Red Hat JBoss Web Server 5.5

Red Hat JBoss Web Server for OpenShift

Installing and using Red Hat JBoss Web Server for OpenShift

Last Updated: 2021-10-19

Red Hat JBoss Web Server 5.5 Red Hat JBoss Web Server for OpenShift

Installing and using Red Hat JBoss Web Server for OpenShift

Legal Notice

Abstract

Guide to using Red Hat JBoss Web Server for OpenShift

Table of Contents

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

1.1. OVERVIEW OF RED HAT JBOSS WEB SERVER FOR OPENSHIFT

The Apache Tomcat 9 component of Red Hat JBoss Web Server (JWS) 5.5 is available as a containerized image designed for OpenShift. Developers can use this image to build, scale, and test Java web applications for deployment across hybrid cloud environments.

For more information about supported configurations of the Middleware products running on OpenShift, refer to the article Support of Red Hat Middleware products and components on Red Hat OpenShift .

CHAPTER 2. BEFORE YOU BEGIN

2.1. THE DIFFERENCE BETWEEN RED HAT JBOSS WEB SERVER AND JWS FOR OPENSHIFT

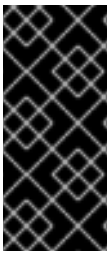
The differences between the JWS for OpenShift images and the regular release of JWS are:

- The location of ***JWS_HOME*** inside a JWS for OpenShift image is: ***/opt/jws-5.5/***.
- All load balancing is handled by the OpenShift router, not Apache HTTP Server mod_cluster or mod_jk connectors.

Documentation for JWS functionality not specific to JWS for OpenShift images is found in the Red Hat JBoss Web Server documentation.

2.2. VERSION COMPATIBILITY AND SUPPORT

See the xPaaS table on the OpenShift Container Platform Tested 3.X Integrations page and OpenShift Container Platform Tested 4.X Integrations page for details about OpenShift image version compatibility.



IMPORTANT

The 5.5 version of JWS for OpenShift images and application templates should be used for deploying new applications.

The 5.4 version of JWS for OpenShift images and application templates are deprecated and no longer receives updates.

2.3. SUPPORTED ARCHITECTURES BY JBOSS WEB SERVER

JBoss Web server supports the following architectures:

- x86_64 (AMD64)
- IBM Z (s390x) in the OpenShift environment
- IBM Power (ppc64le) in the OpenShift environment

Different images are supported for different architectures. The example codes in this guide demonstrate the commands for x86_64 architecture. If you are using other architectures, specify the relevant image name in the commands. See the Red Hat Container Catalog for more information about images.

2.4. HEALTH CHECKS FOR RED HAT CONTAINER IMAGES

All container images available for OpenShift have a health rating associated with it. You can find the health rating for Red Hat JBoss Web Server by navigating to the catalog of container images, searching for **JBoss Web Server** and selecting the 5.5 version.

For more information on how OpenShift container can be tested for liveliness and readiness, please refer to the following documentation

CHAPTER 3. GET STARTED

3.1. INITIAL SETUP

Before you follow the instructions in this guide, you must ensure that an OpenShift cluster is already installed and configured as a prerequisite. For more information about installing and configuring OpenShift clusters, see the OpenShift Container Platform Installing guide.

NOTE

The JWS for OpenShift application templates are distributed for Tomcat 9.

3.2. CONFIGURE AUTHENTICATION TO THE RED HAT CONTAINER REGISTRY

Before you can import and use the Red Hat JBoss Web Server image, you must first configure authentication to the Red Hat Container Registry.

Red Hat recommends that you create an authentication token using a registry service account to configure access to the Red Hat Container Registry. This means that you don't have to use or store your Red Hat account's username and password in your OpenShift configuration.

1. Follow the instructions on Red Hat Customer Portal to create an authentication token using a registry service account.
2. Download the YAML file containing the OpenShift secret for the token. You can download the YAML file from the **OpenShift Secret** tab on your token's **Token Information** page.
3. Create the authentication token secret for your OpenShift project using the YAML file that you downloaded:

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

4. Configure the secret for your OpenShift project using the following commands, replacing the secret name below with the name of your secret created in the previous step.

