表は、このイメージで使用されるテクノロジーのバージョンを示しています。

表5.1 このイメージで使用されるテクノロジーバージョン

Technology	バージョン
Red Hat ビルドの OpenJDK	11
Jolokia	1.6.2
Maven	3.6

5.2. 情報環境変数

以下の情報環境変数は、イメージに関する情報を伝達するように設計されています。これらの変数は変更しないでください。

表5.2 情報環境変数

変数名	値
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. 設定環境変数

設定環境変数は、再ビルドを必要とせずにイメージを便利に調整するように設計されており、必要に応じてユーザーが設定する必要があります。

表5.3 設定環境変数

変数名	説明	デフォルト値	値の例
AB_JOLOKIA_CONFIG	設定された場合、(Jolokia の リファレンスマニュアル で説明されているように) (パスを含む) このファイルを Jolokia JVM エージェントプロパティーとして使用します。設定しない場合、マニュアルで定義された設定を使用して、**/opt/jolokia/etc/jolokia.properties** が作成されます。それ以外の場合、本書の残りの設定は無視されます。	-	/opt/jolokia/custom.properties
AB_JOLOKIA_DISCOVERY_ENABLED	Jolokia の検索を有効にします。	false	true
AB_JOLOKIA_HOST	バインド先のホストアドレス。	0.0.0.0	127.0.0.1
AB_JOLOKIA_ID	使用するエージェント ID(コンテナー ID)。	**$HOSTNAME**	openjdk-app-1-xqlsj
AB_JOLOKIA_OFF	Joloka のアクティベーションを無効にする場合 (つまり、空の値がエコーされます)。	Jolokia が有効になっている	true
AB_JOLOKIA_OPTS	エージェント設定に追加されるその他のオプション。**key=value,key=value,...** の形式で指定する必要があります。	-	backlog=20
AB_JOLOKIA_PASSWORD	Basic 認証のパスワード。デフォルトでは認証は無効になっています。	-	mypassword
AB_JOLOKIA_PORT	リッスンするポート。	8778	5432
AB_JOLOKIA_USER	Basic 認証のユーザー。	jolokia	myusername
AB_PROMETHEUS_ENABLE	Prometheus エージェントの使用を有効にします。	-	True

変数名	説明	デフォルト値	値の例
AB_PROMETHEUS_JMX_EXPORTER_PORT	Prometheus JMX Exporter に使用するポート。	-	9799
CONTAINER_CORE_LIMIT	CFS Bandwidth Control で説明されているように、計算されたコア制限。	-	2
CONTAINER_MAX_MEMORY	コンテナーに割り当てられるメモリー制限。	-	1024
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	現在のガベージコレクター時間と以前のガベージコレクター時間に指定される重み。	-	90
GC_CONTAINER_OPTIONS	使用する Java GC を指定します。この変数の値には、必要な GC を指定するのに必要な JRE コマンドラインオプションが含まれ、デフォルト値を上書きする必要があります。	-XX:+UseParallelOldGC	-XX:+UseG1GC
GC_MAX_HEAP_FREE_RATIO	縮小を回避するための GC 後のヒープ解放の最大パーセンテージ。	-	40
GC_MAX_METASPACE_SIZE	メタスペースの最大サイズ。	-	100
GC_METASPACE_SIZE	初期メタスペースのサイズ。	-	20
GC_MIN_HEAP_FREE_RATIO	拡大を回避するための GC 後のヒープ解放の最小パーセンテージ。	-	20
GC_TIME_RATIO	ガベージコレクションに費やした時間に対する、ガベージコレクション外で費やした時間 (アプリケーションの実行に費やした時間など) の比率を指定します。	-	4

