Red Hat JBoss Web Server 5.4

Red Hat JBoss Web Server for OpenShift

Red Hat JBoss Web Server for OpenShift のインストールおよび使用

Last Updated: 2023-05-19

Red Hat JBoss Web Server 5.4 Red Hat JBoss Web Server for OpenShift

Red Hat JBoss Web Server for OpenShift のインストールおよび使用

法律上の通知

概要

Red Hat JBoss Web Server for OpenShift の使用ガイド

目次

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 をご覧ください。

第1章 はじめに

1.1. RED HAT JBOSS WEB SERVER FOR OPENSHIFT の概要

Red Hat JBoss Web Server (JWS) 5.4 の Apache Tomcat 9 コンポーネントは、OpenShift 用に設計されたコンテナー化イメージとして利用できます。開発者は、このイメージを使用して、ハイブリッドクラウド環境にまたがるデプロイメントのために Java Web アプリケーションをビルド、スケーリング、テストできます。

第2章 作業開始前の準備

2.1. RED HAT JBOSS WEB SERVER と JWS FOR OPENSHIFT の相違点

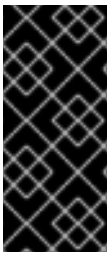
OpenShift イメージの JWS と JWS の定期的なリリースの違いは次のとおりです。

- OpenShift イメージの JWS 内部の **JWS_HOME** の場所は、**/opt/jws-5.4/** になります。
- すべての負荷分散は、Apache HTTP Server の mod_cluster または mod_jk コネクターではなく、OpenShift ルーターによって処理されます。

JWS for OpenShift イメージに固有の JWS 機能のドキュメントは、Red Hat JBoss Web Server のドキュメント を参照してください。

2.2. バージョンの互換性とサポート

OpenShift イメージバージョンの互換性についての詳細は、OpenShift Container Platform Tested 3.X Integrations ページ と OpenShift Container Platform Tested 4.X Integrations ページ の xPaaS テーブルを参照してください。



重要

新しいアプリケーションのデプロイには、JWS for OpenShift イメージおよびアプリケーションテンプレートの 5.4 バージョンを使用する必要があります。

JWS for OpenShift イメージおよびアプリケーションテンプレートの 5.3 バージョンは非推奨となり、更新を受信しなくなりました。

2.3. JBOSS WEB SERVER でサポートされるアーキテクチャー

JBoss Web サーバーは以下のアーキテクチャーをサポートします。

- x86_64 (AMD64)
- OpenShift 環境の IBM Z (s390x)
- OpenShift 環境の IBM Power (ppc64le)

アーキテクチャーごとに異なるイメージがサポートされます。本ガイドのコード例は、x86_64 アーキテクチャーのコマンドを示しています。他のアーキテクチャーを使用している場合は、コマンドに適切なイメージ名を指定します。イメージの詳細は、Red Hat Container Catalog を参照してください。

2.4. RED HAT コンテナーイメージのヘルスチェック

OpenShift で利用可能なすべてのコンテナーイメージには、ヘルス評価 (Health rating) が関連付けられています。Red Hat JBoss Web Server のヘルス評価を確認するには、コンテナーイメージのカタログ に移動し、**JBoss Web Server** を検索し、5.4 バージョンを選択します。

OpenShift コンテナーの Liveliness および Readiness についてテストする方法の詳細は、以下のドキュメント を参照してください。

第3章 はじめに

3.1. 初期設定

本書の手順は OpenShift Primer に従います。OpenShift Primer に記述されているサポートされる OpenShift の設定または非実稼働環境の OpenShift インスタンスを想定しています。

注記

JWS for OpenShift アプリケーションテンプレートは、Tomcat 9 に対して配布されます。

3.2. RED HAT コンテナーレジストリーへの認証の設定

Red Hat JBoss Web Server イメージをインポートおよび使用する前に、最初に Red Hat コンテナーレジストリーへの認証を設定する必要があります。

Red Hat は、レジストリーサービスアカウントを使用して認証トークンを作成し、Red Hat コンテナーレジストリーへのアクセスを設定することを推奨します。こうすると、お持ちの Red Hat アカウントのユーザー名やパスワードを OpenShift 設定に使用または保存する必要がありません。

1. Red Hat カスタマーポータルの手順にしたがって、レジストリーサービスアカウントを使用して認証トークンを作成します。
2. トークンの OpenShift シークレットが含まれる YAML ファイルをダウンロードします。YAML ファイルは、トークンの **Token Information** ページの **OpenShift Secret** タブからダウンロードできます。
3. ダウンロードした YAML ファイルを使用して、OpenShift プロジェクトの認証トークンシークレットを作成します。

```
oc create -f 1234567_myserviceaccount-secret.yaml
```

4. 以下のコマンドを使用して、OpenShift プロジェクトのシークレットを設定します。シークレット名は前のステップで作成したシークレットの名前に置き換えてください。

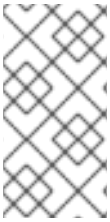
```
oc secrets link default 1234567-myserviceaccount-pull-secret --for=pull
oc secrets link builder 1234567-myserviceaccount-pull-secret --for=pull
```

セキュリティー保護されたレジストリーへのアクセス設定方法 については、OpenShift ドキュメントを参照してください。

Red Hat コンテナーレジストリーへの認証の設定 に関する詳細は、Red Hat カスタマーポータルを参照してください。

3.3. 最新の RED HAT JBOSS WEB SERVER イメージストリームおよびテンプレートのインポート

JDK の最新の Red Hat JBoss Web Server for OpenShift イメージストリームおよびテンプレートを OpenShift プロジェクトの名前空間にインポートする必要があります。