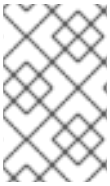
```
oc secrets link default 1234567-myserviceaccount-pull-secret --for=pull
oc secrets link builder 1234567-myserviceaccount-pull-secret --for=pull
```

See the OpenShift documentation for more information on other methods for configuring access to secured registries.

See the Red Hat Customer Portal for more information on configuring authentication to the Red Hat Container Registry.

3.3. IMPORT THE LATEST RED HAT JBOSS WEB SERVER IMAGE STREAMS AND TEMPLATES

You must import the latest Red Hat JBoss Web Server for OpenShift image streams and templates for your JDK into the namespace of your OpenShift project.



NOTE

Log in to the Red Hat Container Registry using your Customer Portal credentials to import the Red Hat JBoss Web Server image streams, templates and update image streams. For more information, see Red Hat Container Registry Authentication .

Import command for JDK 8

```
for resource in \
jws55-openjdk8-tomcat9-ubi8-basic-s2i.json \
jws55-openjdk8-tomcat9-ubi8-https-s2i.json \
jws55-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/jws55el8-v1.0/templates/${resource}
done
```

This command imports the following image streams and templates.

- The RHEL8 JDK 8 imagestream: jboss-webserver55-openjdk8-tomcat9-openshift-rhel8
- All templates specified in the command.

Import command for JDK 11

```
for resource in \
jws55-openjdk11-tomcat9-ubi8-basic-s2i.json \
jws55-openjdk11-tomcat9-ubi8-https-s2i.json \
jws55-openjdk11-tomcat9-ubi8-image-stream.json
do
oc replace -n openshift --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-webserver-5-openshift-image/jws55el8-v1.0/templates/${resource}
done
```

This command imports the following image streams and templates.

- The RHEL8 JDK 11 image stream: jboss-webserver55-openjdk11-tomcat9-openshift-rhel8
- All templates specified in the command.

3.3.1. Update Commands

- In order to update the core JWS 5.5 tomcat 9 OpenJDK8 RHEL8 OpenShift, you must execute

```
$ oc -n openshift import-image \
  jboss-webserver55-openjdk8-tomcat9-openshift-rhel8:1.0
```

- In order to update the core JWS 5.5 tomcat 9 OpenJDK11 RHEL8 OpenShift image, you must execute

```
$ oc -n openshift import-image \
  jboss-webserver55-openjdk11-tomcat9-openshift-rhel8:1.0
```

NOTE

The **1.0** tag at the end of each image you import refers to the stream version that is set in the image stream.

3.4. USING THE JWS FOR OPENSHIFT SOURCE-TO-IMAGE (S2I) PROCESS

To run and configure the JWS for OpenShift images, use the OpenShift S2I process with the application template parameters and environment variables.

The S2I process for the JWS for OpenShift images works as follows:

- If there is a Maven ***settings.xml*** file in the ***configuration***/source directory, it is moved to ***$HOME/.m2***/ of the new image.
  See the Apache Maven Project website for more information on Maven and the Maven ***settings.xml*** file.

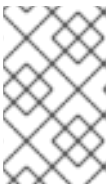
- If there is a ***pom.xml*** file in the source repository, a Maven build is triggered using the contents of the **$MAVEN_ARGS** environment variable.
  By default, the ***package*** goal is used with the ***openshift*** profile, including the arguments for skipping tests (**-DskipTests**) and enabling the Red Hat GA repository (**-Dcom.redhat.xpaas.repo.redhatga**).

- The results of a successful Maven build are copied to ***/opt/jws-5.5/tomcat/webapps***. This includes all WAR files from the source directory specified by the **$ARTIFACT_DIR** environment variable. The default value of **$ARTIFACT_DIR** is the ***target***/ directory.
  Use the ***MAVEN_ARGS_APPEND*** environment variable to modify the Maven arguments.

- All WAR files from the ***deployments***/source directory are copied to ***/opt/jws-5.5/tomcat/webapps***.

- All files in the ***configuration***/source directory are copied to ***/opt/jws-5.5/tomcat/conf/*** (excluding the Maven ***settings.xml*** file).

