



Red Hat JBoss Web Server 6.0

Red Hat JBoss Web Server Operator

Installing and using the Red Hat JBoss Web Server Operator 2.0 for OpenShift

Red Hat JBoss Web Server 6.0 Red Hat JBoss Web Server Operator

Installing and using the Red Hat JBoss Web Server Operator 2.0 for OpenShift

Legal Notice

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.

Abstract

Install and use the Red Hat JBoss Web Server Operator 2.0 to manage web applications in Red Hat OpenShift

Table of Contents

PROVIDING FEEDBACK ON RED HAT JBOSS WEB SERVER DOCUMENTATION	3
MAKING OPEN SOURCE MORE INCLUSIVE	4
MAKING OPEN SOURCE MORE INCLUSIVE	5
CHAPTER 1. RED HAT JBOSS WEB SERVER OPERATOR	6
CHAPTER 2. WHAT IS NEW IN THE JWS OPERATOR 2.0 RELEASE?	7
Level-2 Operator capabilities	7
Enabling level-2 seamless integration for new images	7
Level-2 seamless integration for rebuilding existing images	8
Support for Red Hat JBoss Web Server metering labels	8
Enhanced webImage parameter	9
Enhanced webApp parameter	9
CHAPTER 3. JWS OPERATOR INSTALLATION FROM OPERATORHUB	11
3.1. INSTALLING THE JWS OPERATOR BY USING THE WEB CONSOLE	11
3.2. INSTALLING THE JWS OPERATOR BY USING THE COMMAND LINE	12
CHAPTER 4. DEPLOYING AN EXISTING JWS IMAGE	16
CHAPTER 5. JWS OPERATOR DELETION FROM A CLUSTER	18
5.1. DELETING THE JWS OPERATOR BY USING THE WEB CONSOLE	18
5.2. DELETING THE JWS OPERATOR BY USING THE COMMAND LINE	18
CHAPTER 6. CREATING A SECRET FOR A GENERIC OR GITHUB WEBHOOK	20
CHAPTER 7. JWS OPERATOR CRD PARAMETERS	23
7.1. CRD PARAMETER HIERARCHY	23
7.2. CRD PARAMETER DETAILS	24

PROVIDING FEEDBACK ON RED HAT JBOSS WEB SERVER DOCUMENTATION

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

Procedure

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

MAKING OPEN SOURCE MORE INCLUSIVE

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

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. RED HAT JBOSS WEB SERVER OPERATOR

An Operator is a Kubernetes-native application that makes it easy to manage complex stateful applications in Kubernetes and OpenShift environments. Red Hat JBoss Web Server (JWS) provides an Operator to manage JWS for OpenShift images. You can use the JWS Operator to create, configure, manage, and seamlessly upgrade instances of web server applications in OpenShift.

Operators include the following key concepts:

- The *Operator Framework* is a toolkit to manage Operators in an effective, automated, and scalable way. The Operator Framework consists of three main components:
 - You can use *OperatorHub* to discover Operators that you want to install.
 - You can use the *Operator Lifecycle Manager* (OLM) to install and manage Operators in your OpenShift cluster.
 - You can use the *Operator SDK* if you want to develop your own custom Operators.
- An *Operator group* is an OLM resource that provides multitenant configuration to OLM-installed Operators. An Operator group selects target namespaces in which to generate role-based access control (RBAC) for all Operators that are deployed in the same namespace as the **OperatorGroup** object.
- *Custom resource definitions* (CRDs) are a Kubernetes extension mechanism that Operators use. CRDs allow the custom objects that an Operator manages to behave like native Kubernetes objects. The JWS Operator provides a set of CRD parameters that you can specify in custom resource files for web server applications that you want to deploy.

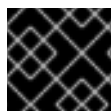
This document describes how to install the JWS Operator, deploy an existing JWS image, and delete Operators from a cluster. This document also provides details of the CRD parameters that the JWS Operator provides.



NOTE

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.

For a faster but less detailed guide to deploying a prepared image or building an image from an existing image stream, see the JWS Operator [QuickStart](#) guide.



IMPORTANT

Red Hat supports images for JWS 5.4 or later versions only.

Additional resources

- [Understanding Operators](#)
- [Installing and configuring OpenShift Container Platform clusters](#)

CHAPTER 2. WHAT IS NEW IN THE JWS OPERATOR 2.0 RELEASE?

The JWS Operator 2.0 release provides level-2 Operator capabilities such as seamless integration. JWS Operator 2.0 also supports Red Hat JBoss Web Server metering labels and includes some enhanced Custom Resource Definition (CRD) parameters.

Level-2 Operator capabilities

JWS Operator 2.0 provides the following level-2 Operator capability features:

- Enables seamless upgrades
- Supports patch and minor version upgrades
- Manages web servers deployed by the JWS Operator 1.1.x.

Enabling level-2 seamless integration for new images

The **DeploymentConfig** object definition includes a trigger that OpenShift uses to deploy new pods when a new image is pushed to the image stream. The image stream can monitor the repository for new images or you can instruct the image stream that a new image is available for use.

Procedure

1. In your project namespace, create an image stream by using the **oc import-image** command to import the tag and other information for an image.

