



# Red Hat JBoss Enterprise Application Platform 7.4

## Getting Started with JBoss EAP for OpenShift Online

Guide to developing with Red Hat JBoss Enterprise Application Platform for  
OpenShift Online



# Red Hat JBoss Enterprise Application Platform 7.4 Getting Started with JBoss EAP for OpenShift Online

---

Guide to developing with Red Hat JBoss Enterprise Application Platform for OpenShift Online

## 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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

Guide to using Red Hat JBoss Enterprise Application Platform for OpenShift Online

## Table of Contents

<b>PROVIDING FEEDBACK ON JBOSS EAP DOCUMENTATION</b> .....	<b>5</b>
<b>MAKING OPEN SOURCE MORE INCLUSIVE</b> .....	<b>6</b>
<b>CHAPTER 1. INTRODUCTION</b> .....	<b>7</b>
1.1. WHAT IS RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (JBOSS EAP)?	7
1.2. HOW DOES JBOSS EAP WORK ON OPENSIFT?	7
1.3. COMPARISON: JBOSS EAP AND JBOSS EAP FOR OPENSIFT	7
1.4. VERSION COMPATIBILITY AND SUPPORT	8
Supported architectures for OpenJDK images	9
1.4.1. OpenShift 4.x Support	9
1.4.2. IBM Z Support	9
1.4.3. Upgrades from JBoss EAP 7.1 to JBoss EAP 7.4 on OpenShift	10
1.5. DEPLOYMENT OPTIONS	10
<b>CHAPTER 2. BUILD AND RUN A JAVA APPLICATION ON THE JBOSS EAP FOR OPENSIFT IMAGE</b> .....	<b>11</b>
2.1. PREREQUISITES	11
2.2. PREPARE OPENSIFT FOR APPLICATION DEPLOYMENT	11
2.3. IMPORT THE LATEST JBOSS EAP FOR OPENSIFT IMAGESTREAMS AND TEMPLATES	12
Import command for JDK 8	12
Import command for JDK 11	13
Import command for templates	13
2.4. DEPLOY A JBOSS EAP SOURCE-TO-IMAGE (S2I) APPLICATION TO OPENSIFT	14
2.5. POST DEPLOYMENT TASKS	16
2.6. CHAINED BUILD SUPPORT IN JBOSS EAP FOR OPENSIFT	17
<b>CHAPTER 3. DEPLOYING JBOSS EAP 7 APPLICATION ON OPENSIFT USING HELM CHART</b> .....	<b>18</b>
3.1. PREREQUISITES	18
3.2. CREATING A JBOSS EAP EAP 7 APPLICATION WITH HELM	18
3.3. VIEWING THE HELM RELEASE	19
3.4. VIEWING THE ASSOCIATED CODE	20
3.5. VIEWING THE BUILD STATUS	20
3.6. VIEWING THE POD STATUS	21
3.7. RUNNING THE JBOSS EAP 7 APPLICATION	22
<b>CHAPTER 4. CONFIGURING THE JBOSS EAP FOR OPENSIFT IMAGE FOR YOUR JAVA APPLICATION</b>	<b>23</b>
4.1. HOW THE JBOSS EAP FOR OPENSIFT S2I PROCESS WORKS	23
4.2. CONFIGURING JBOSS EAP FOR OPENSIFT USING ENVIRONMENT VARIABLES	24
4.2.1. JVM Memory Configuration	25
4.2.1.1. JVM Default Memory Settings	25
4.2.1.2. JVM Garbage Collection Settings	25
4.2.1.3. Resource Limits in Default Settings	26
4.2.1.4. JVM Environment Variables	26
4.3. BUILD EXTENSIONS AND PROJECT ARTIFACTS	33
4.3.1. S2I Artifacts	34
4.3.1.1. Modules, Drivers, and Generic Deployments	35
4.3.2. Runtime Artifacts	37
4.3.2.1. Datasources	37
4.3.2.2. Resource Adapters	38
4.4. RESULTS OF USING JBOSS EAP TEMPLATES FOR OPENSIFT	39
4.5. SSO CONFIGURATION OF RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM FOR OPENSIFT IMAGES	40
4.6. DEFAULT DATASOURCE	40

<b>CHAPTER 5. CAPABILITY TRIMMING IN JBOSS EAP FOR OPENSIFT .....</b>	<b>41</b>
5.1. PROVISION A CUSTOM JBOSS EAP SERVER	41
5.2. AVAILABLE JBOSS EAP LAYERS	41
5.2.1. Base Layers	41
datasources-web-server	41
jaxrs-server	42
cloud-server	43
5.2.2. Decorator Layers	43
sso	43
observability	43
web-clustering	43
5.3. PROVISIONING USER-DEVELOPED LAYERS IN JBOSS EAP	44
5.3.1. Building and using custom Galleon layers for JBoss EAP	44
5.3.1.1. Preparing the Maven project	44
5.3.1.2. Adding the feature pack content	45
5.3.1.3. Using the custom Galleon feature-pack during S2I build	47
5.3.1.4. Custom Provisioning Files for JBoss EAP	50
5.3.2. Configure Galleon by using advanced environment variables	51
5.3.3. Custom Galleon feature pack environment variables	51
<b>CHAPTER 6. TROUBLESHOOTING .....</b>	<b>53</b>
6.1. TROUBLESHOOTING POD RESTARTS	53
6.2. TROUBLESHOOTING USING THE JBOSS EAP MANAGEMENT CLI	53
<b>CHAPTER 7. EAP OPERATOR FOR AUTOMATING APPLICATION DEPLOYMENT ON OPENSIFT .....</b>	<b>55</b>
7.1. INSTALLING EAP OPERATOR USING THE WEB CONSOLE	55
7.2. INSTALLING EAP OPERATOR USING THE CLI	57
7.3. THE EAP-S2I-BUILD TEMPLATE FOR CREATING APPLICATION IMAGES	58
7.4. BUILDING AN APPLICATION IMAGE USING EAP-S2I-BUILD TEMPLATE	58
7.5. DEPLOYING A JAVA APPLICATION ON OPENSIFT USING THE EAP OPERATOR	60
7.5.1. Creating a Secret	63
7.5.2. Creating a ConfigMap	63
7.5.3. Creating a ConfigMap from a standalone.xml File	64
7.5.4. Configuring Persistent Storage for Applications	64
7.6. DEPLOYING THE RED HAT SINGLE SIGN-ON-ENABLED IMAGE BY USING EAP OPERATOR	64
7.7. VIEWING METRICS OF AN APPLICATION USING THE EAP OPERATOR	66
7.8. UNINSTALLING EAP OPERATOR USING WEB CONSOLE	66
7.9. UNINSTALLING EAP OPERATOR USING THE CLI	67
7.10. EAP OPERATOR FOR SAFE TRANSACTION RECOVERY	68
7.10.1. StatefulSets for Stable Network Host Names	69
7.10.2. Monitoring the Scaledown Process	69
7.10.2.1. Pod Status During Scaledown	70
7.10.3. Scaling Down During Transactions with Heuristic Outcomes	70
7.10.4. Configuring the transactions subsystem to use the JDBC storage for transaction log	71
7.11. AUTOMATICALLY SCALING PODS WITH THE HORIZONTAL POD AUTOSCALER HPA	72
7.12. JAKARTA ENTERPRISE BEANS REMOTING ON OPENSIFT	73
7.12.1. Configuring Jakarta Enterprise Beans on OpenShift	74
<b>CHAPTER 8. REFERENCE INFORMATION .....</b>	<b>76</b>
8.1. PERSISTENT TEMPLATES	76
8.2. INFORMATION ENVIRONMENT VARIABLES	76
8.3. CONFIGURATION ENVIRONMENT VARIABLES	77
8.4. APPLICATION TEMPLATES	83
8.5. EXPOSED PORTS	83

---

8.6. DATASOURCES	83
8.6.1. JNDI Mappings for Datasources	83
8.6.1.1. Datasource Configuration Environment Variables	84
8.6.1.2. Examples	86
8.6.1.2.1. Single Mapping	86
8.6.1.2.2. Multiple Mappings	87
8.7. CLUSTERING	87
8.7.1. Configuring a JGroups Discovery Mechanism	87
8.7.1.1. Configuring KUBE_PING	87
8.7.1.2. Configuring DNS_PING	88
8.7.2. Configuring JGroups to Encrypt Cluster Traffic	89
8.7.2.1. Configuring SYM_ENCRYPT	90
8.7.2.2. Configuring ASYM_ENCRYPT	91
8.7.3. Considerations for scaling up pods	91
8.8. HEALTH CHECKS	91
8.9. MESSAGING	92
8.9.1. Configuring External Red Hat AMQ Brokers	92
Example OpenShift Application Definition	92
8.10. SECURITY DOMAINS	92
8.11. HTTPS ENVIRONMENT VARIABLES	93
8.12. ADMINISTRATION ENVIRONMENT VARIABLES	93
8.13. S2I	94
8.13.1. Custom Configuration	94
8.13.1.1. Custom Modules	94
8.13.2. Deployment Artifacts	94
8.13.3. Artifact Repository Mirrors	94
8.13.3.1. Secure Artifact Repository Mirror URLs	95
8.13.4. Scripts	95
8.13.5. Custom Scripts	95
8.13.5.1. Mounting a configmap to execute custom scripts	96
8.13.5.2. Using install.sh to execute custom scripts	96
8.13.6. Environment Variables	97
8.14. SINGLE SIGN-ON IMAGE	99
8.15. UNSUPPORTED TRANSACTION RECOVERY SCENARIOS	100
8.16. INCLUDED JBOSS MODULES	101
8.17. EAP OPERATOR: API INFORMATION	101
8.17.1. WildFlyServer	101
8.17.2. WildFlyServerList	101
8.17.3. WildFlyServerSpec	102
8.17.4. Resources	104
8.17.5. StorageSpec	104
8.17.6. StandaloneConfigMapSpec	104
8.17.7. WildFlyServerStatus	105
8.17.8. PodStatus	105





## PROVIDING FEEDBACK ON JBOSS EAP 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. Please include the **Document URL**, the **section number** and **describe the issue**.
3. Enter a brief description of the issue in the **Summary**.
4. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
5. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

## MAKING OPEN SOURCE MORE INCLUSIVE

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

# CHAPTER 1. INTRODUCTION

## 1.1. WHAT IS RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (JBOSS EAP)?

Red Hat JBoss Enterprise Application Platform 7.4 (JBoss EAP) is a middleware platform built on open standards and compliant with the Jakarta EE 8 specification. It provides preconfigured options for features such as high-availability clustering, messaging, and distributed caching. It includes a modular structure that allows you to enable services only when required, which results in improved startup speed.