変数名	説明	デフォルト値	値の例
HTTPS_PROXY	HTTPS プロキシーの場所。これは **http_proxy** および **HTTP_PROXY** に優先され、Maven ビルドと Java ランタイムの両方に使用されます。	-	myuser@127.0.0.1:8080
HTTP_PROXY	HTT プロキシーの場所。これは、Maven ビルドと Java ランタイムの両方に使用されます。	-	127.0.0.1:8080
JAVA_APP_DIR	アプリケーションがあるディレクトリー。アプリケーションのすべてのパスは、このディレクトリーに相対的です。	-	myapplication/
JAVA_ARGS	**java** アプリケーションに渡される引数。	-	-
JAVA_CLASSPATH	使用するクラスパス。指定しない場合、起動スクリプトは **JAVA_APP_DIR/classpath** ファイルを確認し、その内容をそのままクラスパスとして使用します。このファイルが存在しない場合は、app ディレクトリーのすべての jar が追加されます (**classes:JAVA_APP_DIR/**)。	-	-
JAVA_DEBUG	設定されている場合は、リモートデバッグが有効になります。	false	true
JAVA_DEBUG_PORT	リモートデバッグに使用されるポート。	5005	8787
JAVA_DIAGNOSTICS	これを設定して、コマンドの実行中に一部の診断情報を標準出力に出力します。	false	true

変数名	説明	デフォルト値	値の例
JAVA_INITIAL_MEM_RATIO	**JAVA_OPTS** に **-Xms** オプションが指定されていない場合に使用されます。これは、最大ヒープメモリーを基にデフォルトの初期ヒープメモリーを算出するために使用されます。コンテナーのメモリー制約のないコンテナーで使用されると、このオプションは効果がありません。メモリー制約がある場合、**-Xms** はここで設定されている **-Xmx** メモリーの比率に設定されます。デフォルトは **25** で、**-Xmx** の 25% が初期ヒープサイズとして使用されることを意味します。このメカニズムを省略するには、この値を **0** に設定します。この場合、**-Xms** オプションは追加されません。	25	25
JAVA_LIB_DIR	Java jar ファイルと、クラスパスを保持するオプションの **classpath** ファイルを保持するディレクトリー。単一行のクラスパス (コロン区切り) または行ごとにリストされた jar ファイルのいずれかになります。設定されていない場合は、**JAVA_LIB_DIR** が **JAVA_APP_DIR** の値に設定されます。	JAVA_APP_DIR	-
JAVA_MAIN_CLASS	**java** の引数として使用するメインクラス。この環境変数が指定されると、**JAVA_APP_DIR** のすべての jar ファイルがクラスパスと **JAVA_LIB_DIR** に追加されます。	-	com.example.MainClass

変数名	説明	デフォルト値	値の例
JAVA_MAX_INITIAL_MEM	**JAVA_OPTS** に **-Xms** オプションが指定されていない場合に使用されます。これは、初期ヒープメモリーの最大値を算出するために使用されます。コンテナーのメモリー制約のないコンテナーで使用されると、このオプションは効果がありません。メモリー制約がある場合、**-Xms** はここで設定されている値に制限されます。デフォルトは 4096 です。これは、**-Xms** の計算された値が 4096 を超えることはありません。この変数の値は、MB で表されます。	4096	4096
JAVA_MAX_MEM_RATIO	**JAVA_OPTS** に **-Xmx** オプションが指定されていない場合に使用されます。これは、コンテナーの制限に基づいてデフォルトの最大ヒープメモリーを算出するために使用されます。コンテナーのメモリー制約のないコンテナーで使用されると、このオプションは効果がありません。メモリー制約がある場合、**-Xmx** はコンテナーで利用可能なメモリーの比率をここで設定します。デフォルト値の **50** は、利用可能なメモリーの 50% が上限として使用されることを意味します。このメカニズムを省略するには、この値を **0** に設定します。この場合、**-Xmx** オプションは追加されません。	50	-
JAVA_OPTS	**java** コマンドに渡される JVM オプション。	-	-verbose:class

変数名	説明	デフォルト値	値の例
JAVA_OPTS_APPEND	JAVA_OPTS で生成されたオプションに追加されるユーザー指定の Java オプション。	-	-Dsome.property=foo
LOGGING_SCRIPT_DEBUG	スクリプトのデバッグを有効にするには true に設定します。**SCRIPT_DEBUG** を非推奨にします。	true	True
MAVEN_ARGS	Maven の呼び出し時に使用する引数。デフォルトの **package hawt-app:build -DskipTests -e** を置き換えます。(**package** 実行フェーズにバインドされていない場合は)**hawt-app:build** ゴールを実行していることを確認してください。そうしないと、起動スクリプトが機能しません。	package hawt-app:build -DskipTests -e	-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package
MAVEN_ARGS_APPEND	追加の Maven 引数。	-	-X -am -pl
MAVEN_CLEAR_REPO	設定すると、アーティファクトのビルド後に Maven リポジトリーが削除されます。これは、作成されたアプリケーションイメージのサイズを小さくするために便利ですが、**増分** ビルドを防ぎます。**S2I_ENABLE_INCREMENTAL_BUILDS** で上書きされます。	false	-
MAVEN_LOCAL_REPO	ローカルの Maven リポジトリーとして使用するディレクトリー。	-	/home/jboss/.m2/repository