For example:

```
oc import-image <my-image>-imagestream:latest \
--from=quay.io/$user/<my-image>:latest \
--confirm
```

In the preceding example, replace each occurrence of **<my-image>** with the name of the image that you want to import.

The preceding command creates an image stream named **<my-image>-imagestream** by importing information for the **quay.io/\$user/<my-image>** image. For more information about the format and management of image streams, see [Managing image streams](#).

2. Create a custom resource of the **WebServer** kind for the web application that you want the JWS Operator to deploy whenever the image stream is updated. You can define the custom resource in YAML file format.

For example:

```
apiVersion: web.servers.org/v1alpha1
kind: WebServer
metadata:
  name: <my-image>
spec:
  # Add fields here
  applicationName: my-app
  useSessionClustering: true
  replicas: 2
```

```
weblmageStream:
  imageStreamNamespace: <project-name>
  imageStreamName: <my-image>-imagestream
```

3. Trigger an update to the image stream by using the **oc tag** command.
For example:

```
oc tag quay.io/$user/<my-image> <my-image>-imagestream:latest --scheduled
```

The preceding command causes OpenShift Container Platform to update the specified image stream tag periodically. This period is a cluster-wide setting that is set to 15 minutes by default.

Level-2 seamless integration for rebuilding existing images

The **BuildConfig** object definition includes a trigger for image stream updates and a webhook, which is either a GitHub or Generic webhook, that enables the rebuilding of images when the webhook is triggered by Git or GitHub.

For more information about creating a secret for a webhook, see [Creating a secret for a generic or GitHub webhook](#).

For more information about configuring a generic or GitHub webhook in a custom resource WebServer file, see [JWS Operator CRD parameters](#).

Support for Red Hat JBoss Web Server metering labels

JWS Operator 2.0 supports the ability to add metering labels to the Red Hat JBoss Web Server pods that the JWS Operator creates.

Red Hat JBoss Web Server can use the following metering labels:

- **com.company: Red_Hat**
- **rht.prod_name: Red_Hat_Runtimes**
- **rht.prod_ver: 2023-Q4**
- **rht.comp: JBoss_Web_Server**
- **rht.comp_ver: 6.0.0**
- **rht.subcomp: Tomcat 10**
- **rht.subcomp_t: application**

You can add labels under the **metadata** section in the custom resource **WebServer** file for a web application that you want to deploy. For example:

```
---
apiVersion: web.servers.org/v1alpha1
kind: WebServer
metadata:
  name: <my-image>
  labels:
    com.company: Red_Hat
    rht.prod_name: Red_Hat_Runtimes
    rht.prod_ver: 2023-Q4
    rht.comp: JBoss_Web_Server
```

```
rht.comp_ver: 6.0.0
rht.subcomp: Tomcat 10
rht.subcomp_t: application
spec:
----
```



NOTE

If you change any label key or label value for a deployed web server, the JWS Operator redeploys the web server application. If the deployed web server was built from source code, the JWS Operator also rebuilds the web server application.

Enhanced **webImage** parameter

In the JWS Operator 2.0 release, the **webImage** parameter in the CRD contains the following additional fields:

- **imagePullSecret**

The secret that the JWS Operator uses to pull images from the repository



NOTE

The secret must contain the key **.dockerconfigjson**. The JWS Operator mounts and uses the secret (for example, **--authfile /mount_point/.dockerconfigjson**) to pull the images from the repository. The **Secret** object definition file might contain server username and password values or tokens to allow access to images in the image stream, the builder image, and images built by the JWS Operator.

- **webApp**

A set of parameters that describe how the JWS Operator builds the web server application

Enhanced **webApp** parameter

In the JWS Operator 2.0 release, the **webApp** parameter in the CRD contains the following additional fields:

- **name**

The name of the web server application

- **sourceRepositoryURL**

The URL where the application source files are located

- **sourceRepositoryRef**

The branch of the source repository that the Operator uses

- **sourceRepositoryContextDir**

The subdirectory where the **pom.xml** file is located and where the **mvn install** command must be run

- **webAppWarImage**

The URL of the images where the JWS Operator pushes the built image

- **webAppWarImagePushSecret**

The secret that the JWS Operator uses to push images to the repository

- **builder**

A set of parameters that contain all the information required to build the web application and create and push the image to the image repository

**NOTE**

To ensure that the builder can operate successfully and run commands with different user IDs, the builder must have access to the **anyuid** security context constraint (SCC).

To grant the builder access to the **anyuid** SCC, enter the following command:

```
oc adm policy add-scc-to-user anyuid -z builder
```

The **builder** parameter contains the following fields:

- **image**

The image of the container where the web application is built (for example, **quay.io/\$user/tomcat10-buildah**)

- **imagePullSecret**

The secret (if specified) that the JWS Operator uses to pull the builder image from the repository

- **applicationBuildScript**

The script that the builder image uses to build the application **.war** file and move it to the **/mnt** directory

**NOTE**

If you do not specify a value for this parameter, the builder image uses a default script that uses Maven and Buildah.