The web-based management console and management command line interface (CLI) make editing XML configuration files unnecessary and add the ability to script and automate tasks. In addition, JBoss EAP includes APIs and development frameworks that allow you to quickly develop, deploy, and run secure and scalable Jakarta EE applications. JBoss EAP 7.4 is a Jakarta EE 8 compatible implementation for both Web Profile and Full Platform specifications.

## 1.2. HOW DOES JBOSS EAP WORK ON OPENSHIFT?

Red Hat offers a containerized image for JBoss EAP that is designed for use with OpenShift. Using this image, developers can quickly and easily build, scale, and test applications that are deployed across hybrid environments.

## 1.3. COMPARISON: JBOSS EAP AND JBOSS EAP FOR OPENSHIFT

There are some notable differences when comparing the JBoss EAP product with the JBoss EAP for OpenShift image. The following table describes these differences and notes which features are included or supported in the current version of JBoss EAP for OpenShift.

**Table 1.1. Differences between JBoss EAP and JBoss EAP for OpenShift**

JBoss EAP Feature	Status in JBoss EAP for OpenShift	Description
JBoss EAP management console	Not included	The JBoss EAP management console is not included in this release of JBoss EAP for OpenShift.
JBoss EAP management CLI	Not recommended	The JBoss EAP management CLI is not recommended for use with JBoss EAP running in a containerized environment. Any configuration changes made using the management CLI in a running container will be lost when the container restarts. <a href="#">The management CLI is accessible from within a pod for troubleshooting purposes.</a>
Managed domain	Not supported	Although a JBoss EAP managed domain is not supported, creation and distribution of applications are managed in the containers on OpenShift.
Default root page	Disabled	The default root page is disabled, but you can deploy your own application to the root context as <b>ROOT.war</b> .

JBoss EAP Feature	Status in JBoss EAP for OpenShift	Description
Remote messaging	Supported	Red Hat AMQ for inter-pod and remote messaging is supported. ActiveMQ Artemis is only supported for messaging within a single pod with JBoss EAP instances, and is only enabled when Red Hat AMQ is absent.
Transaction recovery	Partially supported	<p>The EAP operator is the only tested and supported option of transaction recovery in OpenShift 4. For more information about recovering transactions using the EAP operator, see <a href="#">EAP Operator for Safe Transaction Recovery</a>.</p> <p>Some scenarios are not supported. For more information about the unsupported scenarios, see <a href="#">Unsupported Transaction Recovery Scenarios</a>.</p>
Embedded messaging broker	Deprecated	<p>The use of an embedded messaging broker in OpenShift containers is deprecated. Support for an embedded broker will be removed in a future release.</p> <p>If a container is configured to use an embedded messaging broker, and if no remote broker is configured, a warning is logged.</p> <p>If the container configuration does not include messaging destinations, set the <b>DISABLE_EMBEDDED_JMS_BROKER</b> environment variable to <b>true</b> to disable the ability to configure an embedded messaging broker.</p>

## 1.4. VERSION COMPATIBILITY AND SUPPORT

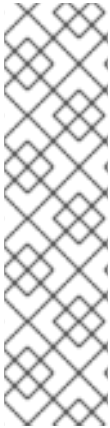
JBoss EAP for OpenShift provides images for OpenJDK 8 and OpenJDK 11.

Two variants of each image are available: an S2I builder image and a runtime image. The S2I builder image contains a complete JBoss EAP server with tooling needed during S2I build. The runtime image contains dependencies needed to run JBoss EAP but does not contain a server. The server is installed in the runtime image during a chained build.

The following modifications were applied to images in JBoss EAP 7.4 for OpenShift.

- Default drivers and modules are removed.
- Templates for MySQL and PostgreSQL are removed. You can provision these capabilities with a custom layer.
- The Hawkular agent is not active in these images. If configured, it is ignored.
- The datasource, **ExampleDS**, is no longer added by default at container startup. If you need the default datasource, use the environment variable

**ENABLE\_GENERATE\_DEFAULT\_DATASOURCE** with a value of **true** (**ENABLE\_GENERATE\_DEFAULT\_DATASOURCE=true**) to include it.



## NOTE

The following discovery mechanism protocols were deprecated and are replaced by other protocols:

- The **openshift.DNS\_PING** protocol was deprecated and is replaced with the **dns.DNS\_PING** protocol. If you referenced the **openshift.DNS\_PING** protocol in a **customized standalone-openshift.xml** file, replace the protocol with the **dns.DNS\_PING** protocol.
- The **openshift.KUBE\_PING** discovery mechanism protocol was deprecated and is replaced with the **kubernetes.KUBE\_PING** protocol.

## Supported architectures for OpenJDK images

OpenJDK images support several architectures. This information is summarized in the following table:

### 1. OpenJDK images and architectures

JDK (OS)	Architecture supported	Red Hat Ecosystem Catalog
OpenJDK8 (RHEL 7)	x86_64	<a href="#">Builder image</a> and <a href="#">Runtime image</a>
OpenJDK11 (RHEL 8)	x86_64, IBM Z, and IBM Power Systems	<a href="#">Builder image</a> and <a href="#">Runtime image</a>

JBoss EAP for OpenShift is updated frequently. Therefore, it is important to understand which versions of the images are compatible with which versions of OpenShift.

### Additional resources

- [OpenShift and Atomic Platform Tested Integrations](#)
- [Capability Trimming in JBoss EAP for OpenShift](#)

### 1.4.1. OpenShift 4.x Support

Changes in OpenShift 4.1 affect access to Jolokia, and the Open Java Console is no longer available in the OpenShift 4.x web console.

In previous releases of OpenShift, certain kube-apiserver proxied requests were authenticated and passed through to the cluster. This behavior is now considered insecure, and so, accessing Jolokia in this manner is no longer supported.

Due to changes in codebase for the OpenShift console, the link to the Open Java Console is no longer available.

### 1.4.2. IBM Z Support

The s390x variant of **libartemis-native** is not included in the image. Thus, any settings related to AIO will not be taken into account.

- **journal-type**: Setting the **journal-type** to **ASYNCIO** has no effect. The value of this attribute defaults to **NIO** at runtime.
- **journal-max-io**: This attribute has no effect.
- **journal-store-enable-async-io**: This attribute has no effect.

### 1.4.3. Upgrades from JBoss EAP 7.1 to JBoss EAP 7.4 on OpenShift

The file **standalone-openshift.xml** installed with JBoss EAP 7.1 on OpenShift is not compatible with JBoss EAP 7.4 and later. You must modify a **standalone-openshift.xml** file installed with JBoss EAP 7.1 before you use it to start a JBoss EAP 7.4 or later container for OpenShift.

#### Additional resources

Updates to [standalone-openshift.xml](#) when upgrading JBoss EAP 7.1 to JBoss EAP 7.4 on OpenShift

## 1.5. DEPLOYMENT OPTIONS

You can deploy the JBoss EAP Java applications on OpenShift using one of the following options:

- A JBoss EAP for OpenShift template.
- The EAP operator, a JBoss EAP-specific controller that extends the OpenShift API to create, configure, and manage instances of complex stateful applications on behalf of an OpenShift user.



#### NOTE

The EAP operator is supported only on OpenShift 4 and later versions.

#### Additional resources

- For more information about JBoss EAP for OpenShift templates, see [jboss-eap-openshift-templates](#) on *Git Hub*.
- For more information about the EAP operator, see [EAP Operator for Automating Application Deployment on OpenShift](#).

## CHAPTER 2. BUILD AND RUN A JAVA APPLICATION ON THE JBOSS EAP FOR OPENSIFT IMAGE

The following workflow demonstrates using the Source-to-Image (S2I) process to build and run a Java application on the JBoss EAP for OpenShift image.

As an example, the **kitchensink** quickstart is used in this procedure. It demonstrates a Jakarta EE web-enabled database application using Jakarta Server Faces, Jakarta Contexts and Dependency Injection, Jakarta Enterprise Beans, Jakarta Persistence, and Jakarta Bean Validation. See the **kitchensink** quickstart that ships with JBoss EAP 7 for more information.

### 2.1. PREREQUISITES

This workflow assumes that you already have an active [OpenShift Online](#) subscription and that you have installed the [OpenShift CLI](#).

### 2.2. PREPARE OPENSIFT FOR APPLICATION DEPLOYMENT

1. Log in to your OpenShift instance using the **oc login** command.
2. Create a new project in OpenShift.  
A project allows a group of users to organize and manage content separately from other groups. You can create a project in OpenShift using the following command.

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

For example, for the **kitchensink** quickstart, create a new project named **eap-demo** using the following command.

```
$ oc new-project eap-demo
```

3. **Optional:** Create a keystore and a secret.



#### NOTE

Creating a keystore and a secret is required if you are using any HTTPS-enabled features in your OpenShift project. For example, if you are using the **eap74-https-s2i** template, you must create a keystore and secret.

This workflow demonstration for the **kitchensink** quickstart does not use an HTTPS template, so a keystore and secret are not required.

- a. Create a keystore.

**WARNING**

The following commands generate a self-signed certificate, but for production environments Red Hat recommends that you use your own SSL certificate purchased from a verified Certificate Authority (CA) for SSL-encrypted connections (HTTPS).

You can use the Java **keytool** command to generate a keystore:

```
$ keytool -genkey -keyalg RSA -alias <alias_name> -keystore <keystore_filename.jks> -
validity 360 -keysize 2048
```

For example, for the **kitchensink** quickstart, use the following command to generate a keystore:

```
$ keytool -genkey -keyalg RSA -alias eapdemo-selfsigned -keystore keystore.jks -validity
360 -keysize 2048
```

- b. Create a secret from the keystore.  
Create a secret from the previously created keystore using the following command.

```
$ oc create secret generic <secret_name> --from-file=<keystore_filename.jks>
```

For example, for the **kitchensink** quickstart, use the following command to create a secret.

```
$ oc create secret generic eap7-app-secret --from-file=keystore.jks
```

## 2.3. IMPORT THE LATEST JBOSS EAP FOR OPENSIFT IMAGESTREAMS AND TEMPLATES

You must import the latest JBoss EAP for OpenShift imagestreams 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 JBoss EAP imagestreams and templates. For more information, see [Red Hat Container Registry Authentication](#).