注記

カスタマーポータルの認証情報を使用して Red Hat Container Registry にログインし、Red Hat JBoss Web Server イメージストリームおよびテンプレートをインポートしてイメージストリームを更新します。詳細は、Red Hat コンテナーレジストリーの認証 を参照してください。

JDK 8 (RHEL7) の import コマンド

```
for resource in \
jws54-openjdk8-tomcat9-rhel7-basic-s2i.json \
jws54-openjdk8-tomcat9-rhel7-https-s2i.json \
jws54-openjdk8-tomcat9-rhel7-image-stream.json \
jws54-openjdk8-tomcat9-rhel7-mongodb-persistent-s2i.json \
jws54-openjdk8-tomcat9-rhel7-mongodb-s2i.json \
jws54-openjdk8-tomcat9-rhel7-mysql-persistent-s2i.json \
jws54-openjdk8-tomcat9-rhel7-mysql-s2i.json \
jws54-openjdk8-tomcat9-rhel7-postgresql-persistent-s2i.json \
jws54-openjdk8-tomcat9-rhel7-postgresql-s2i.json
do
oc replace -n openshift --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-webserver-5-openshift-image/jws54el7-v1.0/templates/${resource}
done
```

このコマンドは以下のイメージストリームおよびテンプレートをインポートします。

- RHEL7 JDK 8 イメージストリーム: jboss-webserver54-openjdk8-tomcat9-openshift-rhel7
- コマンドで指定したすべてのテンプレート

JDK 8 (RHEL8) の import コマンド

```
for resource in \
jws54-openjdk8-tomcat9-ubi8-basic-s2i.json \
jws54-openjdk8-tomcat9-ubi8-https-s2i.json \
jws54-openjdk8-tomcat9-ubi8-image-stream.json
do
oc replace -n openshift --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-webserver-5-openshift-image/jws54el8-v1.0/templates/${resource}
done
```

このコマンドは以下のイメージストリームおよびテンプレートをインポートします。

- RHEL8 JDK 8 イメージストリーム: jboss-webserver54-openjdk8-tomcat9-openshift-ubi8
- コマンドで指定したすべてのテンプレート

JDK 11 (RHEL7) の import コマンド

```
for resource in \
jws54-openjdk11-tomcat9-rhel7-basic-s2i.json \
jws54-openjdk11-tomcat9-rhel7-https-s2i.json \
jws54-openjdk11-tomcat9-rhel7-image-stream.json \
```

```
jws54-openjdk11-tomcat9-rhel7-mongodb-persistent-s2i.json \
jws54-openjdk11-tomcat9-rhel7-mongodb-s2i.json \
jws54-openjdk11-tomcat9-rhel7-mysql-persistent-s2i.json \
jws54-openjdk11-tomcat9-rhel7-mysql-s2i.json \
jws54-openjdk11-tomcat9-rhel7-postgresql-persistent-s2i.json \
jws54-openjdk11-tomcat9-rhel7-postgresql-s2i.json
do
oc replace -n openshift --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-webserver-5-openshift-image/jws54el7-v1.0/templates/${resource}
done
```

このコマンドは以下のイメージストリームおよびテンプレートをインポートします。

- RHEL7 JDK 11 イメージストリーム: jboss-webserver54-openjdk11-tomcat9-openshift-rhel7
- コマンドで指定したすべてのテンプレート

JDK 11 (RHEL8) の import コマンド

```
for resource in \
jws54-openjdk8-tomcat9-ubi8-basic-s2i.json \
jws54-openjdk8-tomcat9-ubi8-https-s2i.json \
jws54-openjdk8-tomcat9-ubi8-image-stream.json
do
oc replace -n openshift --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-webserver-5-openshift-image/jws54el8-v1.0/templates/${resource}
done
```

このコマンドは以下のイメージストリームおよびテンプレートをインポートします。

- RHEL8 JDK 11 イメージストリーム: jboss-webserver54-openjdk11-tomcat9-openshift-ubi8
- コマンドで指定したすべてのテンプレート

3.3.1. コマンドの更新

- コア JWS 5.4 tomcat 9 OpenJDK8 RHEL7 OpenShift イメージを更新するには、以下を実行する必要があります。

```
$  oc -n openshift import-image \
   jboss-webserver54-openjdk8-tomcat9-openshift-rhel7:1.0
```

- コア JWS 5.4 tomcat 9 OpenJDK8 RHEL8 OpenShift を更新するには、以下を実行する必要があります。

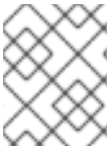
```
$ oc -n openshift import-image \
  jboss-webserver54-openjdk8-tomcat9-openshift-ubi8:1.0
```

- コア JWS 5.4 tomcat 9 OpenJDK11 RHEL7 OpenShift イメージを更新するには、以下を実行する必要があります。

```
$ oc -n openshift import-image \
  jboss-webserver54-openjdk11-tomcat9-openshift-rhel7:1.0
```

- コア JWS 5.4 tomcat 9 OpenJDK11 RHEL8 OpenShift イメージを更新するには、以下を実行する必要があります。

```
$ oc -n openshift import-image \
  jboss-webserver54-openjdk11-tomcat9-openshift-ubi8:1.0
```



注記

インポートしている各イメージの末尾の **1.0** タグは、イメージストリーム に設定されるストリームバージョンを参照します。

3.4. OPENSHIFT SOURCE-TO-IMAGE (S2I) プロセスでの JWS の使用

JWS for OpenShift イメージを実行および設定するには、アプリケーションテンプレートパラメーターおよび環境変数を使用して OpenShift S2I プロセスを使用します。