Additional resources

- [Operator Capability Levels](#)

CHAPTER 3. JWS OPERATOR INSTALLATION FROM OPERATORHUB

You can install the JWS Operator from OperatorHub to facilitate the deployment and management of JBoss Web Server applications in an OpenShift cluster. OperatorHub is a component of the Operator Framework that you can use to discover Operators that you want to install. OperatorHub works in conjunction with the Operator Lifecycle Manger (OLM), which installs and manages Operators in a cluster.

You can install the JWS Operator from OperatorHub in either of the following ways:

- [Use the OpenShift web console.](#)
- [Use the `oc` command-line tool.](#)

3.1. INSTALLING THE JWS OPERATOR BY USING THE WEB CONSOLE

If you want to install the JWS Operator by using a graphical user interface, you can use the OpenShift web console to install the JWS Operator.



NOTE

When you install the JWS Operator by using the web console, and the Operator is using **SingleNamespace** installation mode, the **OperatorGroup** and **Subscription** objects are installed automatically.

Prerequisites

- You have deployed an OpenShift Container Platform cluster by using an account with cluster administrator and Operator installation permissions.

Procedure

1. Open the web console and select **Operators > OperatorHub**.
2. In the **Filter by keyword** search field, type "JWS".
3. Select the JWS Operator.
4. On the **JBoss Web Server Operator** menu, select the **Capability level** that you want to use and click **Install**.
5. On the **Install Operator** page, perform the following steps:
 - a. Select the **Update channel** where the JWS Operator is available.



NOTE

The JWS Operator is currently available through one channel only.

- b. Select the **Installation mode** for the Operator.
You can install the Operator to all namespaces or to a specific namespace on the cluster. If you select the specific namespace option, use the **Installed Namespace** field to specify the namespace where you want to install the Operator.

**NOTE**

If you do not specify a namespace, the Operator is installed to all namespaces on the cluster by default.

- c. Select the **Approval strategy** for the Operator.

Consider the following guidelines:

- If you select **Automatic** updates, when a new version of the Operator is available, the OLM upgrades the running instance of your Operator automatically.
- If you select **Manual** updates, when a newer version of the Operator is available, the OLM creates an update request. As a cluster administrator, you must then manually approve the update request to ensure that the Operator is updated to the new version.

6. Click **Install**.

**NOTE**

If you have selected a **Manual** approval strategy, you must approve the install plan before the installation is complete.

The JWS Operator then appears in the **Installed Operators** section of the **Operators** tab.

3.2. INSTALLING THE JWS OPERATOR BY USING THE COMMAND LINE

If you want to install the JWS Operator by using a command-line interface, you can use the **oc** command-line tool to install the JWS Operator. The JWS Operator that Red Hat provides is named **jws-operator**.

The steps to install the JWS Operator from the command line include verifying the supported installation modes and available channels for the Operator and creating a Subscription object. Depending on the installation mode that the Operator uses, you might also need to create an Operator group in the project namespace before you create the Subscription object.

Prerequisites

- You have deployed an OpenShift Container Platform cluster by using an account with Operator installation permissions.
- You have installed the **oc** tool on your local system.

Procedure

1. To inspect the JWS Operator, perform the following steps:
 - a. View the list of JWS Operators that are available to the cluster from OperatorHub:

```
$ oc get packagemanifests -n openshift-marketplace | grep jws
```

The preceding command displays the name, catalog, and age of each available Operator.

For example:


```
NAME          CATALOG          AGE
jws-operator  Red Hat Operators 16h
```

- b. Inspect the JWS Operator to verify the supported installation modes and available channels for the Operator:

```
$ oc describe packagemanifests jws-operator -n openshift-marketplace
```

2. Check the actual list of Operator groups:

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

In the preceding example, replace **<project_name>** with your OpenShift project name.

The preceding command displays the name and age of each available Operator group.

For example:

```
NAME    AGE
mygroup 17h
```

3. If you need to create an Operator group, perform the following steps:



NOTE

If the Operator you want to install uses **SingleNamespace** installation mode and you do not already have an appropriate Operator group in place, you must complete this step to create an Operator group. You must ensure that you create only one Operator group in the specified namespace.

If the Operator you want to install uses **AllNamespaces** installation mode or you already have an appropriate Operator group in place, you can ignore this step.

- a. Create a YAML file for the **OperatorGroup** object.
For example:

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

In the preceding example, replace **<operatorgroup_name>** with the name of the Operator group that you want to create, and replace **<project_name>** with the name of the project where you want to install the Operator. To view the project name, you can run the **oc project -q** command.

- b. Create the **OperatorGroup** object from the YAML file:

```
$ oc apply -f <filename>.yaml
```

In the preceding example, replace **<filename>.yaml** with the name of the YAML file that you have created for the **OperatorGroup** object.

4. To create a Subscription object, perform the following steps:

a. Create a YAML file for the **Subscription** object.

For example:

```
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
```

In the preceding example, replace **<project_name>** with the name of the project where you want to install the Operator. To view the project name, you can run the **oc project -q** command.

The namespace that you specify must have an **OperatorGroup** object that has the same installation mode setting as the Operator. If the Operator uses **AllNamespaces** installation mode, replace **<project_name>** with **openshift-operators**, which already provides an appropriate Operator group. If the Operator uses **SingleNamespace** installation mode, ensure that this namespace has only one **OperatorGroup** object.