### Import command for JDK 8

```
oc replace -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-
templates/eap74/eap74-openjdk8-image-stream.json
```

This command imports the following imagestreams and templates.



- The JDK 8 builder imagestream: **jboss-eap74-openjdk8-openshift**
- The JDK 8 runtime imagestream: **jboss-eap74-openjdk8-runtime-openshift**



## NOTE

If you use OpenShift 3 and create an EAP 7.4 ImageStream for the first time, run the following command instead of **oc replace**:

```
oc create -f https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-templates/eap74/eap74-openjdk8-image-stream.json
```

## Import command for JDK 11

```
oc replace -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-templates/eap74/eap74-openjdk11-image-stream.json
```

This command imports the following imagestreams and templates.

- The JDK 11 builder imagestream: **jboss-eap74-openjdk11-openshift**
- The JDK 11 runtime imagestream: **jboss-eap74-openjdk11-runtime-openshift**

## Import command for templates

```
for resource in \
  eap74-amq-persistent-s2i.json \
  eap74-amq-s2i.json \
  eap74-basic-s2i.json \
  eap74-https-s2i.json \
  eap74-sso-s2i.json
do
  oc replace -f \
  https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-templates/eap74/templates/${resource}
done
```

This command imports all templates specified in the command.



## NOTE

The JBoss EAP imagestreams and templates imported using these commands are only available within that OpenShift project.

If you want to import the image streams and templates into a different project, add the **-n PROJECT\_NAME** to the **oc replace** line of the command. For example:

```
...
oc replace -n PROJECT_NAME --force -f
...
```

If you use the `cluster-samples-operator`, refer to the OpenShift documentation on configuring the cluster samples operator. See [Configuring the Samples Operator](#) for details about configuring the cluster samples operator.

## 2.4. DEPLOY A JBOSS EAP SOURCE-TO-IMAGE (S2I) APPLICATION TO OPENSIFT

After you import the images and templates, you can deploy applications to OpenShift.

### Prerequisites

**Optional:** A template can specify default values for many template parameters, and you might have to override some, or all, of the defaults. To see template information, including a list of parameters and any default values, use the command `oc describe template TEMPLATE_NAME`.

### Procedure

1. Create a new OpenShift application that uses the JBoss EAP for OpenShift image and the source code of your Java application. You can use one of the provided JBoss EAP for OpenShift templates for S2I builds. You can also choose to provision a trimmed server. For example, to deploy the `kitchensink` quickstart using the JDK 8 builder image, enter the following command to use the `eap74-basic-s2i` template in the `eap-demo` project, created in [Prepare OpenShift for Application Deployment](#), with the `kitchensink` source code on GitHub. This quickstart does not support the trimming capability.

```
oc new-app --template=eap74-basic-s2i \ 1
-p IMAGE_STREAM_NAMESPACE=eap-demo \ 2
-p EAP_IMAGE_NAME=jboss-eap74-openjdk8-openshift:7.4.0 \ 3
-p EAP_RUNTIME_IMAGE_NAME=jboss-eap74-openjdk8-runtime-openshift:7.4.0 \ 4
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \ 5
-p SOURCE_REPOSITORY_REF=7.4.x \ 6
-p CONTEXT_DIR=kitchensink 7
```

- 1 The template to use.
- 2 The latest imagestreams and templates [were imported into the project's namespace](#), so you must specify the namespace where to find the imagestream. This is usually the project's name.
- 3 The name of the EAP builder image stream for JDK8.
- 4 The name of the EAP runtime image stream for JDK8.
- 5 URL to the repository containing the application source code.
- 6 The Git repository reference to use for the source code. This can be a Git branch or tag reference.
- 7 The directory within the source repository to build.

As another example, to deploy the `helloworld-html5` quickstart using the JDK 11 runtime image and trimming JBoss EAP to include only the `jaxrs-server` layer, enter the following command. The command uses the `eap74-basic-s2i` template in the `eap-demo` project, created in [Prepare](#)

OpenShift for Application Deployment, with the **helloworld-html5** source code on GitHub.

```
oc new-app --template=eap74-basic-s2i \ 1
-p IMAGE_STREAM_NAMESPACE=eap-demo \ 2
-p EAP_IMAGE_NAME=jboss-eap74-openjdk11-openshift:7.4.0 \ 3
-p EAP_RUNTIME_IMAGE_NAME=jboss-eap74-openjdk11-runtime-openshift:7.4.0 \ 4
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \ 5
-p SOURCE_REPOSITORY_REF=7.4.x \ 6
-p GALLEON_PROVISION_LAYERS=jaxrs-server \ 7
-p CONTEXT_DIR=helloworld-html5 8
```

- 1 The template to use.
- 2 The latest imagestreams and templates [were imported into the project's namespace](#), so you must specify the namespace where to find the imagestream. This is usually the project's name.
- 3 The name of the EAP builder image stream for JDK11.
- 4 The name of the EAP runtime image stream for JDK11.
- 5 URL to the repository containing the application source code.
- 6 The Git repository reference to use for the source code. This can be a Git branch or tag reference.
- 7 Provision a trimmed server with only the **jaxrs-server** layer.
- 8 The directory within the source repository to build.



#### NOTE

You might also want to [configure environment variables](#) when creating your new OpenShift application.

For example, if you are using an HTTPS template such as **eap74-https-s2i**, you must specify the required [HTTPS environment variables](#) **HTTPS\_NAME**, **HTTPS\_PASSWORD**, and **HTTPS\_KEYSTORE** to match your keystore details.



#### NOTE

If the template uses AMQ, you must include the **AMQ\_IMAGE\_NAME** parameter with the appropriate value.

If the template uses SSO, you must include the **SSO\_IMAGE\_NAME** parameter with the appropriate value.

2. Retrieve the name of the build configuration.

```
$ oc get bc -o name
```

- Use the name of the build configuration from the previous step to view the Maven progress of the build.

```
$ oc logs -f buildconfig/BUILD_CONFIG_NAME
```

For example, for the **kitchensink** quickstart, the following command shows the progress of the Maven build.

```
$ oc logs -f buildconfig/eap-app
```

## Additional Resources

[Capability Trimming in JBoss EAP for OpenShift](#)

## 2.5. POST DEPLOYMENT TASKS

Depending on your application, some tasks might need to be performed after your OpenShift application has been built and deployed. This might include exposing a service so that the application is viewable from outside of OpenShift, or scaling your application to a specific number of replicas.

- Get the service name of your application using the following command.

```
$ oc get service
```

- Expose the main service as a route so you can access your application from outside of OpenShift. For example, for the **kitchensink** quickstart, use the following command to expose the required service and port.

```
$ oc expose service/eap-app --port=8080
```



### NOTE

If you used a template to create the application, the route might already exist. If it does, continue on to the next step.

- Get the URL of the route.

```
$ oc get route
```

- Access the application in your web browser using the URL. The URL is the value of the **HOST/PORT** field from the previous command's output.

If your application does not use the JBoss EAP root context, append the context of the application to the URL. For example, for the **kitchensink** quickstart, the URL might be **http://*HOST\_PORT\_VALUE*/kitchensink/**.

- Optionally, you can also scale up the application instance by running the following command. This increases the number of replicas to **3**.

```
$ oc scale deploymentconfig DEPLOYMENTCONFIG_NAME --replicas=3
```

For example, for the **kitchensink** quickstart, use the following command to scale up the application.

```
$ oc scale deploymentconfig eap-app --replicas=3
```

## 2.6. CHAINED BUILD SUPPORT IN JBOSS EAP FOR OPENSIFT

JBoss EAP for OpenShift supports chained builds in OpenShift.

JBoss EAP for OpenShift templates employ chained builds. When you use these templates, two builds result:

- An intermediate image named **[application name]-build-artifacts**
- The final image, **[application name]**

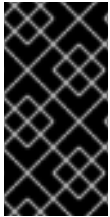
For details about chained builds, see the OpenShift documentation.

### Additional Resources

[OpenShift Chained build documentation](#)

## CHAPTER 3. DEPLOYING JBOSS EAP 7 APPLICATION ON OPENSIFT USING HELM CHART

You can deploy and run your Jakarta EE application with JBoss EAP 7 on OpenShift using Helm charts. Helm is a software package manager that simplifies deployment of applications and services to OpenShift Container Platform clusters. Helm uses a packaging format called charts. A Helm chart is a collection of files that describes the OpenShift Container Platform resources. The following procedures demonstrate how to deploy and run a Jakarta EE application using a Helm chart on the OpenShift Container Platform web console.



### IMPORTANT

This feature is provided as Technology Preview only. It is not supported for use in a production environment, and it might be subject to significant future changes. See [Technology Preview Features Support Scope](#) on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

### 3.1. PREREQUISITES

- You have an OpenShift instance installed and operational. For more information about installing and configuring your OpenShift instance, see [OpenShift Container Platform Getting Started guide](#).
- You are logged in to the OpenShift Container Platform web console. For more information about using the OpenShift web console, see the [OpenShift Container Platform Getting Started guide](#).



### NOTE

- You can also use the [OpenShift Sandbox](#) to deploy and run a JBoss EAP application on OpenShift Container Platform. This is a trial sandbox available for a limited period of time.
- This documentation uses a sample Jakarta EE application. You can use the same procedure to deploy your own Jakarta EE application. For more information, see <https://github.com/jboss-eap-up-and-running/eap7-getting-started>

### 3.2. CREATING A JBOSS EAP EAP 7 APPLICATION WITH HELM

You can create your JBoss EAP 7 application using Helm chart on the OpenShift web console.

#### Procedure

1. In the main navigation, click the drop-down menu and select **Developer**.
2. In the navigation menu, click **Add**.  
The **Add** page opens.
3. In the **Add** page, click **Helm Chart**.
4. In the **Helm Charts** catalog, search for **JBoss EAP 7.4**.
5. Click the **JBoss EAP 7.4** Helm chart tile.

The side panel displays information about the JBoss EAP 7 Helm chart.

#### 6. Click **Install Helm Chart**

Some form sections are collapsed by default. Click > to expand and view its content.



#### NOTE

No updates are required to these sections to proceed.

The details about the Jakarta EE application that you are building and deploying are specified in the **build.uri** field.

```
build:
  uri: https://github.com/jboss-eap-up-and-running/eap7-getting-started
```



#### NOTE

If you are building a different application, you must change this **uri** field to point to the Git repository of that application.