JWS for OpenShift イメージの S2I プロセスは以下のようになります。

- **configuration/** Maven **settings.xml** ファイルがある場合は、新しいイメージの **$HOME/.m2/** に移動します。
  Maven および Maven の **settings.xml** ファイルの詳細は、Apache Maven Project の Web サイト を参照してください。

- ソースリポジトリーに **pom.xml** ファイルがある場合、**$MAVEN_ARGS** 環境変数の内容を使用して Maven ビルドがトリガーされます。
  デフォルトでは、**package** のゴールは **openshift** プロファイルで使用されます。これには、テストをスキップする引数 (**-DskipTests**) および Red Hat GA リポジトリーを有効にするための引数 (**-Dcom.redhat.xpaas.repo.redhatga**) が含まれます。

- 成功した Maven ビルドの結果は **/opt/webserver/webapps/** にコピーされます。これには、**$ARTIFACT_DIR** 環境変数によって指定されたソースディレクトリーからの WAR ファイルがすべて含まれます。**$ARTIFACT_DIR** のデフォルト値は **target/** ディレクトリーです。**MAVEN_ARGS_APPEND** 環境変数を使用して Maven 引数を変更します。

- **deployments/** ソースディレクトリーのすべての WAR ファイルが **/opt/webserver/webapps/** にコピーされます。

- **configuration/** ソースディレクトリーのすべてのファイルは、**/opt/webserver/conf/** にコピーされます (Maven の **settings.xml** ファイルを除く)。

- **lib/** ソースディレクトリーのすべてのファイルは **/opt/webserver/lib/** にコピーされます。

注記

カスタムの Tomcat 設定ファイルを使用する場合は、通常の Tomcat インストールと同じファイル名を使用する必要があります。例: **context.xml** および **server.xml**。

S2I プロセスを設定してカスタム Maven アーティファクトリーポジトリーミラー を使用する方法は、アーティファクトリーポジトリーミラー の項を参照してください。

3.4.1. 既存の Maven バイナリーを使用した OpenShift アプリケーションの JWS の作成

既存のアプリケーションは **oc start-build** コマンドを使用して OpenShift にデプロイされます。

前提条件: OpenShift 用の JWS にデプロイする既存の **.war**、**.ear**、または **.jar** アプリケーション。

1. 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。
アプリケーションが必要とするコンテンツがバイナリーに含まれていないコンテンツを含むソースディレクトリーを作成します。必要に応じて、Using the JWS for OpenShift Source-to-Image(S2I) プロセス を参照してください。次に、**deployments/** サブディレクトリーを作成します。

```
$ mkdir -p <build_dir>/deployments
```

2. バイナリー (**.war**,**.ear**,**.jar**) を **deployments/** にコピーします。

```
$ cp /path/to/binary/<filenames_with_extensions> <build_dir>/deployments/
```

注記

ソースディレクトリーの **deployments/** サブディレクトリーにあるアプリケーションアーカイブは、OpenShift 上にビルドされているイメージの **$JWS_HOME/tomcat/webapps/** ディレクトリーにコピーされます。アプリケーションをデプロイするには、web アプリケーションデータが含まれるディレクトリー階層が正しく設定される必要があります (「OpenShift Source-to-Image (S2I) プロセスでの JWS の使用」 を参照)。

3. OpenShift インスタンスにログインします。

```
$ oc login <url>
```

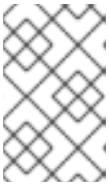
4. 必要に応じて新規プロジェクトを作成します。

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

5. **oc get is -n openshift** でアプリケーションに使用する OpenShift イメージストリームの JWS を特定します。

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

jboss-webserver50-tomcat9-openshift
```



注記

オプション **-n openshift** は使用するプロジェクトを指定します。**oc get is -n openshift** は **openshift** プロジェクトからイメージストリームリソース (**is**) を取得 (**get**) します。

6. イメージストリームおよびアプリケーション名を指定して、新規ビルド設定を作成します。

```
$ oc new-build --binary=true \
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel7:latest \
--name=<my-jws-on-openshift-app>
```

7. OpenShift イメージビルドのバイナリー入力用に上記で作成されたソースディレクトリーを使用するように OpenShift に指示します。

```
$ oc start-build <my-jws-on-openshift-app> --from-dir=./<build_dir> --follow
```

8. イメージに基づいて新しい OpenShift アプリケーションを作成します。

```
$ oc new-app <my-jws-on-openshift-app>
```

9. サービスを公開して、アプリケーションがユーザーにアクセスできるようにします。

```
# to check the name of the service to expose
$ oc get svc -o name

service/<my-jws-on-openshift-app>

# to expose the service
$ oc expose svc/my-jws-on-openshift-app

route "my-jws-on-openshift-app" exposed
```

10. 公開されたルートのアドレスを取得します。

```
oc get routes --no-headers -o custom-columns='host:spec.host' my-jws-on-openshift-app
```

11. ブラウザーでアプリケーションにアクセスするには **http://<address_of_exposed_route>/<my-war-ear-jar-filename-without-extension>** を入力します。

3.4.2. 例: 既存の Maven バイナリーを使用した OpenShift アプリケーションの JWS の作成

以下の例では、「既存の Maven バイナリーを使用した OpenShift アプリケーションの JWS の作成」の手順を使用して tomcat-websocket-chat クイックスタートを使用します。

3.4.2.1. 要件:

A. **WAR アプリケーションアーカイブを取得したり、アプリケーションをローカルにビルドしたりします。**

- 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
```