Ensure that the **source** setting matches the **Catalog Source** value that was displayed when you verified the available channels for the Operator (for example, **redhat-operators**).

b. Create the **Subscription** object from the YAML file:

```
$ oc apply -f <filename>.yaml
```

In the preceding example, replace **<filename>.yaml** with the name of the YAML file that you have created for the **Subscription** object.

Verification

- To verify that the JWS Operator is installed successfully, enter the following command:

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

In the preceding example, replace **<project_name>** with the name of the project where you have installed the Operator.

The preceding command displays details of the installed Operator.

For example:

NAME	DISPLAY	VERSION	REPLACES	PHASE
jws-operator.v2.0.x	JWS Operator	2.0.x	jws-operator.v2.0.y	Succeeded

In the preceding output, **2.0.x** represents the current Operator version (for example, **2.0.6**), and **2.0.y** represents the previous Operator version that the current version replaces (for example, **2.0.5**).

CHAPTER 4. DEPLOYING AN EXISTING JWS IMAGE

You can use the JWS Operator to facilitate the deployment of an existing image for a web server application that you want to deploy in an OpenShift cluster. In this situation, you must create a custom resource **WebServer** file for the web server application that you want to deploy. The JWS Operator uses the custom resource **WebServer** file to handle the application deployment.

Prerequisites

- You have [installed the JWS Operator from OperatorHub](#).
To ensure that the JWS Operator is installed, enter the following command:

```
$ oc get deployment.apps/jws-operator-controller-manager
```

The preceding command displays the name and status details of the Operator.

For example:

```
NAME          READY UP-TO-DATE AVAILABLE AGE
jws-operator  1/1   1           1       15h
```



NOTE

If you want to view more detailed output, you can use the following command:

```
oc describe deployment.apps/jws-operator-controller-manager
```

Procedure

- Prepare your image and push it to the location where you want to display the image (for example, **quay.io/<USERNAME>/tomcat-demo:latest**).
- To create a custom resource file for your web server application, perform the following steps:
 - Create a YAML file named, for example, **webservers_cr.yaml**.
 - Enter details in the following format:

```
apiVersion: web.servers.org/v1alpha1
kind: WebServer
metadata:
  name: <image name>
spec:
  # Add fields here
  applicationName: <application name>
  replicas: 2
webImage:
  applicationImage: <URL of the image>
```

For example:

```
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

```

3. To deploy your web application, perform the following steps:
 - a. Go to the directory where you have created the web application.
 - b. Enter the following command:

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

The preceding command displays a message to confirm that the web application is deployed.

For example:

```
webserver/example-image-webserver created
```

When you run the preceding command, the Operator also creates a route automatically.

4. Verify the route that the Operator has automatically created:

```
$ oc get routes
```

5. Optional: Delete the **webserver** that you created in [Step 3](#):

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



NOTE

Alternatively, you can delete the **webserver** by deleting the YAML file. For example:

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

Additional resources

- [Route configuration: Creating an HTTP-based route](#)

CHAPTER 5. JWS OPERATOR DELETION FROM A CLUSTER

If you no longer need to use the JWS Operator, you can subsequently delete the JWS Operator from a cluster.

You can delete the JWS Operator from a cluster in either of the following ways:

- [Use the OpenShift web console.](#)
- [Use the **oc** command-line tool.](#)

5.1. DELETING THE JWS OPERATOR BY USING THE WEB CONSOLE

If you want to delete the JWS Operator by using a graphical user interface, you can use the OpenShift web console to delete the JWS Operator.

Prerequisites

- You have deployed an OpenShift Container Platform cluster by using an account with **cluster admin** permissions.



NOTE

If you do not have **cluster admin** permissions, you can circumvent this requirement. For more information, see [Allowing non-cluster administrators to install Operators](#).

Procedure

1. Open the web console and click **Operators > Installed Operators**
2. Select the **Actions** menu and click **Uninstall Operator**.



NOTE

The **Uninstall Operator** option automatically removes the Operator, any Operator deployments, and Pods.

Deleting the Operator does not remove any custom resource definitions or custom resources for the Operator, including CRDs or CRs. If the Operator has deployed applications on the cluster, or if the Operator has configured resources outside the cluster, you must clean up these applications and resources manually.

5.2. DELETING THE JWS OPERATOR BY USING THE COMMAND LINE

If you want to delete the JWS Operator by using a command-line interface, you can use the **oc** command-line tool to delete the JWS Operator.

Prerequisites

- You have deployed an OpenShift Container Platform cluster by using an account with **cluster admin** permissions.

**NOTE**

If you do not have **cluster admin** permissions, you can circumvent this requirement. For more information, see [Allowing non-cluster administrators to install Operators](#).

- You have installed the **oc** tool on your local system.

Procedure

1. Check the current version of the subscribed Operator:

```
$ oc get subscription jws-operator -n <project_name> -o yaml | grep currentCSV
```

In the preceding example, replace **<project_name>** with the namespace of the project where you installed the Operator. If your Operator was installed to all namespaces, replace **<project_name>** with **openshift-operators**.