- All files in the ***lib***/source directory are copied to ***/opt/jws-5.5/tomcat/lib/***.



NOTE

If you want to use custom Tomcat configuration files, the file names should be the same as for a normal Tomcat installation. For example, ***context.xml*** and ***server.xml***.

See the Artifact Repository Mirrors section for guidance on configuring the S2I process to use a custom Maven artifacts repository mirror.

3.4.1. Create a JWS for OpenShift application using existing maven binaries

Existing applications are deployed on OpenShift using the **oc start-build** command.

Prerequisite: An existing **.war**, **.ear**, or **.jar** of the application to deploy on JWS for OpenShift.

1. Prepare the directory structure on the local file system.

Create a source directory containing any content required by your application not included in the binary (if required, see Using the JWS for OpenShift Source-to-Image (S2I) process), then create a subdirectory **deployments**/:

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

2. Copy the binaries (**.war**,**.ear**,**.jar**) to **deployments**/:

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

NOTE

Application archives in the **deployments**/ subdirectory of the source directory are copied to the **$JWS_HOME**/**tomcat**/**webapps**/ directory of the image being built on OpenShift. For the application to deploy, the directory hierarchy containing the web application data must be structured correctly (see Section 3.4, "Using the JWS for OpenShift Source-to-Image (S2I) process").

3. Log in to the OpenShift instance:

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

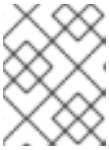
4. Create a new project if required:

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

5. Identify the JWS for OpenShift image stream to use for your application with **oc get is -n openshift**:

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

jboss-webserver50-tomcat9-openshift
```



NOTE

The option **-n openshift** specifies the project to use. **oc get is -n openshift** retrieves (**get**) the image stream resources (**is**) from the **openshift** project.

6. Create the new build configuration, specifying image stream and application name:

```
$ oc new-build --binary=true \
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel8:latest \
--name=<my-jws-on-openshift-app>
```

7. Instruct OpenShift to use the source directory created above for binary input of the OpenShift image build:

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

8. Create a new OpenShift application based on the image:

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

9. Expose the service to make the application accessible to users:

```
# 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. Retrieve the address of the exposed route:

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

11. To access the application in your browser: **http://*<address_of_exposed_route>*/*<my-war-ear-jar-filename-without-extension>***

3.4.2. Example: Creating a JWS for OpenShift application using existing maven binaries

The example below uses the tomcat-websocket-chat quickstart using the procedure from Section 3.4.1, "Create a JWS for OpenShift application using existing maven binaries".

3.4.2.1. Prerequisites:

A. **Get the WAR application archive or build the application locally.**

- Clone the source code:

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

- Configure the Red Hat JBoss Middleware Maven Repository
  - Additional information for the Red Hat JBoss Middleware Maven Repository
- Build the application:

```
$ 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. **Prepare the directory structure on the local file system.**
Create the source directory for the binary build on your local file system and the **deployments/** subdirectory. Copy the WAR archive to **deployments/**:

```
[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. To setup the example application on OpenShift

1. Log in to the OpenShift instance:

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

2. Create a new project if required:

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

3. Identify the JWS for OpenShift image stream to use for your application with **oc get is -n openshift**:

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

jboss-webserver50-tomcat9-openshift
```

4. Create new build configuration, specifying image stream and application name:

```
$ oc new-build --binary=true \
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel8: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. Start the binary build. Instruct OpenShift to use source directory for the binary input for the OpenShift image build:

```
$ *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-5.5/tomcat/webapps` for later deployment...
'/home/jboss/source/deployments/websocket-chat.war' -> '/opt/jws-
5.5/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. Create a new OpenShift application based on the image:

```
$ 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. Expose the service to make the application accessible to users:

```
# 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. Retrieve the address of the exposed route:

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

9. Access the application in your browser: **http://*<address_of_exposed_route>*/websocket-chat**

3.4.3. Create a JWS for OpenShift application from source code

For detailed instructions on creating new OpenShift applications from source code, see OpenShift.com - Creating an Application From Source Code.

NOTE

Before proceeding, ensure that the applications' data is structured correctly (see Section 3.4, “Using the JWS for OpenShift Source-to-Image (S2I) process”).

1. Log in to the OpenShift instance:

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

2. Create a new project if required:

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

3. Identify the JWS for OpenShift image stream to use for your application with **oc get is -n openshift**:

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

jboss-webserver50-tomcat9-openshift
```