#### 7. Click **Install** to create the JBoss EAP 7 application using the Helm chart.

### Verification

- The Helm release is represented by a dashed box that contains the JBoss EAP icon and **eap74** text. This content is placed outside the dashed box. The deployment is indicated by a circle inside the dashed box with text **D eap74**.
  - Verify that you see an **eap74** Helm Release.
  - Verify that you see an **eap74** deployment.

## 3.3. VIEWING THE HELM RELEASE

After you have successfully created your JBoss EAP 7 application using Helm chart, you can view all the information related to the Helm release.

### Prerequisites

- You have created your JBoss EAP 7 application using Helm chart. See [creating a JBoss EAP 7 application with Helm](#).

### Procedure

1. In the navigation menu, click **Helm**.
2. Click **eap74** Helm release.
 

The **Helm Release details** page opens. It shows all the information related to the Helm release that you installed.
3. Click the **Resources** tab. It lists all the resources created by this Helm release.

### Verification

- Verify that you see a **Deployed** label next to the Helm release **eap74**.

### 3.4. VIEWING THE ASSOCIATED CODE

After you have successfully created your JBoss EAP 7 application using Helm chart, you can view the associated code.

#### Prerequisites

- You have created your JBoss EAP 7 application using Helm chart. See [creating a JBoss EAP EAP 7 application with Helm](#).

#### Procedure

1. In the navigation menu, and click **Topology**.  
In the **Topology view** the **eap74** deployment displays a code icon in the bottom right-hand corner.

This icon either represents the Git repository of the associated code, or if the appropriate operators are installed, it will bring up the associated code in your IDE.

2. If the icon shown is CodeReady Workspaces or Eclipse Che, click it to bring up the associated code in your IDE. Otherwise, click it to navigate to the associated Git repository.

#### Verification

- Verify that you can see the code associated with your application either in your Git repository or in your IDE.

### 3.5. VIEWING THE BUILD STATUS

After you have successfully created your JBoss EAP 7 application using Helm chart, you can view the build status.

#### Prerequisites

- You have created your JBoss EAP 7 application using Helm chart. See [creating a JBoss EAP EAP 7 application with Helm](#).

#### Procedure

1. In the navigation menu, click **Topology**.
2. In the **Topology view**, click the **D eap74** icon.  
A side panel opens with detailed information about the application.
3. In the side panel, click the **Resources** tab. The **Builds** section shows all the details related to builds of the application.





## NOTE

The JBoss EAP 7 application is built in two steps:

- The first build configuration **eap74-build-artifacts** compiles and packages the Jakarta EE application, and creates a JBoss EAP server. The application is run on this JBoss EAP server.

The build may take a few minutes to complete. The build progresses through various states such as **Pending**, **Running**, and **Complete**. The build state is indicated by a relevant message.

When the build is complete, a checkmark and the following message is displayed: **Build #1 was complete**

- The second build configuration **eap74** puts the Jakarta EE deployment and the JBoss EAP server in a runtime image that contains only what is required to run the application.

When the second build is complete, a checkmark and the following message are displayed: **Build #2 was complete**

- When the first build is complete, the second build starts.

## Verification

- Verify the two builds for **eap74-build-artifacts** and **eap74** are complete:
  - The message **Build #1 was complete** is displayed for the **eap74-build-artifacts** build configuration.
  - The message **Build #2 was complete** is displayed for the **eap74** build configuration.

## 3.6. VIEWING THE POD STATUS

After you have successfully created your JBoss EAP 7 application using Helm chart you can view the pod status.

### Prerequisites

- You have created your JBoss EAP 7 application using Helm chart. See [creating a JBoss EAP EAP 7 application with Helm](#).

### Procedure

1. In the navigation menu, and click **Topology**.
2. In the **Topology** view, click **D eap74**.  
A side panel opens with detailed information about the application.
3. In the Details tab, hover over the pod to see the pod status in a tooltip.
  - The number of pods is displayed inside the pod circle.
  - The color of the pod circle indicates the pod status: **Light blue = Pending**, **Blue = Not Ready**, **Dark blue = Running**.

**NOTE**

In the **Topology** view, the dark outer circle of the **Deap74** deployment icon also indicates the pod status.

**Verification**

- Verify that the text inside the pod circle displays **1 pod**.
- Verify that the Pod circle displays **1 Running** when you hover over it.

## 3.7. RUNNING THE JBOSS EAP 7 APPLICATION

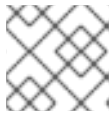
After you have successfully created and built your JBoss EAP 7 application with Helm, you can access it.

**Prerequisites**

- You have created your JBoss EAP 7 application. See [creating a JBoss EAP EAP 7 application with Helm](#).

**Procedure**

- In the **Topology view**, click the external link icon in the top right-hand corner to open the URL and run the application in a separate browser window.

**NOTE**

This action opens the URL on a web browser window.

**Verification**

- Verify that the application **JBoss EAP 7 on Red Hat OpenShift** opens in a separate browser window.

## CHAPTER 4. CONFIGURING THE JBOSS EAP FOR OPENSIFT IMAGE FOR YOUR JAVA APPLICATION

The JBoss EAP for OpenShift image is preconfigured for basic use with your Java applications. However, you can configure the JBoss EAP instance inside the image. The recommended method is to use the OpenShift S2I process, together with application template parameters and environment variables.



### IMPORTANT

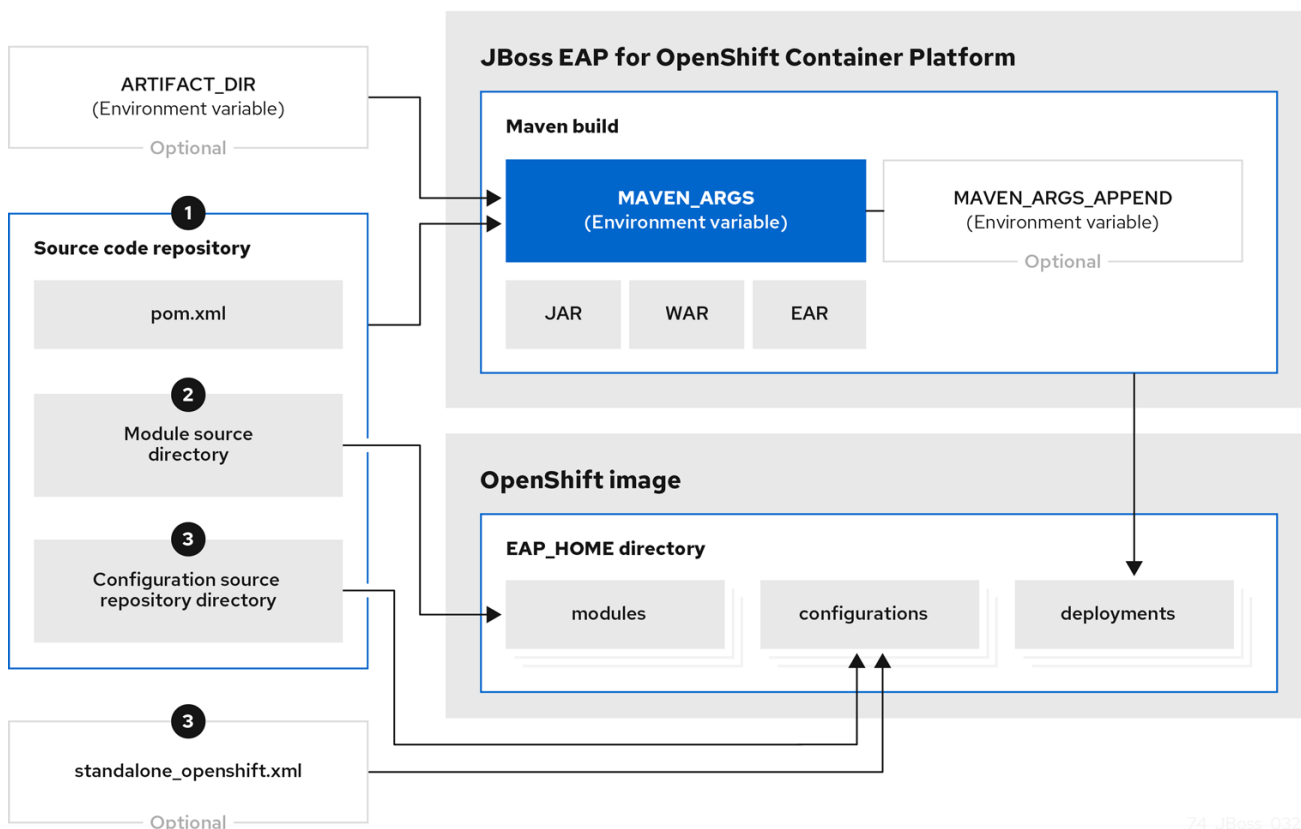
Any configuration changes made on a running container will be lost when the container is restarted or terminated.

This includes any configuration changes made using scripts that are included with a traditional JBoss EAP installation, for example **add-user.sh** or the management CLI.

It is strongly recommended that you use the OpenShift S2I process, together with application template parameters and environment variables, to make any configuration changes to the JBoss EAP instance inside the JBoss EAP for OpenShift image.

### 4.1. HOW THE JBOSS EAP FOR OPENSIFT S2I PROCESS WORKS

Flowchart illustrating the S2I process for JBoss EAP:



1. If a **pom.xml** file is present in the source code repository, the S2I builder image initiates a Maven build process. The Maven build uses the contents of **\$MAVEN\_ARGS**. If a **pom.xml** file is not present in the source code repository, the S2I builder image initiates a binary type build.

To add custom Maven arguments or options, use **\$MAVEN\_ARGS\_APPEND**. The **\$MAVEN\_ARGS\_APPEND** variable appends options to **\$MAVEN\_ARGS**.

By default, the OpenShift profile uses the Maven **package** goal, which includes system properties for skipping tests (**-DskipTests**) and enabling the Red Hat GA repository (**-Dcom.redhat.xpaas.repo**).

The results of a successful Maven build are copied to the **EAP\_HOME/standalone/deployments/** directory inside the JBoss EAP for OpenShift image. This includes all JAR, WAR, and EAR files from the source repository specified by the **\$ARTIFACT\_DIR** environmental variable. The default value of **ARTIFACT\_DIR** is the Maven target directory.



#### NOTE

To use Maven behind a proxy on JBoss EAP for OpenShift image, set the **\$HTTP\_PROXY\_HOST** and **\$HTTP\_PROXY\_PORT** environment variables. Optionally, you can also set the **\$HTTP\_PROXY\_USERNAME**, **\$HTTP\_PROXY\_PASSWORD**, and **\$HTTP\_PROXY\_NONPROXYHOSTS** variables.