The preceding command displays the following output, where **v2.0.x** refers to the Operator version (for example, **v2.0.6**):

```
f:currentCSV: {}
currentCSV: jws-operator.v2.0.x
```

2. Delete the subscription for the Operator:

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

In the preceding example, replace **<project_name>** with the namespace of the project where you installed the Operator. If your operator was installed to all namespaces, replace **<project_name>** with **openshift-operators**.

3. Delete the CSV for the Operator in the target namespace:

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

In the preceding example, replace **<currentCSV>** with the **currentCSV** value that you obtained in [Step 1](#) (for example, **jws-operator.v2.0.6**). Replace **<project_name>** with the namespace of the project where you installed the Operator. If your operator was installed to all namespaces, replace **<project_name>** with **openshift-operators**.

The preceding command displays a message to confirm that the CSV is deleted.

For example:

```
clusterserviceversion.operators.coreos.com "jws-operator.v2.0.x" deleted
```

CHAPTER 6. CREATING A SECRET FOR A GENERIC OR GITHUB WEBHOOK

You can create a secret that you can use with a generic or GitHub webhook to trigger application builds in a Git repository. Depending on the type of Git hosting platform that you use for your application code, the JWS Operator provides a **genericWebhookSecret** parameter and a **githubWebhookSecret** parameter that you can use to specify the secret in the custom resource file for a web application.

Procedure

1. Create an encoded secret string:
 - a. Create a file named, for example, **secret.txt**.
 - b. In the **secret.txt** file, enter the secret string in plain text.
For example:

```
qwerty
```

- c. To encode the string, enter the following command:

```
base64 secret.txt
```

The preceding command displays the encoded string.

For example:

```
cXdlcnR5Cg==
```

2. Create a **secret.yaml** file that defines an object of kind **Secret**.
For example:

```
kind: Secret
apiVersion: v1
metadata:
  name: jws-secret
data:
  WebHookSecretKey: cXdlcnR5Cg==
```

In the preceding example, **jws-secret** is the name of the secret and **cXdlcnR5Cg==** is the encoded secret string.

3. To create the secret, enter the following command:

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

The preceding command displays a message to confirm that the secret is created.

For example:

```
secret/jws-secret created
```

Verification

Verification

1. Get the URL for the webhook:

```
oc describe BuildConfig | grep webhooks
```

The preceding command generates the webhook URL in the following format:

```
https://<host>:
<port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/
<secret>/generic
```

2. Create a minimal JSON file named, for example, **payload.json**:

```
{}
```

3. To send a request to the webhook, enter the following **curl** command:

```
curl -H "X-GitHub-Event: push" -H "Content-Type: application/json" -k -X POST --data-binary
@payload.json https://<host>:
<port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/
<secret>/generic
```

In the preceding example, **payload.json** is the name of the minimal JSON file you have created.

Replace **<host>**, **<port>**, **<namespace>**, and **<name>** in the URL string with values that are appropriate for your environment. Replace **<secret>** with the name of the secret you have created for the webhook.