4. Create the new OpenShift application from source code using Red Hat JBoss Web Server for OpenShift images, use the **--image-stream** option:

```
$ oc new-app \
 <source_code_location>\
```

```
--image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel8\
--name=<openshift_application_name>
```

For Example:

```
$ oc new-app \
 https://github.com/jboss-openshift/openshift-quickstarts.git#master \
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel8\
 --context-dir='tomcat-websocket-chat' \
 --name=jws-wsch-app
```

The source code is added to the image and the source code is compiled. The build configuration and services are also created.

5. To expose the application:

```
# 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. To retrieve the address of the exposed route:

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

7. To access the application in your browser:
http://*<address_of_exposed_route>*/***<java_application_name>***

3.5. ADDING ADDITIONAL JAR FILES IN TOMCAT/LIB/ DIRECTORY

Additional jar files can be added to **tomcat/lib/** directory using docker.

For adding jar files in **tomcat/lib/**

1. Get the image started in docker

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

2. Find the **CONTAINER ID**

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

3. Copy the library to **tomcat/lib/** directory

```
docker cp <yourLibrary> <CONTAINER ID>:/opt/jws-5.5/tomcat/lib/
```

4. Commit the changes to a new image

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

5. Create a new image tag

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

6. Push the image to a registry

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

CHAPTER 4. JWS OPERATOR

4.1. JBOSS WEB SERVER OPERATOR

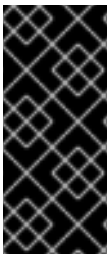
4.1.1. OpenShift Operators

The Operator Framework is a toolkit to manage Kubernetes native applications, called Operators, in an effective, automated, and scalable way. Operators make it easy to manage complex stateful applications on top of Kubernetes. All Operators are based around 3 key components: The Operator SDK, The Operator Lifecycle Manager, and OperatorHub.io. These tools allow you to develop your own operators, manage any operators you are using on your Kubernetes cluster, and discover or share any Operators the community creates.

The Red Hat JBoss Web Server project provides an Operator to manage its OpenShift images. This section covers how to build, test, and package the OpenShift Operator for JWS.

For full instructions on cluster setup please refer to the Openshift Documentation subsection 'Install'

Additionally, The JWS operator uses different environment variables than the JWS-on-OpenShift setup. A full listing of these parameters can be found here.



IMPORTANT

At this time, the **Use Session Clustering** functionality is available as technology preview (not supported). The clustering is **Off** by default. The current operator version uses the **DNS Membership Provider** which is limited due to DNS limitations. **InetAddress.getAllByName()** results are cached and as a result, session replications may not work while scaling up.

This guide covers installation, deployment, and deletion of the JWS Operator in detail. For a faster, but less detailed, guide Please refer to the quickstarts guide .

IMPORTANT

Currently, we only support JWS 5.4 images. Images older than 5.4 are **NOT** supported.

4.1.2. Installing the JWS Operator

This section covers the installation of the JWS Operator on the OpenShift Container Platform.

4.1.2.1. Prerequisites

- OpenShift Container Platform cluster using an account with **cluster admin** permissions (web console only)
- OpenShift Container Platform cluster using an account with operator installation permissions
- **oc** tool installed on your local system (CLI only)

4.1.2.2. Installing the JWS Operator - web console

1. Navigate to the 'Operators' tab, found in the menu on the left-hand side

2. This will open OpenShift OperatorHub. From here, search for JWS and select the 'JWS Operator'

3. A new menu should appear - Select your desired Capacity Level and then click 'Install' at the top to install the Operator.

4. You are now able to set up the operator installation. You will specify the following 3 options:

   - **Installation Mode:** Specify a specific namespace on your cluster to install. If you do not specify this, it will install the operator to all namespaces on your cluster by default.

   - **Update Channel:** The JWS operator is currently available only through one channel.

   - **Approval Strategy:** You can choose Automatic or Manual updates. If you choose Automatic updates for an installed Operator, when a new version of that Operator is available, Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention. If you select Manual updates, when a newer version of an Operator is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

5. Click 'Install' at the bottom. If you selected **Manual Approval Strategy**, you must approve the install plan before installation is complete. The JWS Operator will now appear in the 'Installed Operators' section of the 'Operators' tab.

4.1.2.3. Installing the JWS Operator - command line interface

1. Inspect the JWS operator to verify its supported installModes and available channels using the following commands:

```
$   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. An OperatorGroup is an OLM resource that selects target namespaces in which to generate required RBAC access for all Operators in the same namespace as the OperatorGroup.
The namespace to which you subscribe the Operator must have an OperatorGroup that matches the InstallMode of the Operator, either the AllNamespaces or SingleNamespace mode. If the Operator you intend to install uses the AllNamespaces, then the openshift-operators namespace already has an appropriate OperatorGroup in place.

   However, if the Operator uses the SingleNamespace mode, exactly one OperatorGroup has to be created in that namespace. To check actual list of OperatorGroups use the following command:

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

   Example of an output for OperatorGroup listing:

```
NAME       AGE
mygroup    17h
```

NOTE

The web console version of this procedure handles the creation of the OperatorGroup and Subscription objects automatically behind the scenes for you when choosing SingleNamespace mode.

- Create an OperatorGroup object YAML file, for example: **OperatorGroupExample.yaml**:

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

<project_name> is the namespace of the project where you install the operator (oc project -q). <operatorgroup_name> is the name of the OperatorGroup.

- Create the OperatorGroup object using the following command:

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

3. Create a subscription object YAML file, for example **jws-operator-sub.yaml**. Configure your **Subscription** object YAML file to look as follows:

```
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> is the namespace of the project where you install the operator (oc project -q). to install in all namespace use openshift-operators.

The **source** is the Catalog Source. This is the value from the **$ oc describe packagemanifests jws-operator -n openshift-marketplace | grep "Catalog Source:"** command we ran in step 1 of this section. The value should be **redhat-operators**.

4. Create the **Subscription** object from the YAML file with the following command:

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

To verify a successful installation, run the following command:

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

NAME	DISPLAY	VERSION	REPLACES PHASE
jws-operator.V1.0.0	JBoss Web Server Operator	1.0.0	Succeeded

4.1.3. Deploying an existing JWS image

1. Ensure your operator is installed with the following command:

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

   Or if you need a more detailed output:

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

2. Prepare your image and push it to the desired location. In this example it is pushed to **quay.io/<USERNAME>/tomcat-demo:latest**

3. Create a **Custom Resource** WebServer .yaml file. In this example a file called **webservers_cr.yaml** is used. Your file should follow this format:

```
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. Deploy your webapp, from the directory in which you created it, with the following command:

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

NOTE

The operator will create a route automatically. You can verify the route with the following command:

```
$   oc get routes
```

For more information on routes, please see the OpenShift documentation

5. If you need delete the **webserver** you created in step 4:

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

OR

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

4.1.4. Deleting Operators from a cluster

4.1.4.1. Prerequisites

- OpenShift Container platform cluster with admin privileges (Alternatively, you can circumvent this requirement by following these instructions)
- **oc** tool installed on your local system (CLI only)

4.1.4.2. Deleting an operator from a cluster - web console

1. In the left hand menu, click 'Operators' → 'Installed Operators'
2. Underneath 'Operator Details' select the 'Actions' menu, and then click 'Uninstall Operator'
3. Selecting this option will remove the Operator, any Operator deployments, and Pods. **HOWEVER** removing the Operator **will not** remove any of its custom resource definitions or custom resources, including CRDs or CRs. If your operator has deployed applications on the cluster or configured off-cluster resources, these will continue to run and need to be cleaned up manually.

4.1.4.3. Deleting an operator from a cluster - command line interface

1. Check the current version of the subscribed operator in the **currentCSV** field by using the following command:

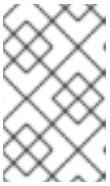
```
$ oc get subscription jws-operator -n <project_name> -o yaml | grep currentCSV
  f:currentCSV: {}
  currentCSV: jws-operator.v1.0.0
```

NOTE

In the above command, **<project_name>** refers to the namespace of the project where you installed the operator. If your operator was installed to all namespaces, use **openshift-operators** in place of **<project_name>**.