2. All files in the **modules** source repository directory are copied to the **EAP\_HOME/modules/** directory in the JBoss EAP for OpenShift image.
3. All files in the **configuration** source repository directory are copied to the **EAP\_HOME/standalone/configuration/** directory in the JBoss EAP for OpenShift image. If you want to use a custom JBoss EAP configuration file, name the file **standalone-openshift.xml**.

#### Additional Resources

- See [Binary \(local\) source](#) on the OpenShift 4.2 documentation for additional information on binary type builds.
- See [Artifact Repository Mirrors](#) for additional guidance on how to instruct the S2I process to use the custom Maven artifacts repository mirror.

## 4.2. CONFIGURING JOSS EAP FOR OPENSIFT USING ENVIRONMENT VARIABLES

Using environment variables is the recommended method of configuring the JBoss EAP for OpenShift image. See the OpenShift documentation for instructions on [specifying environment variables](#) for application containers and build containers.

For example, you can set the JBoss EAP instance's management username and password using environment variables when creating your OpenShift application:

```
oc new-app --template=eap74-basic-s2i \
-p IMAGE_STREAM_NAMESPACE=eap-demo \
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \
-p SOURCE_REPOSITORY_REF=7.4.x \
-p CONTEXT_DIR=kitchensink \
-e ADMIN_USERNAME=myspecialuser \
-e ADMIN_PASSWORD=myspecialp@ssw0rd
```

Available environment variables for the JBoss EAP for OpenShift image are listed in [Reference Information](#).

## 4.2.1. JVM Memory Configuration

The OpenShift EAP image has a mechanism to automatically calculate the default JVM memory settings based on the current environment, but you can also configure the JVM memory settings using environment variables.

### 4.2.1.1. JVM Default Memory Settings

If a memory limit is defined for the current container, and the limit is lower than the total available memory, the default JVM memory settings are calculated automatically. Otherwise, the default JVM memory settings are the default defined in the **standalone.conf** file of the EAP version used as the base server for the image.

The container memory limit is retrieved from the file **/sys/fs/cgroup/memory/memory.limit\_in\_bytes**. The total available memory is retrieved using the **/proc/meminfo** command.

When memory settings are calculated automatically, the following formulas are used:

- Maximum heap size (-Xmx): fifty percent (50%) of user memory
- Initial heap size (-Xms): twenty-five percent (25%) of the calculated maximum heap size.

For example, the defined memory limit is 1 GB, and this limit is lower than the total available memory reported by **/proc/meminfo**, then the memory settings will be: **-Xms128m -Xmx512**

You can use the following environment variables to modify the JVM settings calculated automatically. Note that these variables are only used when default memory size is calculated automatically (in other words, when a valid container memory limit is defined).

- **JAVA\_MAX\_MEM\_RATIO**
- **JAVA\_INITIAL\_MEM\_RATIO**
- **JAVA\_MAX\_INITIAL\_MEM**

You can disable automatic memory calculation by setting the value of the following two environment variables to 0.

- **JAVA\_INITIAL\_MEM\_RATIO**
- **JAVA\_MAX\_MEM\_RATIO**

### 4.2.1.2. JVM Garbage Collection Settings

The EAP image for OpenShift includes settings for both garbage collection and garbage collection logging

#### Garbage Collection Settings

**-XX:+UseParallelOldGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -XX:+ExitOnOutOfMemoryError**

#### Garbage Collection Logging Settings for Java 8 (non-modular JVM)

```
-verbose:gc -Xloggc:/opt/eap/standalone/log/gc.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps
-XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=3M -XX:-
TraceClassUnloading
```

### Garbage Collection Logging Settings for Java 11 (modular JVM)

```
-Xlog:gc*:file=/opt/eap/standalone/log/gc.log:time,uptimemillis:filecount=5,filesize=3M
```

#### 4.2.1.3. Resource Limits in Default Settings

If set, additional default settings are included in the image.

```
-XX:ParallelGCThreads={core-limit} -Djava.util.concurrent.ForkJoinPool.common.parallelism={core-
limit} -XX:CICompilerCount=2
```

The value of {core-limit} is defined using the **JAVA\_CORE\_LIMIT** environment variable, or by the CPU core limit imposed by the container.

The value of **CICompilerCount** is always fixed as 2.

#### 4.2.1.4. JVM Environment Variables

Use these environment variables to configure the JVM in the EAP for OpenShift image.

**Table 4.1. JVM Environment Variables**

Variable Name	Example	Default Value	JVM Settings	Description
JAVA_OPTS	-verbose:class	No default	Multiple	<p>JVM options to pass to the <b>java</b> command.</p> <p>Use <b>JAVA_OPTS_APPEND</b> to configure additional JVM settings. If you use <b>JAVA_OPTS</b>, some unconfigurable defaults are not added to the server JVM settings. You must explicitly add these settings.</p> <p>Using <b>JAVA_OPTS</b> disables certain settings added by default by the container scripts. Disabled settings include</p>



Variable Name	Example	Default Value	JVM Settings	Description
JAVA_MAX_MEM_RATIO	50	50	-Xmx	<p>Use this variable when the <b>-Xmx</b> option is not specified in <b>JAVA_OPTS</b>. The value of this variable is used to calculate a default maximum heap memory size based on the restrictions of the container. If this variable is used in a container without a memory constraint, the variable has no effect. If this variable is used in a container that does have a memory constraint, the value of <b>-Xmx</b> is set to the specified ratio of the container's available memory. The default value, 50 means that 50% of the available memory is used as an upper boundary. To skip calculation of maximum memory, set the value of this variable to 0. No <b>-Xmx</b> option will be added to <b>JAVA_OPTS</b>.</p>



Variable Name	Example	Default Value	JVM Settings	Description
JAVA_INITIAL_MEMORY_RATIO	25	25	-Xms	Use this variable when the <b>-Xms</b> option is not specified in <b>JAVA_OPTS</b> . The value of this variable is used to calculate the default initial heap memory size based on the maximum heap memory. If this variable is used in a container without a memory constraint, the variable has no effect. If this variable is used in a container that does have a memory constraint, the value of <b>-Xms</b> is set to the specified ratio of the <b>-Xmx</b> memory. The default value, 25 means that 25% of the maximum memory is used as the initial heap size. To skip calculation of initial memory, set the value of this variable to 0. No <b>-Xms</b> option will be added to <b>JAVA_OPTS</b> .

Variable Name	Example	Default Value	JVM Settings	Description
JAVA_MAX_INITIAL_MEM	4096	4096	-Xms	Use this variable when the <b>-Xms</b> option is not specified in <b>JAVA_OPTS</b> . The value of this variable is used to calculate the maximum size of the initial memory heap. The value is expressed in megabytes (MB). If this variable is used in a container without a memory constraint, the variable has no effect. If this variable is used in a container that does have a memory constraint, the value of <b>-Xms</b> is set to the value specified in the variable. The default value, 4096, specifies that the maximum initial heap will never be larger than 4096MB.

Variable Name	Example	Default Value	JVM Settings	Description
JAVA_DIAGNOSTICS	true	false (disabled)	<p>The settings depend on the JDK used by the container.</p> <ul style="list-style-type: none"> <li>OpenJDK 8: - XX:NativeMemoryTracking=summary - XX:+PrintGC - XX:+PrintGCDateStamps - XX:+PrintGCTimeStamps - XX:+UnlockDiagnosticVMOptions</li> <li>OpenJDK 11: - Xlog:gc:utctime - XX:NativeMemoryTracking=summary</li> </ul>	<p>Set the value of this variable to <i>true</i> to include diagnostic information in standard output when events occur. If this variable is defined as <i>true</i> in an environment where <b>JAVA_DIAGNOSTICS</b> has already been defined as <i>true</i>, diagnostics are still included.</p>
DEBUG	true	false	- agentlib:jdwp=transport=dt_socket,address=\$DEBUG_PORT,server=y,suspend=n	Enables remote debugging.
DEBUG_PORT	8787	8787	- agentlib:jdwp=transport=dt_socket,address=\$DEBUG_PORT,server=y,suspend=n	Specifies the port used for debugging.

Variable Name	Example	Default Value	JVM Settings	Description
JAVA_CORE_LIMIT		Undefined	- XX:parallelGCThreads - Djava.util.concurrent.ForkJoinPool.common.parallelism - XX:CICompilerCount	A user-defined limit on the number of cores. If the container reports a limit constraint, the value of the JVM settings is limited to the container core limit. The value of -XXCICompilerCount is always 2. By default, this variable is undefined. In that case, if a limit is not defined on the container, the JVM settings are not set.
GC_MIN_HEAP_FREE_RATIO	20	10	- XX:MinHeapFreeRatio	Minimum percentage of heap free after garbage collection to avoid expansion.
GC_MAX_HEAP_FREE_RATIO	40	20	- XX:MaxHeapFreeRatio	Maximum percentage of heap free after garbage collection to avoid shrinking.
GC_TIME_RATIO	4	4	-XX:GCTimeRatio	Specifies the ratio of the time spent outside of garbage collection (for example, time spent in application execution) to the time spent in garbage collection.

Variable Name	Example	Default Value	JVM Settings	Description
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	90	90	-XX:AdaptiveSizePolicyWeight	The weighting given to the current garbage collection time versus the previous garbage collection times.
GC_METASPACE_SIZE	20	96	-XX:MetaspaceSize	The initial metaspace size.
GC_MAX_METASPACE_SIZE	100	256	-XX:MaxMetaspaceSize	The maximum metaspace size.
GC_CONTAINER_OPTIONS	-XX:+UseG1GC	-XX:-UseParallelOldGC	-XX:-UseParallelOldGC	Specifies the Java garbage collection to use. The value of the variable should be the JRE command-line options to specify the required garbage collection. The JRE command specified overrides the default.