The preceding command generates the following type of webhook response in JSON format:

```
{
  "kind": "Build",
  "apiVersion": "build.openshift.io/v1",
  "metadata": {
    "name": "test-2",
    "namespace": "jfc",
    "selfLink": "/apis/build.openshift.io/v1/namespaces/jfc/buildconfigs/test-2/instantiate",
    "uid": "a72dd529-edc6-4e1c-898e-7c0dbbea176e",
    "resourceVersion": "846159",
    "creationTimestamp": "2020-10-30T12:29:30Z",
    "labels": {
      "application": "test",
      "buildconfig": "test",
      "openshift.io/build-config.name": "test",
      "openshift.io/build.start-policy": "Serial",
      "annotations": {
        "openshift.io/build-config.name": "test",
        "openshift.io/build.number": "2"
      },
      "ownerReferences": [
        {
          "apiVersion": "build.openshift.io/v1",
          "kind": "BuildConfig",
          "name": "test",
          "uid": "1f78fa3f-2f3b-421b-9f49-192184cc2280",
          "controller": true
        }
      ],
      "managedFields": [
        {
          "manager": "openshift-apiserver",
          "operation": "Update",
          "apiVersion": "build.openshift.io/v1",
          "time": "2020-10-30T12:29:30Z",
          "fieldsType": "FieldsV1",
          "fieldsV1": {
            "f:metadata": {
              "f:annotations": {
                ".": {}
              },
              "f:openshift.io/build-config.name": {},
              "f:openshift.io/build.number": {},
              "f:labels": {
                ".": {}
              },
              "f:application": {},
              "f:buildconfig": {},
              "f:openshift.io/build-config.name": {},
              "f:openshift.io/build.start-policy": {},
              "f:ownerReferences": {
                ".": {
                  "k: {\"uid\": \"1f78fa3f-2f3b-421b-9f49-192184cc2280\"}": {
                    ".": {
                      "f:apiVersion": {},
                      "f:controller": {},
                      "f:kind": {},
                      "f:name": {},
                      "f:uid": {}
                    }
                  },
                  "f:spec": {
                    "f:output": {
                      "f:to": {
                        ".": {
                          "f:kind": {},
                          "f:name": {}
                        }
                      },
                      "f:serviceAccount": {},
                      "f:source": {
                        "f:contextDir": {},
                        "f:git": {
                          ".": {
                            "f:ref": {},
                            "f:uri": {}
                          },
                          "f:type": {},
                          "f:strategy": {
                            "f:sourceStrategy": {
                              ".": {
                                "f:env": {},
                                "f:forcePull": {},
                                "f:from": {
                                  ".": {
                                    "f:kind": {},
                                    "f:name": {}
                                  }
                                },
                                "f:pullSecret": {
                                  ".": {
                                    "f:name": {},
                                    "f:type": {}
                                  },
                                  "f:triggeredBy": {},
                                  "f:status": {
                                    "f:conditions": {
                                      ".": {
                                        "k: {\"type\": \"New\"}": {
                                          ".": {
                                            "f:lastTransitionTime": {},
                                            "f:lastUpdateTime": {},
                                            "f:status": {}
                                          }
                                        },
                                        "f:type": {}
                                      }
                                    },
                                    "f:config": {
                                      ".": {
                                        "f:kind": {},
                                        "f:name": {},
                                        "f:namespace": {},
                                        "f:phase": {}
                                      }
                                    }
                                  },
                                  "spec": {
                                    "serviceAccount": "builder",
                                    "source": {
                                      "type": "Git",
                                      "git": {
                                        "uri": "https://github.com/jfclere/demo-webapp.git",
                                        "ref": "master",
                                        "contextDir": "/"
                                      },
                                      "strategy": {
                                        "type": "Source",
                                        "sourceStrategy": {
                                          "from": {
                                            "kind": "DockerImage",
                                            "name": "image-registry.openshift-image-
```

```
registry.svc:5000/jfc/jboss-webserver54-tomcat9-
openshift@sha256:75dcdf81011e113b8c8d0a40af32dc705851243baa13b6835270615417431
9e7"},"pullSecret":{"name":"builder-dockercfg-rvbh8"},"env":
[{"name":"MAVEN_MIRROR_URL"},{"name":"ARTIFACT_DIR"},"forcePull":true},"output":
{"to":{"kind":"ImageStreamTag","name":"test:latest"},"resources":{"postCommit":
{},"nodeSelector":null,"triggeredBy":{"message":"Generic WebHook","genericWebHook":
{"secret":"\u003csecret\u003e"}},"status":{"phase":"New","config":
{"kind":"BuildConfig","namespace":"jfc","name":"test"},"output":{"conditions":
[{"type":"New","status":"True","lastUpdateTime":"2020-10-
30T12:29:30Z","lastTransitionTime":"2020-10-30T12:29:30Z"}]}}
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Success",
  "message": "no git information found in payload, ignoring and continuing with build",
  "code": 200
}
```

Additional resources

- [Webhook Triggers](#)
- [JWS Operator CRD parameters](#)

CHAPTER 7. JWS OPERATOR CRD PARAMETERS

The JWS Operator provides a set of custom resource definition (CRD) parameters. When you create a custom resource **WebServer** file for a web application, you can specify parameter values in a **<key>: <value>** format. The JWS Operator uses the information that you specify in the custom resource **WebServer** file to deploy the web application.

7.1. CRD PARAMETER HIERARCHY

The JWS Operator provides CRD parameters in the following hierarchical format:

```

applicationName: <value>
replicas: <value>
useSessionClustering: <value>
webImage:
  applicationImage: <value>
  imagePullSecret: <value>
  webApp:
    name: <value>
    sourceRepositoryURL: <value>
    sourceRepositoryRef: <value>
    contextDir: <value>
    webAppWarImage: <value>
    webAppWarImagePushSecret: <value>
    builder:
      image: <value>
      imagePullSecret: <value>
      applicationBuildScript: <value>
  webServerHealthCheck:
    serverReadinessScript: <value>
    serverLivenessScript: <value>
webImageStream:
  imageStreamName: <value>
  imageStreamNamespace: <value>
webSources:
  sourceRepositoryUrl: <value>
  sourceRepositoryRef: <value>
  contextDir: <value>
  webSourcesParams:
    mavenMirrorUrl: <value>
    artifactDir: <value>
    genericWebHookSecret: <value>
    githubWebHookSecret: <value>
webServerHealthCheck:
  serverReadinessScript: <value>
  serverLivenessScript: <value>

```




NOTE

When you create a custom resource **WebServer** file, specify parameter names and values in the same hierarchical format that the preceding example outlines. For more information about creating a custom resource **WebServer** file, see [Deploying an existing JWS image](#).

7.2. CRD PARAMETER DETAILS

The following table describes the CRD parameters that the JWS Operator provides. This table shows each parameter name in the context of any higher-level parameters that are above it in the hierarchy.