2. Delete the operator's subscription using the following command:

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



NOTE

In the above command, **<project_name>** refers to the namespace of the project where you installed the operator. If your operator was installed to all namespaces, use **openshift-operators** in place of **<project_name>**.

3. Delete the CSV for the operator in the target namespace using the currentCSV value from the previous step, using the following command:

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

where **<currentCSV>** is the value obtained in step 1

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

NOTE

In the above command, **<project_name>** refers to the namespace of the project where you installed the operator. If your operator was installed to all namespaces, use **openshift-operators** in place of **<project_name>**.

4.1.5. Additional resources

For additional information on Operators, you may refer to the formal OpenShift Documentation:

What are Operators?

And

Openshift Container Platform

CHAPTER 5. REFERENCE

5.1. SOURCE-TO-IMAGE (S2I)

The Red Hat JBoss Web Server for OpenShift image includes S2I scripts and Maven.

5.1.1. Using maven artifact repository mirrors with JWS for OpenShift

A Maven repository holds build artifacts and dependencies, such as the project jars, library jars, plugins or any other project specific artifacts. It also defines locations to download artifacts from while performing the S2I build. Along with using the Maven Central Repository, some organizations also deploy a local custom repository (mirror).

Benefits of using a local mirror are:

- Availability of a synchronized mirror, which is geographically closer and faster.
- Greater control over the repository content.
- Possibility to share artifacts across different teams (developers, CI), without the need to rely on public servers and repositories.
- Improved build times.

A Maven repository manager can serve as local cache to a mirror. Assuming that the repository manager is already deployed and reachable externally at ***http://10.0.0.1:8080/repository/internal/***, the S2I build can use this repository. To use an internal Maven repository, add the **MAVEN_MIRROR_URL** environment variable to the build configuration of the application.

For a new build configuration, use the **--build-env** option with **oc new-app** or **oc new-build**:

```
$ oc new-app \
 https://github.com/jboss-openshift/openshift-quickstarts.git#master \
 --image-stream=jboss-webserver<version>-openjdk8-tomcat9-openshift-rhel8:latest\*
 --context-dir='tomcat-websocket-chat' \
 --build-env MAVEN_MIRROR_URL=http://10.0.0.1:8080/repository/internal/ \
 --name=jws-wsch-app
```

For an existing build configuration:

1. Identify the build configuration which requires the **MAVEN_MIRROR_URL** variable:

```
$ oc get bc -o name

buildconfig/jws
```