変数名	説明	デフォルト値	値の例
MAVEN_MIRRORS	設定すると、マルチミラーサポートが有効になり、他の **MAVEN_MIRROR_*** 変数に接頭辞が付けられます。たとえば、**DEV_ONE_MAVEN_MIRROR_URL** および **QE_TWO_MAVEN_MIRROR_URL** のようになります。	-	dev-one、qe-two
MAVEN_MIRROR_URL	アーティファクトの取得に使用されるミラーのベース URL。	-	http://10.0.0.1:8080/repository/internal/
MAVEN_REPOS	設定すると、マルチリポジトリーサポートが有効になり、他の **MAVEN_REPO_*** 変数に接頭辞が付けられます。たとえば、**DEV_ONE_MAVEN_REPO_URL** および **QE_TWO_MAVEN_REPO_URL** のようになります。	-	dev-one、qe-two
MAVEN_S2I_ARTIFACT_DIRS	ビルド出力用にスキャンするソースディレクトリーの相対パス。これは **$DEPLOY_DIR** にコピーされます。	target	target
MAVEN_S2I_GOALS	Maven ビルドで実行されるゴールのスペース区切りリスト。例: **mvn $MAVEN_S2I_GOALS**	package	package install
MAVEN_SETTINGS_XML	使用するカスタムの Maven **settings.xml** ファイルの場所。	-	/home/jboss/.m2/settings.xml

変数名	説明	デフォルト値	値の例
NO_PROXY	直接アクセスできるホスト、IP アドレス、またはドメインのコンマ区切りリスト。これは、Maven ビルドと Java ランタイムの両方に使用されます。	-	foo.example.com,bar.example.com
S2I_ARTIFACTS_DIR	アーティファクトのロケーションマウントは、インクリメンタルビルドで使用される **save-artifacts** スクリプトで永続化されます。これはエンドユーザーで上書きしないでください。	-	**${S2I_DESTINATION_DIR}/artifacts}**
S2I_DESTINATION_DIR	**io.openshift.s2i.destination** ラベルで指定された S2I マウントの root ディレクトリー。これはエンドユーザーで上書きしないでください。	-	**/tmp**
S2I_ENABLE_INCREMENTAL_BUILDS	将来のビルドで使用するために保存できるように、ソースおよび中間ビルドファイルを削除しないでください。	true	true

変数名	説明	デフォルト値	値の例
S2I_IMAGE_SOURCE_MOUNTS	イメージに含めるべきソースディレクトリーの相対パスのコンマ区切りリスト。リストにはワイルドカードが含まれる可能性があります。これは find を使用してデプロイメントされます。デフォルトでは、マウントされたディレクトリーの内容は、ソースフォルダーと同様に処理されます。ここでは、**$S2I_SOURCE_CONFIGURATION_DIR**、**$S2I_SOURCE_DATA_DIR**、および **$S2I_SOURCE_DEPLOYMENTS_DIR** の内容がそれぞれのターゲットディレクトリーにコピーされます。または、**install.sh** ファイルがマウントポイントのルートにある場合は、代わりに実行されます。**CUSTOM_INSTALL_DIRECTORIES** を非推奨にします。	-	**extras/*`**
S2I_SOURCE_CONFIGURATION_DIR	製品設定ディレクトリーにコピーされるアプリケーション設定ファイルを含むディレクトリーへの相対パス。**S2I_TARGET_CONFIGURATION_DIR** を参照してください。	configuration	configuration
S2I_SOURCE_DATA_DIR	製品データディレクトリーにコピーされるアプリケーションデータファイルを含むディレクトリーへの相対パス。**S2I_TARGET_DATA_DIR** を参照してください。	data	data