Parameter name	Description
replicas	<p>The number of pods of the JBoss Web Server image that you want to run</p> <p>For example: replicas: 2</p>
applicationName	<p>The name of the web application that you want the JWS Operator to deploy</p> <p>The application name must be a unique value in the OpenShift namespace or project. The JWS Operator uses the application name that you specify to create the route to access the web application.</p> <p>For example: applicationName: my-app</p>
useSessionClustering	<p>Enables DNSping session clustering</p> <p>This is set to false by default. If you set this parameter to true, the image must be based on JBoss Web Server images, because session clustering uses the ENV_FILES environment variable and a shell script to add the clustering in the server.xml file.</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>NOTE</p> <p>In this release, the session clustering functionality is available as a Technology Preview feature only. The current Operator version uses the DNS Membership Provider, which is limited because of DNS limitations. InetAddress.getAllByName() results are cached, which means session replications might not work while scaling up.</p> </div> </div> <p>For example: useSessionClustering: true</p>

Parameter name	Description
webImage	<p>A set of parameters that controls how the JWS Operator deploys pods from existing images</p> <p>This parameter contains applicationImage, imagePullSecret, webApp, and webServerHealthCheck fields.</p>
webImage: applicationImage	<p>The full path to the name of the application image that you want to deploy</p> <p>For example: applicationImage: quay.io/\$user/my-image-name</p>
webImage: imagePullSecret	<p>The name of the secret that the JWS Operator uses to pull images from the repository</p> <p>The secret must contain the key .dockerconfigjson. The JWS Operator mounts the secret and uses it similar to --authfile /mount_point/.dockerconfigjson to pull the images from the repository.</p> <p>The Secret object definition file might contain several username and password values or tokens to allow access to images in the image stream, the builder image, and images built by the JWS Operator.</p> <p>For example: imagePullSecret: mysecret</p>
webImage: webApp	<p>A set of parameters that describe how the JWS Operator builds the web application that you want to add to the application image</p> <p>If you do not specify the webApp parameter, the JWS Operator deploys the web application without building the application.</p> <p>This parameter contains name, sourceRepositoryURL, sourceRepositoryRef, contextDir, webAppWarImage, webAppWarImagePushSecret, and builder fields.</p>


Parameter name	Description
webImage: webApp: name	<p>The name of the web application file</p> <p>The default name is ROOT.war.</p> <p>For example: name: my-app.war</p>
webImage: webApp: sourceRepositoryURL	<p>The URL where the application source files are located</p> <p>The source should contain a Maven pom.xml file to support a Maven build. When Maven generates a .war file for the application, the .war file is copied to the webapps directory of the image that the JWS Operator uses to deploy the application (for example, /opt/jws-5.x/tomcat/webapps).</p> <p>For example: sourceRepositoryUrl: 'https://github.com/\$user/demo-webapp.git'</p>
webImage: webApp: sourceRepositoryRef	<p>The branch of the source repository that the JWS Operator uses</p> <p>For example: sourceRepositoryRef: main</p>
webImage: webApp: contextDir	<p>The subdirectory in the source repository where the pom.xml file is located and the mvn install command is run</p> <p>For example: contextDir: /</p>
webImage: webApp: webAppWarImage	<p>The URL of the images where the JWS Operator pushes the built image</p>

Parameter name	Description
webImage: webApp: webAppWarImagePushSecret	<p>The name of the secret that the JWS Operator uses to push images to the repository</p> <p>The secret must contain the key .dockerconfigjson. The JWS Operator mounts the secret and uses it similar to --authfile /mount_point/.dockerconfigjson to push the image to the repository.</p> <p>If the JWS Operator uses a pull secret to pull images from the repository, you must specify the name of the pull secret as the value for the webAppWarImagePushSecret parameter. See imagePullSecret for more information.</p> <p>For example: imagePullSecret: mysecret</p>
webImage: webApp: builder	<p>A set of parameters that describe how the JWS Operator builds the web application and creates and pushes the image to the image repository</p> <div data-bbox="815 1010 922 1420" style="background: repeating-linear-gradient(45deg, transparent, transparent 2px, #ccc 2px, #ccc 4px); width: 67px; height: 183px; display: inline-block;"></div> <p>NOTE</p> <p>To ensure that the builder can operate successfully and run commands with different user IDs, the builder must have access to the anyuid SCC (security context constraint). To grant the builder access to the anyuid SCC, enter the following command:</p> <p>oc adm policy add-scc-to-user anyuid -z builder</p> <p>This parameter contains image, imagePullSecret, and applicationBuildScript fields.</p>
webImage: webApp: builder: image	<p>The image of the container where the JWS Operator builds the web application</p> <p>For example: image: quay.io/\$user/tomcat10-buildah</p>

Parameter name	Description
webImage: webApp: builder: imagePullSecret	<p>The name of the secret (if specified) that the JWS Operator uses to pull the builder image from the repository</p> <p>The secret must contain the key .dockerconfigjson. The JWS Operator mounts the secret and uses it similar to --authfile /mount_point/.dockerconfigjson to pull the images from the repository.</p> <p>The Secret object definition file might contain several username and password values or tokens to allow access to images in the image stream, the builder image, and images built by the JWS Operator.</p> <p>For example: imagePullSecret: mysecret</p>
webImage: webApp: builder: applicationBuildScript	<p>The script that the builder image uses to build the application .war file and move it to the /mnt directory</p> <p>If you do not specify a value for this parameter, the builder image uses a default script that uses Maven and Buildah.</p>
webImage: webServerHealthCheck	<p>The health check that the JWS Operator uses</p> <p>The default behavior is to use the health valve, which does not require any parameters.</p> <p>This parameter contains serverReadinessScript and serverLivenessScript fields.</p>
webImage: webServerHealthCheck: serverReadinessScript	<p>A string that specifies the logic for the pod readiness health check</p> <p>If this parameter is not specified, the JWS Operator uses the default health check by using the OpenShift internal registry to check http://localhost:8080/health.</p> <p>For example: serverReadinessScript: /bin/bash -c " /usr/bin/curl --noproxy '*' -s 'http://localhost:8080/health' /usr/bin/grep -i 'status.*UP'"</p>