2. Add the **MAVEN_MIRROR_URL** environment variable to **buildconfig/jws**:

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

buildconfig "jws" updated
```

3. Verify the build configuration has updated:

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

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

4. Schedule a new build of the application using **oc start-build**

NOTE

During application build, Maven dependencies are download from the repository manager, instead of the default public repositories. Once the build has finished, the mirror contains all the dependencies retrieved and used during the build.

5.1.2. Scripts included on the Red Hat JBoss Web Server for OpenShift image

run

runs Catalina (Tomcat)

assemble

uses Maven to build the source, create package (**.war**) and move it to the **$JWS_HOME/tomcat/webapps** directory.

5.1.3. JWS for OpenShift datasources

There are 3 types of data sources:

1. **Default Internal Datasources:** These are PostgreSQL, MySQL, and MongoDB. These datasources are available on OpenShift by default through the Red Hat Registry and do not require additional environment files to be configured for image streams. To make a database discoverable and used as a datasource, set the **DB_SERVICE_PREFIX_MAPPING** environment variable to the name of the OpenShift service.

2. **Other Internal Datasources:** These are datasources not available by default through the Red Hat Registry but run on OpenShift. Configuration of these datasources is provided by environment files added to OpenShift Secrets.

3. **External Datasources:** Datasources not run on OpenShift.Configuration of external datasources is provided by environment files added to OpenShift Secrets.

The datasources environment files are added to the OpenShift Secret for the project. These environment files are then called within the template using the **ENV_FILES** environment property.

Datasources are automatically created based on the value of certain environment variables.The most important environment variable is **DB_SERVICE_PREFIX_MAPPING**.
DB_SERVICE_PREFIX_MAPPING defines JNDI mappings for the datasources. The allowed value for this variable is a comma-separated list of **POOLNAME-DATABASETYPE=PREFIX** triplets, where:

- **POOLNAME** is used as the pool-name in the datasource.
- **DATABASETYPE** is the database driver to use.
- **PREFIX** is the prefix used in the names of environment variables that are used to configure the datasource.

For each **POOLNAME-DATABASETYPE=PREFIX** triplet defined in the **DB_SERVICE_PREFIX_MAPPING** environment variable, the launch script creates a separate datasource, which is executed when running the image.

For a full listing of datasource configuration environment variables, please see the Datasource Configuration Environment Variables list given here.

5.1.4. JWS for OpenShift compatible environment variables

The build configuration can be modified by including environment variables to the Source-to-Image **build** command (see Section 5.1.1, "Using maven artifact repository mirrors with JWS for OpenShift"). The valid environment variables for the Red Hat JBoss Web Server for OpenShift images are:

Variable Name	Display Name	Description	Example Value
ARTIFACT_DIR	N/A	**.war**, **.ear**, and **.jar** files from this directory will be copied into the **deployments** directory	target
APPLICATION_NAME	Application Name	The name for the application	jws-app
CONTEXT_DIR	Context Directory	Path within Git project to build; empty for root project directory	tomcat-websocket-chat
GITHUB_WEBHOOK_SECRET	Github Webhook Secret	Github trigger secret	Expression from: [a-zA-Z0-9]{8}
GENERIC_WEBHOOK_SECRET	Generic Webhook Secret	Generic build trigger secret	Expression from: [a-zA-Z0-9]{8}
HOSTNAME_HTTP	Custom HTTP Route Hostname	Custom hostname for http service route. Leave blank for default hostname	<application-name>-<project>.<default-domain-suffix>
HOSTNAME_HTTPS	Custom HTTPS Route Hostname	Custom hostname for https service route. Leave blank for default hostname	<application-name>-<project>.<default-domain-suffix>
IMAGE_STREAM_NAMESPACE	Imagestream Namespace	Namespace in which the ImageStreams for Red Hat Middleware images are installed	openshift
JWS_HTTPS_SECRET	Secret Name	The name of the secret containing the certificate files	jws-app-secret

Variable Name	Display Name	Description	Example Value
JWS_HTTPS_CERTIFICATE	Certificate Name	The name of the certificate file within the secret	server.crt
JWS_HTTPS_CERTIFICATE_KEY	Certificate Key Name	The name of the certificate key file within the secret	server.key
JWS_HTTPS_CERTIFICATE_PASSWORD	Certificate Password	The Certificate Password	P5ssw0rd
JWS_ADMIN_USERNAME	JWS Admin Username	JWS Admin account username	ADMIN
JWS_ADMIN_PASSWORD	JWS Admin Password	JWS Admin account password	P5sw0rd
SOURCE_REPOSITORY_URL	Git Repository URL	Git source URI for Application	*https://github.com/jboss-openshift/openshift-quickstarts.git*
SOURCE_REPOSITORY_REFERENCE	Git Reference	Git branch/tag reference	1.2
IMAGE_STREAM_NAMESPACE	Imagestream Namespace	Namespace in which the ImageStreams for Red Hat Middleware images are installed	openshift
MAVEN_MIRROR_URL	Maven Mirror URL	URL of a Maven mirror/repository manager to configure.	http://10.0.0.1:8080/repository/internal/

5.2. VALVES ON JWS FOR OPENSHIFT

5.2.1. JWS for OpenShift compatible environmental variables (valve component)

You can define the following environment variables to insert the valve component into the request processing pipeline for the associated Catalina container.

Variable Name	Description	Example Value	Default Value
ENABLE_ACCESS_LOG	Enable the Access Log Valve to log access messages to the standard output channel.	*true*	*false*

5.3. CHECKING LOGS

To view the OpenShift logs or the logs provided by a running container's console:

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

Access logs are stored in ***/opt/jws-5.5/tomcat/logs/***.