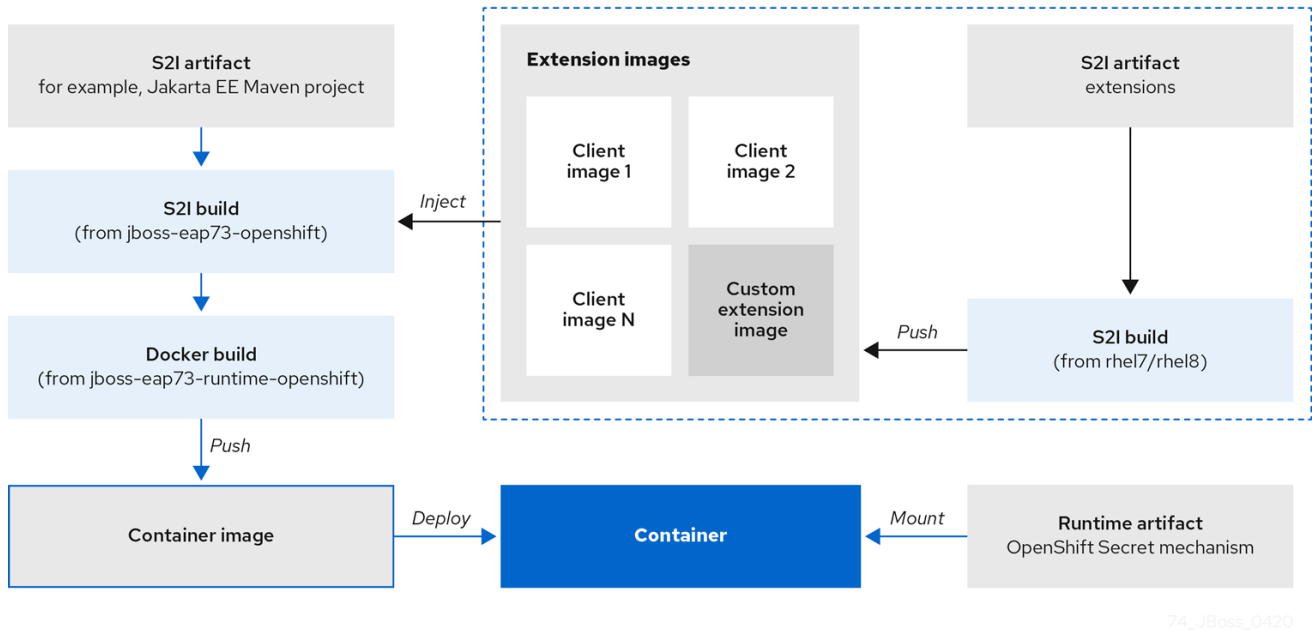
The following environment variables are deprecated:

- **JAVA\_OPTIONS**: Use **JAVA\_OPTS**.
- **INITIAL\_HEAP\_PERCENT**: Use **JAVA\_INITIAL\_MEM\_RATIO**.
- **CONTAINER\_HEAP\_PERCENT**: Use **JAVA\_MAX\_MEM\_RATIO**.

### 4.3. BUILD EXTENSIONS AND PROJECT ARTIFACTS

The JBoss EAP for OpenShift image extends database support in OpenShift using various artifacts. These artifacts are included in the built image through different mechanisms:

- [S2I artifacts](#) that are injected into the image during the S2I process.
- [Runtime artifacts](#) from environment files provided through the OpenShift Secret mechanism.



## IMPORTANT

Support for using the Red Hat-provided internal datasource drivers with the JBoss EAP for OpenShift image is now deprecated. Red Hat recommends that you use JDBC drivers obtained from your database vendor for your JBoss EAP applications.

The following internal datasources are no longer provided with the JBoss EAP for OpenShift image:

- MySQL
- PostgreSQL

For more information about installing drivers, see [Modules, Drivers, and Generic Deployments](#).

For more information on configuring JDBC drivers with JBoss EAP, see [JDBC drivers](#) in the *JBoss EAP Configuration Guide*.

Note that you can also create a custom layer to install these drivers and datasources if you want to add them to a provisioned server.

## Additional Resources

[Capability Trimming in JBoss EAP for OpenShift](#)

### 4.3.1. S2I Artifacts

The S2I artifacts include modules, drivers, and additional generic deployments that provide the necessary configuration infrastructure required for the deployment. This configuration is built into the image during the S2I process so that only the datasources and associated resource adapters need to be configured at runtime.

See [Artifact Repository Mirrors](#) for additional guidance on how to instruct the S2I process to utilize the custom Maven artifacts repository mirror.

### 4.3.1.1. Modules, Drivers, and Generic Deployments

There are a few options for including these S2I artifacts in the JBoss EAP for OpenShift image:

1. Include the artifact in the application source deployment directory. The artifact is downloaded during the build and injected into the image. This is similar to deploying an application on the JBoss EAP for OpenShift image.
2. Include the **CUSTOM\_INSTALL\_DIRECTORIES** environment variable, a list of comma-separated list of directories used for installation and configuration of artifacts for the image during the S2I process. There are two methods for including this information in the S2I:
  - An **install.sh** script in the nominated installation directory. The install script executes during the S2I process and operates with impunity.

#### install.sh Script Example

```
#!/bin/bash

injected_dir=$1
source /usr/local/s2i/install-common.sh
install_deployments ${injected_dir}/injected-deployments.war
install_modules ${injected_dir}/modules
configure_drivers ${injected_dir}/drivers.env
```

The **install.sh** script is responsible for customizing the base image using APIs provided by **install-common.sh**. **install-common.sh** contains functions that are used by the **install.sh** script to install and configure the modules, drivers, and generic deployments.

Functions contained within **install-common.sh**:

- **install\_modules**
- **configure\_drivers**
- **install\_deployments**

#### Modules

A module is a logical grouping of classes used for class loading and dependency management. Modules are defined in the **EAP\_HOME/modules/** directory of the application server. Each module exists as a subdirectory, for example **EAP\_HOME/modules/org/apache/**. Each module directory then contains a slot subdirectory, which defaults to **main** and contains the **module.xml** configuration file and any required JAR files.

For more information about configuring **module.xml** files for MySQL and PostgreSQL JDBC drivers, see the [Datasource Configuration Examples](#) in the JBoss EAP Configuration Guide.

#### Example module.xml File for PostgreSQL Datasource

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="org.postgresql">
  <resources>
    <resource-root path="postgresql-jdbc.jar"/>
  </resources>
</module>
```

```

</resources>
<dependencies>
<module name="javax.api"/>
<module name="javax.transaction.api"/>
</dependencies>
</module>

```

### Example module.xml File for MySQL Connect/J 8 Datasource

```

<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
<resources>
<resource-root path="mysql-connector-java-8.0.Z.jar" />
</resources>
<dependencies>
<module name="javax.api"/>
<module name="javax.transaction.api"/>
</dependencies>
</module>

```



#### NOTE

The ".Z" in **mysql-connector-java-8.0.Z.jar** indicates the version of the **JAR** file downloaded. The file can be renamed, but the name must match the name in the **module.xml** file.

The **install\_modules** function in **install.sh** copies the respective JAR files to the modules directory in JBoss EAP, along with the **module.xml**.

### Drivers

Drivers are installed as modules. The driver is then configured in **install.sh** by the **configure\_drivers** function, the configuration properties for which are defined in a [runtime artifact](#) environment file.

### Adding Datasource Drivers

The MySQL and PostgreSQL datasources are no longer provided as pre-configured internal datasources. You can still install these drivers as modules; see the description in [Modules, Drivers, and Generic Deployments](#). You can obtain these JDBC drivers from the database vendor for your JBoss EAP applications.

Create a **drivers.env** file for each datasource to be installed.

### Example drivers.env File for MySQL Datasource

```

#DRIVER
DRIVERS=MYSQL
MYSQL_DRIVER_NAME=mysql
MYSQL_DRIVER_MODULE=org.mysql
MYSQL_DRIVER_CLASS=com.mysql.cj.jdbc.Driver
MYSQL_XA_DATASOURCE_CLASS=com.mysql.cj.jdbc.MysqlXADataSource

```

### Example drivers.env File for PostgreSQL Datasource

-



```
#DRIVER
DRIVERS=POSTGRES
POSTGRES_DRIVER_NAME=postgresql
POSTGRES_DRIVER_MODULE=org.postgresql
POSTGRES_DRIVER_CLASS=org.postgresql.Driver
POSTGRES_XA_DATASOURCE_CLASS=org.postgresql.xa.PGXADatasource
```

For information about download locations for various drivers, such as MySQL or PostgreSQL, see [JDBC Driver Download Locations](#) in the Configuration Guide.

## Generic Deployments

Deployable archive files, such as JARs, WARs, RARs, or EARs, can be deployed from an injected image using the **install\_deployments** function supplied by the API in **install-common.sh**.

- If the **CUSTOM\_INSTALL\_DIRECTORIES** environment variable has been declared but no **install.sh** scripts are found in the custom installation directories, the following artifact directories will be copied to their respective destinations in the built image:
  - **modules/\*** copied to **\$JBOSS\_HOME/modules/**
  - **configuration/\*** copied to **\$JBOSS\_HOME/standalone/configuration**
  - **deployments/\*** copied to **\$JBOSS\_HOME/standalone/deployments**

This is a basic configuration approach compared to the **install.sh** alternative, and requires the artifacts to be structured appropriately.

## 4.3.2. Runtime Artifacts

### 4.3.2.1. Datasources

There are two types of datasources:

1. Internal datasources. These datasources run on OpenShift, but are not available by default through the Red Hat Registry or in the OpenShift repository. Configuration of these datasources is provided by environment files added to OpenShift Secrets.
2. External datasources. These datasources do not run on OpenShift. Configuration of external datasources is provided by environment files added to OpenShift Secrets.



#### NOTE

For more details about creating and configuring OpenShift Secrets, see [Secrets](#).

You can create the datasource environment file in a directory, such as a configuration directory of your source project. The following example shows the content of a sample datasource environment file:

#### Example: Datasource Environment File

```
DB_SERVICE_PREFIX_MAPPING=PostgresXA-POSTGRES=DS1
DS1_JNDI=java:jboss/datasources/pgds
DS1_DRIVER=postgresql-42.2.5.jar
DS1_USERNAME=postgres
DS1_PASSWORD=postgres
```

```
DS1_MAX_POOL_SIZE=20
DS1_MIN_POOL_SIZE=20
DS1_CONNECTION_CHECKER=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidCo
nnectionChecker
DS1_EXCEPTION_SORTER=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSo
rter
```

The **DB\_SERVICE\_PREFIX\_MAPPING** property is a comma-separated list of datasource property prefixes. These prefixes are then appended to all properties for that datasource. Multiple datasources can then be included in a single environment file. Alternatively, each datasource can be provided in separate environment files.

Datasources contain two types of properties: connection pool-specific properties and database driver-specific properties. The connection pool-specific properties produce a connection to a datasource. Database driver-specific properties determine the driver for a datasource and are configured as a driver S2I artifact.

In the above example, **DS1** is the datasource prefix, **CONNECTION\_CHECKER** specifies a connection checker class used to validate connections for a database, and **EXCEPTION\_SORTER** specifies the exception sorter class used to detect fatal database connection exceptions.

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, the value of which is a comma-separated list of fully qualified environment files as shown below.