- Configure the Red Hat JBoss Middleware Maven Repository
  - Red Hat JBoss Middleware Maven リポジトリーの追加情報
- アプリケーションをビルドします。

```
$ cd openshift-quickstarts/tomcat-websocket-chat/
```

```
$ mvn clean package
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Tomcat websocket example 1.2.0.Final
[INFO] ------------------------------------------------------------------------
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:28 min
[INFO] Finished at: 2018-01-16T15:59:16+10:00
[INFO] Final Memory: 19M/271M
[INFO] ------------------------------------------------------------------------
```

B. **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。**
バイナリービルドのソースディレクトリーを、ローカルファイルシステムと **deployments/** サブディレクトリーに作成します。WAR アーカイブを **deployments/** にコピーします。

```
[tomcat-websocket-chat]$ ls

pom.xml  README.md  src/  target/
```

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

```
$ cp target/websocket-chat.war ocp/deployments/
```

3.4.2.2. OpenShift でサンプルアプリケーションを設定する方法

1. OpenShift インスタンスにログインします。

```
$ oc login <url>
```

2. 必要に応じて新規プロジェクトを作成します。

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

3. **oc get is -n openshift** でアプリケーションに使用する OpenShift イメージストリームの JWS を特定します。

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

jboss-webserver50-tomcat9-openshift
```

4. イメージストリームおよびアプリケーション名を指定して、新規ビルド設定を作成します。

```
$ oc new-build --binary=true \
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel7:latest\
 --name=jws-wsch-app

--> Found image 8c3b85b (4 weeks old) in image stream "openshift/jboss-
webserver<version>-tomcat9-openshift" under tag "latest" for "jboss-webserver<version>"
```

```
    JBoss Web Server 5.0
    --------------------
    Platform for building and running web applications on JBoss Web Server 5.0 - Tomcat v9

    Tags: builder, java, tomcat9

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

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

5. バイナリービルドを開始します。OpenShift イメージビルドのバイナリー入力にソースディレクトリーを使用するように OpenShift に指示します。

```
$ *oc start-build jws-wsch-app --from-dir=./ocp --follow*

Uploading directory "ocp" as binary input for the build ...
build "jws-wsch-app-1" started
Receiving source from STDIN as archive ...

Copying all deployments war artifacts from /home/jboss/source/deployments directory into
`/opt/jws-5.4/tomcat/webapps` for later deployment...
'/home/jboss/source/deployments/websocket-chat.war' -> '/opt/jws-
5.4/tomcat/webapps/websocket-chat.war'

Pushing image 172.30.202.111:5000/jws-bin-demo/jws-wsch-app:latest ...
Pushed 0/7 layers, 7% complete
Pushed 1/7 layers, 14% complete
Pushed 2/7 layers, 29% complete
Pushed 3/7 layers, 49% complete
Pushed 4/7 layers, 62% complete
Pushed 5/7 layers, 92% complete
Pushed 6/7 layers, 100% complete
Pushed 7/7 layers, 100% complete
Push successful
```

6. イメージに基づいて新しい OpenShift アプリケーションを作成します。

```
$ oc new-app jws-wsch-app

--> Found image e5f3a6b (About a minute old) in image stream "jws-bin-demo/jws-wsch-app"
under tag "latest" for "jws-wsch-app"

    JBoss Web Server 5.0
    --------------------
    Platform for building and running web applications on JBoss Web Server 5.0 - Tomcat v9

    Tags: builder, java, tomcat9

    * This image will be deployed in deployment config "jws-wsch-app"
    * Ports 8080/tcp, 8443/tcp, 8778/tcp will be load balanced by service "jws-wsch-app"
```

```
     * Other containers can access this service through the hostname "jws-wsch-app"

--> Creating resources ...
    deploymentconfig "jws-wsch-app" created
    service "jws-wsch-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/jws-wsch-app'
    Run 'oc status' to view your app.
```

7. サービスを公開して、アプリケーションがユーザーにアクセスできるようにします。

```
# to check the name of the service to expose
$ oc get svc -o name

service/jws-wsch-app

# to expose the service
$ oc expose svc/jws-wsch-app

route "jws-wsch-app" exposed
```

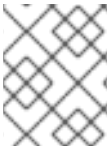
8. 公開されたルートのアドレスを取得します。

```
oc get routes --no-headers -o custom-columns='host:spec.host' jws-wsch-app
```

9. ブラウザーでアプリケーションにアクセスします:
http://<address_of_exposed_route>/websocket-chat

3.4.3. ソースコードから JWS for OpenShift アプリケーションを作成します。

ソースコードから新しい OpenShift アプリケーションを作成する詳細な手順は、OpenShift.com - ソースコードからのアプリケーションの作成 を参照してください。



注記

次に進む前に、アプリケーションのデータが正しく構造化されていることを確認します (「OpenShift Source-to-Image (S2I) プロセスでの JWS の使用」 を参照)。

1. OpenShift インスタンスにログインします。

```
$ oc login <url>
```

2. 必要に応じて新規プロジェクトを作成します。

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

3. **oc get is -n openshift** でアプリケーションに使用する OpenShift イメージストリームの JWS を特定します。

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

jboss-webserver50-tomcat9-openshift
```

4. Red Hat JBoss Web Server for OpenShift イメージを使用して、ソースコードから新しい OpenShift アプリケーションを作成し、**--image-stream** オプションを使用します。

```
$ oc new-app \
 <source_code_location>\
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel7\
 --name=<openshift_application_name>
```

例を以下に示します。