変数名	説明	デフォルト値	値の例
S2I_SOURCE_DEPLOYMENTS_DIR	製品デプロイメントディレクトリーにコピーされるバイナリーファイルを含むディレクトリーへの相対パス。**S2I_TARGET_DATA_DIR** を参照してください。	デプロイメント	デプロイメント
S2I_SOURCE_DIR	ビルドするソースコードのマウントの場所。これはエンドユーザーで上書きしないでください。	-	${S2I_DESTINATION_DIR}/src}
S2I_TARGET_CONFIGURATION_DIR	**$S2I_SOURCE_DIR $S2I_SOURCE_CONFIGURATION_DIR** にあるファイルがコピーされる絶対パス。	-	/opt/eap/standalone/configuration
S2I_TARGET_DATA_DIR	**$S2I_SOURCE_DIR/$S2I_SOURCE_DATA_DIR** にあるファイルがコピーされる絶対パス。	-	/opt/eap/standalone/data
S2I_TARGET_DEPLOYMENTS_DIR	**$S2I_SOURCE_DIR/$S2I_SOURCE_DEPLOYMENTS_DIR** にあるファイルがコピーされる絶対パス。また、これはビルド出力がコピーされるディレクトリーです。	-	/deployments
http_proxy	HTT プロキシーの場所。これは **HTTP_PROXY** よりも優先され、Maven ビルドと Java ランタイムの両方に使用されます。	-	http://127.0.0.1:8080
https_proxy	HTTPS プロキシーの場所。これは、**HTTPS_PROXY**、**http_proxy**、および **HTTP_PROXY** よりも優先され、Maven ビルドと Java ランタイムの両方に使用されます。	-	myuser:mypass@127.0.0.1:8080

変数名	説明	デフォルト値	値の例
no_proxy	直接アクセスできるホスト、IP アドレス、またはドメインのコンマ区切りリスト。これは **NO_PROXY** よりも優先され、Maven ビルドと Java ランタイムの両方に使用されます。	-	*.example.com
prefix_MAVEN_MIRROR_ID	指定されたミラーに使用する ID。省略する場合には、一意の ID が生成されます。	-	internal-mirror
prefix_MAVEN_MIRROR_OF	このエントリーでミラリングされるリポジトリー ID。	external:*	-
prefix_MAVEN_MIRROR_URL	ミラーの URL。	-	http://10.0.0.1:8080/repository/internal
prefix_MAVEN_REPO_DIRECTORY_PERMISSIONS	Maven リポジトリーディレクトリーのパーミッション。	-	775
prefix_MAVEN_REPO_FILE_PERMISSIONS	Maven リポジトリーファイルのパーミッション。	-	664
prefix_MAVEN_REPO_HOST	Maven リポジトリーホスト (完全に定義された URL を使用していない場合、は、サービスにフォールバックする)	-	repo.example.com
prefix_MAVEN_REPO_ID	Maven リポジトリー ID。	-	my-repo-id
prefix_MAVEN_REPO_LAYOUT	Maven リポジトリーのレイアウト。	-	default
prefix_MAVEN_REPO_NAME	Maven リポジトリー名。	-	my-repo-name
prefix_MAVEN_REPO_PASSPHRASE	Maven リポジトリーのパスフレーズ。	-	maven1!

変数名	説明	デフォルト値	値の例
prefix_MAVEN_REPO_PASSWORD	Maven リポジトリーのパスワード。	-	maven1!
prefix_MAVEN_REPO_PATH	Maven リポジトリーのパス (完全に定義された URL を使用しない場合は、サービスにフォールバックする)	-	/maven2/
prefix_MAVEN_REPO_PORT	Maven リポジトリーのポート (完全に定義された URL を使用しない場合は、サービスにフォールバックする)	-	8080
prefix_MAVEN_REPO_PRIVATE_KEY	Maven リポジトリーのプライベートキー。	-	${user.home}/.ssh/id_dsa
prefix_MAVEN_REPO_PROTOCOL	Maven リポジトリーのプロトコル (完全に定義された URL を使用しない場合は、サービスにフォールバックする)	-	http
prefix_MAVEN_REPO_RELEASES_CHECKSUM_POLICY	Maven リポジトリーは、チェックサムポリシーをリリースします。	-	warn
prefix_MAVEN_REPO_RELEASES_ENABLED	Maven リポジトリーのリリースが有効。	-	true
prefix_MAVEN_REPO_RELEASES_UPDATE_POLICY	Maven リポジトリーリリース更新ポリシー。	-	always
prefix_MAVEN_REPO_SERVICE	prefix_MAVEN_REPO_URL が指定されていない場合を検索する Maven リポジトリーサービス。	-	buscentr-myapp
prefix_MAVEN_REPO_SNAPSHOTS_CHECKSUM_POLICY	Maven リポジトリースナップショットチェックサムポリシー。	-	warn
prefix_MAVEN_REPO_SNAPSHOTS_ENABLED	Maven リポジトリーのスナップショットが有効。	-	true