```
{
  "Name": "ENV_FILES",
  "Value": "/etc/extensions/datasources1.env,/etc/extensions/datasources2.env"
}
```

#### 4.3.2.2. Resource Adapters

Configuration of resource adapters is provided by environment files added to OpenShift Secrets.

**Table 4.2. Resource Adapter Properties**

Attribute	Description
<i>PREFIX_ID</i>	The identifier of the resource adapter as specified in the server configuration file.
<i>PREFIX_ARCHIVE</i>	The resource adapter archive.
<i>PREFIX_MODULE_SLOT</i>	The slot subdirectory, which contains the <b>module.xml</b> configuration file and any required JAR files.
<i>PREFIX_MODULE_ID</i>	The JBoss Module ID where the object factory Java class can be loaded from.
<i>PREFIX_CONNECTION_CLASS</i>	The fully qualified class name of a managed connection factory or admin object.

Attribute	Description
<code>PREFIX_CONNECTION_JNDI</code>	The JNDI name for the connection factory.
<code>PREFIX_PROPERTY_ParentDirectory</code>	Directory where the data files are stored.
<code>PREFIX_PROPERTY_AllowParentPaths</code>	Set <b>AllowParentPaths</b> to <b>false</b> to disallow <code>..</code> in paths. This prevents requesting files that are not contained in the parent directory.
<code>PREFIX_POOL_MAX_SIZE</code>	The maximum number of connections for a pool. No more connections will be created in each sub-pool.
<code>PREFIX_POOL_MIN_SIZE</code>	The minimum number of connections for a pool.
<code>PREFIX_POOL_PREFILL</code>	Specifies if the pool should be prefilled. Changing this value requires a server restart.
<code>PREFIX_POOL_FLUSH_STRATEGY</code>	How the pool should be flushed in case of an error. Valid values are: <b>FailingConnectionOnly</b> (default), <b>IdleConnections</b> , and <b>EntirePool</b> .

The **RESOURCE\_ADAPTERS** property is a comma-separated list of resource adapter property prefixes. These prefixes are then appended to all properties for that resource adapter. Multiple resource adapter can then be included in a single environment file. In the example below, **MYRA** is used as the prefix for a resource adapter. Alternatively, each resource adapter can be provided in separate environment files.

#### Example: Resource Adapter Environment File

```
#RESOURCE_ADAPTER
RESOURCE_ADAPTERS=MYRA
MYRA_ID=myra
MYRA_ARCHIVE=myra.rar
MYRA_CONNECTION_CLASS=org.javaee7.jca.connector.simple.connector.outbound.MyManagedCo
nnectionFactory
MYRA_CONNECTION_JNDI=java:/eis/MySimpleMFC
```

The resource adapter environment files are added to the OpenShift Secret for the project namespace. These environment files are then called within the template using the **ENV\_FILES** environment property, the value of which is a comma-separated list of fully qualified environment files as shown below.

```
{
  "Name": "ENV_FILES",
  "Value": "/etc/extensions/resourceadapter1.env,/etc/extensions/resourceadapter2.env"
}
```

## 4.4. RESULTS OF USING JBOSS EAP TEMPLATES FOR OPENSIFT

When you use JBoss EAP templates to compile your application, two images might be generated.

An intermediate image named **[application name]-build-artifacts** might be generated before the final image, **[application name]**, is created.

You can remove the **[application name]-build-artifacts** image after your application has been deployed.

## 4.5. SSO CONFIGURATION OF RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM FOR OPENSIFT IMAGES

In Red Hat JBoss Enterprise Application Platform for OpenShift images, SSO is configured to use the legacy **security** subsystem.

The environment variable **SSO\_FORCE\_LEGACY\_SECURITY** is set to **true** in these images.

If you want to use the **elytron** subsystem for SSO security, update the value of the **SSO\_FORCE\_LEGACY\_SECURITY** environment variable to **false**.

## 4.6. DEFAULT DATASOURCE

The datasource **ExampleDS** is not available in JBoss EAP 7.4.

Some quickstarts require this datasource:

- **cmt**
- **thread-racing**

Applications developed by customers might also require the **ExampleDS** datasource.

If you need the default datasource, use the **GENERATE\_DEFAULT\_DATASOURCE** environment variable to include it when provisioning a JBoss EAP server.

```
ENABLE_GENERATE_DEFAULT_DATASOURCE=true
```

## CHAPTER 5. CAPABILITY TRIMMING IN JBOSS EAP FOR OPENSIFT

When building an image that includes JBoss EAP, you can control the JBoss EAP features and subsystems to include in the image.

The default JBoss EAP server included in S2I images includes the complete server and all features. You might want to trim the capabilities included in the provisioned server. For example, you might want to reduce the security exposure of the provisioned server, or you might want to reduce the memory footprint so it is more appropriate for a microservice container.

### 5.1. PROVISION A CUSTOM JBOSS EAP SERVER

To provision a custom server with trimmed capabilities, pass the **GALLEON\_PROVISION\_LAYERS** environment variable during the S2I build phase.

The value of the environment variable is a comma-separated list of the layers to provision to build the server.

For example, if you specify the environment variable as **GALLEON\_PROVISION\_LAYERS=jaxrs-server,sso**, a JBoss EAP server is provisioned with the following capabilities:

- A servlet container
- The ability to configure a datasource
- The **jaxrs**, **weld**, and **jpa** subsystems
- Red Hat SSO integration

### 5.2. AVAILABLE JBOSS EAP LAYERS

Red Hat makes available six layers to customize provisioning of the JBoss EAP server in OpenShift.

Three layers are base layers that provide core functionality. Three are decorator layers that enhance the base layers.

The following Jakarta EE specifications are not supported in any provisioning layer:

- Jakarta Server Faces 2.3
- Jakarta Enterprise Beans 3.2
- Jakarta XML Web Services 2.3

#### 5.2.1. Base Layers

Each base layer includes core functionality for a typical server user case.

##### **datasources-web-server**

This layer includes a servlet container and the ability to configure a datasource.

The following are the JBoss EAP subsystems included by default in the **datasources-web-server**:

- **core-management**

- **datasources**
- **deployment-scanner**
- **ee**
- **elytron**
- **io**
- **jca**
- **jmx**
- **logging**
- **naming**
- **request-controller**
- **security-manager**
- **transactions**
- **undertow**

The following Jakarta EE specifications are supported in this layer:

- Jakarta JSON Processing 1.1
- Jakarta JSON Binding 1.0
- Jakarta Servlet 4.0
- Jakarta Expression Language 3.0
- Jakarta Server Pages 2.3
- Jakarta Standard Tag Library 1.2
- Jakarta Concurrency 1.1
- Jakarta Annotations 1.3
- Jakarta XML Binding 2.3
- Jakarta Debugging Support for Other Languages 1.0
- Jakarta Transactions 1.3
- Jakarta Connectors 1.7

### **jaxrs-server**

This layer enhances the **datasources-web-server** layer with the following JBoss EAP subsystems:

- **jaxrs**
- **weld**

- **jpa**

This layer also adds Infinispan-based second-level entity caching locally in the container.

The following Jakarta EE specifications are supported in this layer in addition to those supported in the **datasources-web-server** layer:

- Jakarta Contexts and Dependency Injection 2.0
- Jakarta Bean Validation 2.0
- Jakarta Interceptors 1.2
- Jakarta RESTful Web Services 2.1
- Jakarta Persistence 2.2

### **cloud-server**

This layer enhances the **jaxrs-server** layer with the following JBoss EAP subsystems:

- **resource-adapters**
- **messaging-activemq** (remote broker messaging, not embedded messaging)

This layer also adds the following observability features to the **jaxrs-server** layer:

- Health subsystem
- Metrics subsystem

The following Jakarta EE specification is supported in this layer in addition to those supported in the **jaxrs-server** layer:

- Jakarta Security 1.0

## 5.2.2. Decorator Layers

Decorator layers are not used alone. You can configure one or more decorator layers with a base layer to deliver additional functionality.

### **sso**

This decorator layer adds Red Hat Single Sign-On integration to the provisioned server.

### **observability**

This decorator layer adds the following observability features to the provisioned server:

- Health subsystem
- Metrics subsystem



#### **NOTE**

This layer is built in to the **cloud-server** layer. You do not need to add this layer to the **cloud-server** layer.

### **web-clustering**

This layer adds embedded Infinispan-based web session clustering to the provisioned server.

## 5.3. PROVISIONING USER-DEVELOPED LAYERS IN JBOSS EAP

In addition to provisioning layers available from Red Hat, you can provision custom layers you develop.

### Procedure

1. Build a custom layer using the Galleon Maven plugin.  
For more information, see [Preparing the Maven project](#).
2. Deploy the custom layer to an accessible Maven repository.
3. You can use custom Galleon feature-pack environment variables to customize Galleon feature-packs and layers during the S2I image build process.  
For more information about customizing Galleon feature-packs and layers, see [Using the custom Galleon feature-pack during S2I build](#).
4. *Optional:* Create a custom provisioning file to reference the user-defined layer and supported JBoss EAP layers and store it in your application directory.  
For more information about creating a custom provisioning file, see [Custom provisioning files for JBoss EAP](#).
5. Run the S2I process to provision a JBoss EAP server in OpenShift.  
For more information, see [Using the custom Galleon feature-pack during S2I build](#).

### 5.3.1. Building and using custom Galleon layers for JBoss EAP

Custom Galleon layers are packaged inside a Galleon feature-pack that is designed to run with JBoss EAP 7.4.

In Openshift, you can build and use a Galleon feature-pack that contains layers to provision, for example, a MariaDB driver and data source for the JBoss EAP 7.4 server. A layer contains the content that is installed in the server. A layer can update the server XML configuration file and add content to the server installation.

This section documents how to build and use in OpenShift a Galleon feature-pack containing layers to provision a MariaDB driver and data source for the JBoss EAP 7.4 server.

#### 5.3.1.1. Preparing the Maven project

Galleon feature-packs are created using Maven. This procedure includes the steps to create a new Maven project.

### Procedure

1. To create a new Maven project, run the following command:

```
mvn archetype:generate -DarchetypeGroupId=org.codehaus.mojo.archetypes -  
DarchetypeArtifactId=pom-root -DgroupId=org.example.mariadb -DartifactId=mariadb-  
galleon-pack -DinteractiveMode=false
```

2. In the directory **mariadb-galleon-pack**, update the **pom.xml** file to include the Red Hat Maven repository:



```

<repositories>
  <repository>
    <id>redhat-ga</id>
    <name>Redhat GA</name>
    <url>https://maven.repository.redhat.com/ga/</url>
  </repository>
</repositories>

```

3. Update the **pom.xml** file to add dependencies on the EAP Galleon feature-pack and the MariaDB driver:

```

<dependencies>
  <dependency>
    <groupId>org.jboss.eap</groupId>
    <artifactId>wildfly-ee-galleon-pack</artifactId>
    <version>7.4.4.GA-redhat-00011</version>
    <type>zip</type>
  </dependency>
  <dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>3.0.5</version>
  </dependency>
</dependencies>

```

4. Update the **pom.xml** file to include the Maven plugin that is used to build the Galleon feature-pack:

```

<build>
  <plugins>
    <plugin>
      <groupId>org.wildfly.galleon-plugins</groupId>
      <artifactId>wildfly-galleon-maven-plugin</artifactId>
      <version>5.2.11.Final</version>
      <executions>
        <execution>
          <id>mariadb-galleon-pack-build</id>
          <goals>
            <goal>build-user-feature-pack</goal>
          </goals>
          <phase>compile</phase>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

### 5.3.1.2. Adding the feature pack content

This procedure helps you add layers to a custom Galleon feature-pack, for example, the feature-pack including the MariaDB driver and datasource layers.