```
$ oc new-app \
 https://github.com/jboss-openshift/openshift-quickstarts.git#master \
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel7\
 --context-dir='tomcat-websocket-chat' \
 --name=jws-wsch-app
```

ソースコードがイメージに追加され、ソースコードがコンパイルされます。ビルド設定とサービスも作成されます。

5. アプリケーションを公開するには、以下を実行します。

```
# to check the name of the service to expose
$ oc get svc -o name

service/<openshift_application_name>

# to expose the service
$ oc expose svc/<openshift_application_name>

route "<openshift_application_name>" exposed
```

6. 公開されたルートのアドレスを取得するには、以下を実行します。

```
oc get routes --no-headers -o custom-columns='host:spec.host'
<openshift_application_name>
```

7. ブラウザーでアプリケーションにアクセスします: http://<address_of_exposed_route>/<java_application_name>

3.5. TOMCAT/LIB/ ディレクトリーに追加の JAR ファイルの追加

docker を使用して、追加の jar ファイルを **tomcat/lib/** ディレクトリーに追加できます。

tomcat/lib/ への jar ファイルの

1. docker で開始されるイメージを取得します。

```
docker run --network host -i -t -p 8080:8080 ImageURL
```

2. **CONTAINER ID** を検索します。

```
docker ps | grep <ImageName>
```

3. ライブラリーを **tomcat/lib/** ディレクトリーにコピーします。

```
docker cp <yourLibrary> <CONTAINER ID>:/opt/jws-5.4/tomcat/lib/
```

4. 新規イメージへの変更のコミット

```
docker commit <CONTAINER ID> <NEW IMAGE NAME>
```

5. 新規イメージタグの作成

```
docker tag <NEW IMAGE NAME>:latest <NEW IMAGE REGISTRY URL>:<TAG>
```

6. イメージをレジストリーにプッシュします。

```
docker push <NEW IMAGE REGISTRY URL>
```

第4章 JWS OPERATOR

4.1. JBOSS WEB SERVER OPERATOR

4.1.1. OpenShift Operators

Operator Framework は Operator という Kubernetes ネイティブアプリケーションを効果的かつ自動化された拡張性のある方法で管理するためのツールキットです。Operator により、Kubernetes の上部に複雑なステートフルアプリケーションを管理することが容易になります。すべての Operator は、Operator SDK、Operator Lifecycle Manager、および OperatorHub.io の 3 つの主要なコンポーネントをベースとしています。これらのツールを使用すると、独自のオペレーターを開発し、Kubernetes クラスターで使用する Operator を管理し、コミュニティーが作成する Operator を検出したり、共有したりできます。

Red Hat JBoss Web Server プロジェクトは、OpenShift イメージを管理するための Operator を提供します。このセクションでは、JWS 用に OpenShift Operator をビルドし、テストし、パッケージ化する方法を説明します。

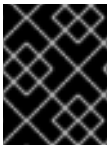
クラスターの設定に関する詳しい手順は、Openshift ドキュメント のサブセクション Install を参照してください。

さらに、JWS Operator は JWS-on-OpenShift 設定とは異なる環境変数を使用します。これらの パラメーターの完全なリストは、こちらを参照してください。

重要

現時点では、Use Session Clustering 機能は完全にサポートされていません。

本ガイドでは、JWS Operator のインストール、デプロイメント、および削除について詳しく説明します。より高速で詳細な情報は、クイックスタートガイドを参照してください。



重要

現在、JWS 5.4 イメージのみをサポートします。5.4 よりも古いイメージには対応して**いません**。

4.1.2. JWS Operator のインストール

このセクションでは、OpenShift Container Platform での JWS Operator のインストールについて説明します。

4.1.2.1. 前提条件

- **cluster admin** のパーミッションを持つアカウントを使用した OpenShift Container Platform クラスター (Web コンソールのみ)
- Operator のインストールパーミッションを持つアカウントを使用した OpenShift Container Platform クラスター
- ローカルシステムにインストールされている **OC** ツール (CLI のみ)

4.1.2.2. JWS Operator - Web コンソールのインストール

1. 左側のメニューにある Operators タブに移動します。
2. これにより、OpenShift OperatorHub を開きます。ここから JWS を検索し、JWS Operator を選択します。
3. 新規メニューが表示されます。必要な容量レベルを選択してから Install をクリックして Operator をインストールします。
4. これで、Operator のインストールを設定できるようになりました。以下の 3 つのオプションを指定します。
   - **Installation Mode:** インストールするクラスターに特定の名前空間を指定します。これを指定しない場合は、デフォルトでオペレーターをクラスターのすべての名前空間にインストールします。
   - **Update Channel:** JWS オペレーターは現在、1 つのチャネルからしか利用できません。
   - **承認ストラテジー**: 自動の更新または手動の更新を選択できます。インストールされた Operator について自動更新を選択する場合は、Operator の新規バージョンが利用可能になると、Operator Lifecycle Manager (OLM) は人の介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。手動更新を選択する場合、Operator の新規バージョンが利用可能になると、OLM は更新要求を作成します。クラスター管理者は、Operator を新規バージョンに更新できるように更新要求を手動で承認する必要があります。
5. 下部の Install をクリックします。**Manual Approval Strategy** を選択した場合は、インストールが完了する前にインストールプランを承認する必要があります。JWS Operator は、Operator タブの Installed Operators セクションに表示されるようになります。

4.1.2.3. JWS Operator のインストール - コマンドラインインターフェイス

1. JWS Operator を検査し、以下のコマンドを使用して、サポートされる installMode および利用可能なチャネルを確認します。