Parameter name	Description
webImage: webServerHealthCheck: serverLivenessScript	<p>A string that specifies the logic for the pod liveness health check</p> <p>This parameter is optional.</p>
webImageStream	<p>A set of parameters that control how the JWS Operator uses an image stream that provides images to run or to build upon</p> <p>The JWS Operator uses the latest image in the image stream.</p> <p>This parameter contains applicationImage, imagePullSecret, webApp, and webServerHealthCheck fields.</p>
webImageStream: imageStreamName	<p>The name of the image stream that you have created to allow the JWS Operator to find the base images</p> <p>For example: imageStreamName: my-image-name-imagestream:latest</p>
webImageStream: imageStreamNamespace	<p>The namespace or project where you have created the image stream</p> <p>For example: imageStreamNamespace: my-namespace</p>
webImageStream: webSources	<p>A set of parameters that describe where the application source files are located and how to build them</p> <p>If you do not specify the webSources parameter, the JWS Operator deploys the latest image in the image stream.</p> <p>This parameter contains sourceRepositoryUrl, sourceRepositoryRef, contextDir, and webSourcesParams fields.</p>

Parameter name	Description
webImageStream: webSources: sourceRepositoryUrl	<p>The URL where the application source files are located</p> <p>The source should contain a Maven pom.xml file to support a Maven build. When Maven generates a .war file for the application, the .war file is copied to the webapps directory of the image that the JWS Operator uses to deploy the application (for example, /opt/jws-5.x/tomcat/webapps).</p> <p>For example: sourceRepositoryUrl: 'https://github.com/\$user/demo-webapp.git'</p>
webImageStream: webSources: sourceRepositoryRef	<p>The branch of the source repository that the JWS Operator uses</p> <p>For example: sourceRepositoryRef: main</p>
webImageStream: webSources: contextDir	<p>The subdirectory in the source repository where the pom.xml file is located and the mvn install command is run</p> <p>For example: contextDir: /</p>
webImageStream: webSources: webSourcesParams	<p>A set of parameters that describe how to build the application images</p> <p>This parameter is optional.</p> <p>This parameter contains mavenMirrorUrl, artifactDir, genericWebHookSecret, and githubWebHookSecret fields.</p>
webImageStream: webSources: webSourcesParams: mavenMirrorUrl	<p>The Maven proxy URL that Maven uses to build the web application</p> <p>This parameter is required if the cluster does not have internet access.</p>

Parameter name	Description
webImageStream: webSources: webSourcesParams: artifactDir	<p>The directory where Maven stores the .war file that Maven generates for the web application</p> <p>The contents of this directory are copied to the webapps directory of the image that the JWS Operator uses to deploy the application (for example, /opt/jws-5.x/tomcat/webapps).</p> <p>The default value is target.</p>
webImageStream: webSources: webSourcesParams: genericWebHookSecret	<p>The name of a secret for a generic webhook that can trigger a build</p> <p>For more information about creating a secret, see Creating a secret for a generic or GitHub webhook</p> <p>For more information about using generic webhooks, see Webhook Triggers.</p> <p>For example: genericWebHookSecret: my-secret</p>
webImageStream: webSources: webSourcesParams: githubWebHookSecret	<p>The name of a secret for a GitHub webhook that can trigger a build</p> <p>For more information about creating a secret, see Creating a secret for a generic or GitHub webhook</p> <p>For more information about using GitHub webhooks, see Webhook Triggers.</p> <div data-bbox="815 1323 922 1518" style="display: inline-block; vertical-align: top;">  </div> <p>NOTE</p> <p>You cannot perform manual tests of a GitHub webhook. GitHub generates the payload and it is not empty.</p>
webImageStream: webServerHealthCheck	<p>The health check that the JWS Operator uses</p> <p>The default behavior is to use the health valve, which does not require any parameters.</p> <p>This parameter contains serverReadinessScript and serverLivenessScript fields.</p>

Parameter name	Description
webImageStream: webServerHealthCheck: serverReadinessScript	<p>A string that specifies the logic for the pod readiness health check</p> <p>If this parameter is not specified, the JWS Operator uses the default health check by using the OpenShift internal registry to check http://localhost:8080/health.</p> <p>For example: serverReadinessScript: /bin/bash -c " /usr/bin/curl --noproxy '*' -s 'http://localhost:8080/health' /usr/bin/grep -i 'status.*UP'"</p>
webImageStream: webServerHealthCheck: serverLivenessScript	<p>A string that specifies the logic for the pod liveness health check</p> <p>This parameter is optional.</p>