変数名	説明	デフォルト値	値の例
prefix_MAVEN_REPO_SNAPSHOTS_UPDATE_POLICY	Maven リポジトリースナップショット更新ポリシー。	-	always
prefix_MAVEN_REPO_URL	Maven リポジトリー URL(完全に定義)	-	http://repo.example.com:8080/maven2/
prefix_MAVEN_REPO_USERNAME	Maven リポジトリーのユーザー名。	-	mavenUser

5.3.1. デフォルト値の設定環境変数

以下の設定環境変数には、上書きできるデフォルト値が指定されています。

表5.4 デフォルト値の設定環境変数

変数名	説明	デフォルト値
AB_JOLOKIA_AUTH_OPENSHIFT	OpenShift TLS 通信のクライアント認証を切り替えます。このパラメーターの値は、提供されるクライアントの証明書に含まれる必要がある相対識別名になります。このパラメーターを有効にすると、Jolokia が自動的に HTTPS 通信モードに切り替わります。デフォルトの CA 証明書は **/var/run/secrets/kubernetes.io/serviceaccount/ca.crt** に設定されます。	true
AB_JOLOKIA_HTTPS	HTTPS を使用したセキュアな通信を有効にします。デフォルトでは、**AB_JOLOKIA_OPTS** で **serverCert** 設定の指定がないと、自己署名サーバー証明書が生成されます。	true
AB_JOLOKIA_PASSWORD_RANDOM	**AB_JOLOKIA_PASSWORD** が無作為に生成されるべきかどうかを決定します。無作為にパスワードを生成するには **true** に設定します。生成された値は **/opt/jolokia/etc/jolokia.pw** に書き込まれます。	true
AB_PROMETHEUS_JMX_EXPORTER_CONFIG	Prometheus JMX エクスポーターに使用する設定へのパス。	/opt/jboss/container/prometheus/etc/jmx-exporter-config.yaml

変数名	説明	デフォルト値
S2I_SOURCE_DEPLOYMENTS_FILTER	デプロイメントのコピー時に適用されるスペースで区切られたフィルターのリスト。デフォルトは * に設定されます。	*

5.4. 公開されたポート

以下の表は、公開されたポートのリストです。

ポート番号	説明
8080	HTTP
8443	HTTPS
8778	Jolokia の監視

5.5. MAVEN の設定

Maven 引数のあるデフォルトの Maven 設定
MAVEN_ARGS 環境変数のデフォルト値には、**-Dcom.redhat.xpaas.repo.redhatga** プロパティーが含まれます。このプロパティーは、デフォルトの **jboss-settings.xml** ファイル内の **https://maven.repository.redhat.com/ga/** リポジトリーでプロファイルを有効にします。これは、S2I for OpenShift イメージにあります。

MAVEN_ARGS 環境変数にカスタム値を指定する場合は、カスタムの **source_dir/configuration/settings.xml** ファイルが指定されていない場合、イメージのデフォルトの **jboss-settings.xml** が使用されます。

デフォルトの jboss-settings.xml 内で使用する Maven リポジトリーを指定するには、2 つのプロパティーがあります。

- https://maven.repository.redhat.com/ga/ リポジトリーを使用する **-Dcom.redhat.xpaas.repo.redhatga** プロパティー。
- https://repository.jboss.org/nexus/content/groups/public/ リポジトリーを使用する **-Dcom.redhat.xpaas.repo.jbossorg** プロパティー。

カスタム Maven 設定の指定
Maven 引数と共にカスタムの **settings.xml** ファイルを指定するには、**source_dir/configuration** ディレクトリーを作成し、設定の '.xml' ファイルを内部に配置します。

サンプルパスは **source_dir/configuration/settings.xml** のようになります。

改訂日時: 2024-05-10