```
$   oc get packagemanifests -n openshift-marketplace | grep jws
    jws-operator    Red Hat Operators   16h
```

```
$   oc describe packagemanifests jws-operator -n openshift-marketplace | grep "Catalog
Source"
    Catalog Source:     redhat-operators
```

2. OperatorGroup は、OperatorGroup と同じ名前空間内のすべての Operator に必要な RBAC アクセスを生成するターゲット名前空間を選択する OLM リソースです。
   Operator をサブスクライブする名前空間には、Operator の InstallMode に一致する OperatorGroup が必要になります (AllNamespaces または SingleNamespace モードのいずれか)。インストールする Operator が AllNamespaces を使用する場合は、openshift-operators 名前空間には適切な OperatorGroup がすでに配置されます。

   ただし、Operator が SingleNamespace モードを使用する場合、1 つの OperatorGroup をその名前空間で作成する必要があります。OperatorGroup の実際の一覧を確認するには、以下のコマンドを使用します。

```
$ oc get operatorgroups -n <project_name>
```

   OperatorGroup リストの出力例:

```
NAME      AGE
mygroup   17h
```

注記

この手順の Web コンソールバージョンは、SingleNamespace モードを選択する際に、OperatorGroup および Subscription オブジェクトの作成を背後で自動的に処理します。

- 以下のように OperatorGroup オブジェクト YAML ファイルを作成します。**OperatorGroupExample.yaml**:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <project_name>
spec:
  targetNamespaces:
  - <project_name>
```

<project_name> はオペレーターをインストールするプロジェクトの名前空間 (oc project -q) です。<operatorgroup_name> は OperatorGroup の名前です。

- 以下のコマンドを使用して OperatorGroup オブジェクトを作成します。

```
$   oc apply -f OperatorGroupExample.yaml
```

3. サブスクリプションオブジェクト YAML ファイルを作成します (例: **jws-operator-sub.yaml**)。**Subscription** オブジェクト YAML ファイルを以下のように設定します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
   name: jws-operator
   namespace: <project_name>
spec:
   channel: alpha
   name: jws-operator
   source: redhat-operators
   sourceNamespace: openshift-marketplace
```

<project_name> は、オペレーター (oc project -q) をインストールするプロジェクトの名前空間であり、すべての名前空間で openshift-operators を使用するようになります。

source は Catalog Source です。これは、このセクションの手順 1 で実行した **$ oc describe packagemanifests jws-operator -n openshift-marketplace | grep "Catalog Source:"** コマンドの値です。この値は **redhat-operators** である必要があります。

4. 以下のコマンドを使用して、YAML ファイルから **Subscription** オブジェクトを作成します。

```
$   oc apply -f jws-operator-sub.yaml
```

インストールが正常に行われたことを確認するには、以下のコマンドを実行します。

```
$   oc get csv -n <project_name>
```

名前	DISPLAY	バージョン	REPLACE PHASE
jws-operator.V1.0.0	JBoss Web Server Operator	1.0.0	成功

4.1.3. 既存の JWS イメージのデプロイ

1. 以下のコマンドを使用してオペレーターがインストールされていることを確認します。

```
$   oc get deployment.apps/jws-operator
    NAME        READY  UP-TO-DATE   AVAILABLE   AGE
    jws-operator   1/1    1         1         15h
```

または、より詳細な出力が必要な場合は、以下を行います。

```
$   oc describe deployment.apps/jws-operator
```

2. イメージを準備し、これを任意の場所にプッシュします。この例では **quay.io/<USERNAME>/tomcat-demo:latest** にプッシュされます。

3. **Custom Resource** WebServer .yaml ファイルを作成します。この例では、**webservers_cr.yaml** というファイルが使用されます。ファイルは、以下の形式に従う必要があります。

```
apiVersion: web.servers.org/v1alpha1
kind: WebServer
metadata:
    name: example-image-webserver
spec:
    # Add fields here
    applicationName: jws-app
    replicas: 2
webImage:
   applicationImage: quay.io/<USERNAME>/tomcat-demo:latest
```

4. 以下のコマンドを使用して、作成したディレクトリーから webapp をデプロイします。

```
$   oc apply -f webservers_cr.yaml
    webserver/example-image-webserver created
```

注記

オペレーターによってルートが自動的に作成されます。以下のコマンドを使用して、ルートを確認できます。

```
$ oc get routes
```

ルートの詳細は、OpenShift ドキュメント を参照してください。

5. ステップ 4 で作成した **webserver** を削除する場合は、以下を行います。

```
$ oc delete webserver example-image-webserver
```

または

```
$ oc delete -f webservers_cr.yaml
```

4.1.4. クラスターからの Operator の削除

4.1.4.1. 前提条件

- 管理者権限を持つ OpenShift Container Platform クラスター (または これらの手順に従って この要件を回避できます)
- ローカルシステムにインストールされている **OC** ツール (CLI のみ)

4.1.4.2. クラスターからのオペレーターの削除 - Web コンソール

1. 左側のメニューで、'Operators' → 'Installed Operators' をクリックします。
2. Operator Details メニューから 'Actions' メニューを選択してから 'Uninstall Operator' をクリックします。
3. このオプションを選択すると、Operator、Operator デプロイメント、および Pod が削除されます。**ただし**、Operator を削除しても、CRD または CR を含め、カスタムリソース定義またはカスタムリソースはいずれも削除**されません**。Operator がクラスターにアプリケーションをデプロイしているか、またはクラスター外のリソースを設定している場合、それらは引き続き実行され、手動でクリーンアップする必要があります。

4.1.4.3. クラスターからの Operator の削除 - コマンドラインインターフェイス

1. 以下のコマンドを使用して、**currentCSV** フィールドでサブスクライブされた Operator の現行バージョンを確認します。

```
$ oc get subscription jws-operator -n <project_name> -o yaml | grep currentCSV
  f:currentCSV: {}
  currentCSV: jws-operator.v1.0.0
```

注記

上記のコマンドでは、**<project_name>** はオペレーターをインストールしたプロジェクトの名前空間を参照します。オペレーターがすべての名前空間にインストールされていた場合は、**<project_name>** の代わりに **openshift-operators** を使用します。

2. 以下のコマンドを使用してオペレーターのサブスクリプションを削除します。

```
$ oc delete subscription jws-operator -n <project_name>
```

注記

上記のコマンドでは、**<project_name>** はオペレーターをインストールしたプロジェクトの名前空間を参照します。オペレーターがすべての名前空間にインストールされていた場合は、**<project_name>** の代わりに **openshift-operators** を使用します。

3. 以下のコマンドを使用して、直前の手順の currentCSV 値を使用して、ターゲット名前空間のオペレーターの CSV を削除します。

```
$ oc delete clusterserviceversion <currentCSV> -n <project_name>
```

ここで、**<currentCSV>** は、手順 1 で取得した値になります。

```
$ oc delete clusterserviceversion jws-operator.v1.0.0
  clusterserviceversion.operators.coreos.com "jws-operator.v1.0.0" deleted
```

注記

上記のコマンドでは、**<project_name>** はオペレーターをインストールしたプロジェクトの名前空間を参照します。オペレーターがすべての名前空間にインストールされていた場合は、**<project_name>** の代わりに **openshift-operators** を使用します。

4.1.5. 参考情報

Operator についての詳細は、公式の OpenShift ドキュメントを参照してください。

Operator とは

および

OpenShift Container Platform

第5章 参照

5.1. SOURCE-TO-IMAGE (S2I)

Red Hat JBoss Web Server for OpenShift イメージには、S2I スクリプト と Maven が含まれています。

5.1.1. OpenShift の JWS での Maven アーティファクトリーポジトリーミラーの使用

Maven リポジトリーは、プロジェクト jar、ライブラリー jar、プラグイン、その他のプロジェクト固有のアーティファクトなどのビルドアーティファクトおよび依存関係を保持します。また、S2I ビルドの実行中にアーティファクトをダウンロードするための場所も定義します。Maven Central Repository の使用に加えて、ローカルのカスタムリポジトリー (mirror) もデプロイします。

ローカルミラーを使用する利点は次のとおりです。

- 地理的に近く、高速な同期ミラーを使用できる。
- リポジトリーコンテンツの制御が強化されます。
- パブリックサーバーおよびリポジトリーに依存する必要なく、異なるチーム (開発者、CI) 全体でアーティファクトを共有できる。
- ビルド時間が改善される。

Maven リポジトリーマネージャー はミラーへのローカルキャッシュとして機能できます。リポジトリーマネージャーがすでにデプロイされ、外部で **http://10.0.0.1:8080/repository/internal/** にアクセスできる場合、S2I ビルドはこのリポジトリーを使用できます。内部 Maven リポジトリーを使用するには、**MAVEN_MIRROR_URL** 環境変数をアプリケーションのビルド設定に追加します。

新規ビルド設定の場合は、**oc new-app** または **oc new-build** で **--build-env** オプションを指定します。

```
$ oc new-app \
 https://github.com/jboss-openshift/openshift-quickstarts.git#master \
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel7:latest\*
 --context-dir='tomcat-websocket-chat' \
 --build-env MAVEN_MIRROR_URL=http://10.0.0.1:8080/repository/internal/ \
 --name=jws-wsch-app
```

既存のビルド設定の場合

1. **MAVEN_MIRROR_URL** 変数を必要とするビルド設定を特定します。

```
$ oc get bc -o name

buildconfig/jws
```

2. **MAVEN_MIRROR_URL** 環境変数を **buildconfig/jws** に追加します。

```
$ oc env bc/jws MAVEN_MIRROR_URL="http://10.0.0.1:8080/repository/internal/"

buildconfig "jws" updated
```

3. ビルド設定が更新されたことを確認します。

```
$ oc env bc/jws --list

# buildconfigs jws
MAVEN_MIRROR_URL=http://10.0.0.1:8080/repository/internal/
```

4. **oc start-build** を使用したアプリケーションの新規ビルドのスケジュール

注記

アプリケーションのビルド中、Maven 依存関係はデフォルトのパブリックリポジトリーではなく、リポジトリーマネージャーからプルされることを確認できます。ビルドが完了すると、ミラーにはビルド時に取得され、使用されるすべての依存関係が含まれます。

5.1.2. Red Hat JBoss Web Server for OpenShift イメージに含まれるスクリプト

run

Catalina (Tomcat) の実行

assemble

Maven を使用してソースを構築し、パッケージ (**.war**) を作成し、**$JWS_HOME/tomcat/webapps** ディレクトリーに移動します。

5.1.3. OpenShift データソースの JWS

データソースには 3 つのタイプがあります。

1. **デフォルトの内部データソース:** これらは PostgreSQL、MySQL、および MongoDB です。これらのデータソースは、Red Hat レジストリー介して OpenShift ではデフォルトで利用でき、イメージストリームに追加の環境ファイルを設定する必要がありません。データベースを検出可能にして、データソースとして使用されるようにするには、**DB_SERVICE_PREFIX_MAPPING** 環境変数を OpenShift サービスの名前に設定します。

2. **その他の内部データソース:** これらは、Red Hat レジストリーを介してデフォルトでは利用できない、OpenShift 上で実行されるデータソースです。これらのデータソースの設定は、OpenShift Secret に追加された環境ファイルによって提供されます。

3. **外部データソース:** 外部データソースの OpenShift.Configuration で実行されないデータソースは、OpenShift のシークレットに追加された環境ファイルによって提供されます。

データソース環境ファイルは、プロジェクトの OpenShift Secret に追加されます。これらの環境ファイルは、**ENV_FILES** 環境プロパティーを使用してテンプレート内で呼び出されます。

データソースは、特定の環境変数の値に基づいて自動的に作成されます。最も重要な環境変数は **DB_SERVICE_PREFIX_MAPPING** です。**DB_SERVICE_PREFIX_MAPPING** はデータソースの JNDI マッピングを定義します。この変数で使用できる値は、**POOLNAME-DATABASETYPE=PREFIX** トリプレットのコンマ区切りリストです。 説明を以下に示します。

- **POOLNAME** はデータソースの pool-name として使用されます。
- **DATABASETYPE** は使用するデータベースドライバーです。
- **PREFIX** は、データソースを設定するために使用される環境変数の名前に使用される接頭辞です。

起動スクリプトは、イメージの起動時に実行される個別のデータソースを **DB_SERVICE_PREFIX_MAPPING** 環境変数に定義された各 **POOLNAME-DATABASETYPE=PREFIX** トリプレットに対して作成します。

データソース設定環境変数の完全リストは、ここに記載されているデータソース設定環境変数のリストを参照してください。

5.1.4. OpenShift と互換性のある環境変数の JWS

ビルド設定は、Source-to-Image **build** コマンドに環境変数を含めることで変更できます (「OpenShift の JWS での Maven アーティファクトリーポジトリーミラーの使用」 を参照)。Red Hat JBoss Web Server for OpenShift イメージに有効な環境変数は以下のとおりです。

変数名	表示名	説明	値の例
ARTIFACT_DIR	該当なし	このディレクトリーからの **.war**、**.ear**、および **.jar** ファイルが **deployments** ディレクトリーにコピーされます。	target
APPLICATION_NAME	アプリケーション名	アプリケーションの名前	jws-app
CONTEXT_DIR	コンテキストディレクトリー	ビルドする Git プロジェクト内のパス。root プロジェクトディレクトリーの場合は空です。	tomcat-websocket-chat
GITHUB_WEBHOOK_SECRET	Github Webhook Secret	GitHub トリガーシークレット	[a-zA-Z0-9]{8} からの式
GENERIC_WEBHOOK_SECRET	Generic Webhook Secret	汎用ビルドトリガーシークレット	[a-zA-Z0-9]{8} からの式
HOSTNAME_HTTP	カスタム HTTP ルートのホスト名	http サービスルートのカスタムホスト名。デフォルトホスト名の場合は空白のままにします。	<application-name>-<project>.<default-domain-suffix>
HOSTNAME_HTTPS	カスタム HTTPS ルートのホスト名	https サービスルートのカスタムホスト名。デフォルトホスト名の場合は空白のままにします。	<application-name>-<project>.<default-domain-suffix>
IMAGE_STREAM_NAMESPACE	イメージストリーム名前空間	Red Hat ミドルウェアイメージの ImageStreams がインストールされている名前空間	openshift

変数名	表示名	説明	値の例
JWS_HTTPS_SECRET	シークレット名	証明書ファイルが含まれるシークレットの名前	jws-app-secret
JWS_HTTPS_CERTIFICATE	証明書名	シークレット内の証明書ファイルの名前	server.crt
JWS_HTTPS_CERTIFICATE_KEY	証明書キー名	シークレット内の証明書キーファイルの名前	server.key
JWS_HTTPS_CERTIFICATE_PASSWORD	証明書のパスワード	証明書のパスワード	P5ssw0rd
JWS_ADMIN_USERNAME	Jws 管理ユーザー名	JWS 管理アカウントのユーザー名	ADMIN
JWS_ADMIN_PASSWORD	Jws 管理パスワード	Jws 管理アカウントのパスワード	P5sw0rd
SOURCE_REPOSITORY_URL	Git リポジトリー URL	アプリケーションの Git ソース URI	https://github.com/jboss-openshift/openshift-quickstarts.git
SOURCE_REPOSITORY_REFERENCE	Git リファレンス	Git ブランチ/タグ参照	1.2
IMAGE_STREAM_NAMESPACE	イメージストリーム名前空間	Red Hat ミドルウェアイメージの ImageStreams がインストールされている名前空間	openshift
MAVEN_MIRROR_URL	Maven ミラー URL	設定する Maven ミラー/リポジトリーマネージャーの URL。	http://10.0.0.1:8080/repository/internal/

5.2. OPENSHIFT 用の JWS 上のバルブ

5.2.1. OpenShift と互換性のある環境変数の JWS (バルブコンポーネント)

以下の環境変数を定義して、バルブコンポーネントを関連付けられた Catalina コンテナーのリクエスト処理パイプラインに挿入することができます。

変数名	説明	値の例	デフォルト値

変数名	説明	値の例	デフォルト値
ENABLE_ACCESS_LOG	Access Log Valve を有効にして、標準出力チャンネルへのアクセスメッセージをログに記録します。	true	false

5.3. ログの確認

実行中のコンテナーのコンソールが提供する OpenShift ログまたはログを表示するには、以下を実行します。

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

アクセスログは **/opt/jws-5.4/tomcat/logs/** に保存されます。