#### Prerequisites

- You have created a Maven project. For more details, see [Preparing the Maven project](#).

## Procedure

- Create the directory, **src/main/resources**, within a custom feature-pack Maven project, for example, see [Preparing the Maven project](#). This directory is the root directory containing the feature-pack content.
- Create the directory **src/main/resources/modules/org/mariadb/jdbc/main**.
- In the **main** directory, create a file named **module.xml** with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<module name="org.mariadb.jdbc" xmlns="urn:jboss:module:1.8">
  <resources>
    <artifact name="${org.mariadb.jdbc:mariadb-java-client}"/> 1
  </resources>
  <dependencies> 2
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

1 The MariaDB driver **groupId** and **artifactId**. At provisioning time, the actual driver jar file gets installed. The version of the driver is referenced from the **pom.xml** file.

2 The **JBoss Modules** modules dependencies for the MariaDB driver.

- Create the directory **src/main/resources/layers/standalone/**. This is the root directory of all the layers that the Galleon feature-pack is defining.
- Create the directory **src/main/resources/layers/standalone/mariadb-driver**.
- In the **mariadb-driver** directory, create the **layer-spec.xml** file with the following content:

```
<?xml version="1.0" ?>
<layer-spec xmlns="urn:jboss:galleon:layer-spec:1.0" name="mariadb-driver">
  <feature spec="subsystem.datasources"> 1
    <feature spec="subsystem.datasources.jdbc-driver">
      <param name="driver-name" value="mariadb"/>
      <param name="jdbc-driver" value="mariadb"/>
      <param name="driver-xa-datasource-class-name"
value="org.mariadb.jdbc.MariaDbDataSource"/>
      <param name="driver-module-name" value="org.mariadb.jdbc"/>
    </feature>
  </feature>
  <packages> 2
    <package name="org.mariadb.jdbc"/>
  </packages>
</layer-spec>
```

1 Update the datasources subsystem configuration with a JDBC-driver named MariaDB, implemented by the module **org.mariadb.jdbc**.

- 2 The **JBoss Modules** module containing the driver classes that are installed when the layer is provisioned.

The **mariadb-driver** layer updates the datasources subsystem with the configuration of a JDBC driver, implemented by the **JBoss Modules** module.

7. Create the directory **src/main/resources/layers/standalone/mariadb-datasource**.
8. In the **mariadb-datasource** directory, create the **layer-spec.xml** file with the following content:

```
<?xml version="1.0" ?>
<layer-spec xmlns="urn:jboss:galleon:layer-spec:1.0" name="mariadb-datasource">
<dependencies>
  <layer name="mariadb-driver"/> 1
</dependencies>

<feature spec="subsystem.datasources.data-source"> 2
  <param name="data-source" value="MariaDBDS"/>
  <param name="jndi-name"
value="java:jboss/datasources/${env.MARIADB_DATASOURCE:MariaDBDS}"/>
  <param name="connection-url"
value="jdbc:mariadb://${env.MARIADB_HOST:localhost}:${env.MARIADB_PORT:3306}/${env.
MARIADB_DATABASE}"/> 3
  <param name="driver-name" value="mariadb"/>
  <param name="user-name" value="${env.MARIADB_USER}"/> 4
  <param name="password" value="${env.MARIADB_PASSWORD}"/>
</feature>
</layer-spec>
```

- 1 This dependency enforces the provisioning of the MariaDB driver when the datasource is provisioned. All the layers a layer depends on are automatically provisioned when that layer is provisioned.
- 2 Update the datasources subsystem configuration with a datasource named MariaDBDS.
- 3 Datasource's name, host, port, and database values are resolved from the environment variables **MARIADB\_DATASOURCE**, **MARIADB\_HOST**, **MARIADB\_PORT**, and **MARIADB\_DATABASE**, which are set when the server is started.
- 4 User name and password values are resolved from the environment variables **MARIADB\_USER** and **MARIADB\_PASSWORD**.

9. Build the Galleon feature-pack by running the following command:

```
mvn clean install
```

The file **target/mariadb-galleon-pack-1.0-SNAPSHOT.zip** is created.

### 5.3.1.3. Using the custom Galleon feature-pack during S2I build

A custom feature-pack must be made available to the Maven build that occurs during OpenShift S2I build. This is usually achieved by deploying the custom feature-pack as an artifact, for example, **org.example.mariadb:mariadb-galleon-pack:1.0-SNAPSHOT** to an accessible Maven repository.

In order to test the feature-pack before deployment, you can use the EAP S2I builder image capability that allows you to make use of a locally built Galleon feature-pack. Use the following procedure example to customize the **todo-backend** EAP quickstart with the use of MariaDB driver instead of PostgreSQL driver.



## NOTE

- For more information about the **todo-backend** EAP quickstart, see [EAP quickstart](#).
- For more information about configuring the JBoss EAP S2I image for custom Galleon feature-pack usage, see [Configure Galleon by using advanced environment variables](#).

## Prerequisites

- You have OpenShift command-line installed
- You are logged in to an OpenShift cluster
- You have installed the JBoss EAP OpenShift images in your cluster
- You have configured access to the **Red Hat Container** registry. For detailed information, see [Red Hat Container Registry](#).
- You have created a custom Galleon feature-pack. For detailed information, see [Preparing the Maven project](#).

## Procedure

1. Start the MariaDB database by running the following command:

```
oc new-app -e MYSQL_USER=admin -e MYSQL_PASSWORD=admin -e
MYSQL_DATABASE=mariadb registry.redhat.io/rhscv/mariadb-101-rhel7
```

The OpenShift service **mariadb-101-rhel7** is created and started.

2. Create a secret from the feature-pack ZIP archive, generated by the custom feature-pack Maven build, by running the following command within the Maven project directory **mariadb-galleon-pack**:

```
oc create secret generic mariadb-galleon-pack --from-file=target/mariadb-galleon-pack-1.0-
SNAPSHOT.zip
```

The secret **mariadb-galleon-pack** is created. When initiating the S2I build, this secret is used to mount the feature-pack zip file in the pod, making the file available during the server provisioning phase.

3. To create a new OpenShift build to build an application image containing the **todo-backend** quickstart deployment running inside a server trimmed with Galleon, run the following command:

```
oc new-build jboss-eap74-openjdk11-openshift:latest~https://github.com/jboss-
developer/jboss-eap-quickstarts#EAP_7.4.0.GA \
--context-dir=todo-backend \
```

```

--env=GALLEON_PROVISION_FEATURE_PACKS="org.example.mariadb:mariadb-galleon-
pack:1.0-SNAPSHOT" \ ❶
--env=GALLEON_PROVISION_LAYERS="jaxrs-server,mariadb-datasource" \ ❷
--env=GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO="/tmp/repo" \ ❸
--env=MAVEN_ARGS_APPEND="-Dcom.redhat.xpaas.repo.jbossorg" \
--build-secret=mariadb-galleon-pack:/tmp/repo/org/example/mariadb/mariadb-galleon-
pack/1.0-SNAPSHOT \ ❹
--name=todos-app-build

```

- ❶ The custom feature-pack environment variable that contains a comma separated list of feature-pack Maven coordinates, such as **groupid:artifactId:version**.
- ❷ The set of Galleon layers that are used to provision the server. **jaxrs-server** is a base server layer and **mariadb-datasource** is the custom layer that brings the MariaDB driver and a new datasource to the server installation.
- ❸ The location of the local Maven repository within the image that contains the MariaDB feature-pack. This repository is populated when mounting the secret inside the image.
- ❹ The **mariadb-galleon-pack** secret is mounted in the **/tmp/repo/org/example/mariadb/mariadb-galleon-pack/1.0-SNAPSHOT** directory.

4. To start a new build from the created OpenShift build, run the following command:

```
oc start-build todos-app-build
```

After successful command execution, the image **todos-app-build** is created.

5. To create a new deployment, provide the environment variables that are required to bind the datasource to the running MariaDB database by executing the following command:

```

oc new-app --name=todos-app todos-app-build \
--env=MARIADB_PORT=3306 \
--env=MARIADB_USER=admin \
--env=MARIADB_PASSWORD=admin \
--env=MARIADB_HOST=mariadb-101-rhel7 \
--env=MARIADB_DATABASE=mariadb \
--env=MARIADB_DATASOURCE=ToDoS ❶

```

- ❶ The quickstart expects the datasource to be named **ToDoS**



#### NOTE

For more details about the custom Galleon feature-pack environment variables, see [Custom Galleon feature-pack environment variables](#)

6. To expose the **todos-app** application, run the following command:

```
oc expose svc/todos-app
```

7. To create a new task, run the following command:

```
curl -X POST http://$(oc get route todos-app --template='{{ .spec.host }}') \
  -H 'Content-Type: application/json' \
  -d '{"title":"todo1"}
```

8. To access the list of tasks, run the following command:

```
curl http://$(oc get route todos-app --template='{{ .spec.host }}')
```

The added task is displayed in a browser.

### 5.3.1.4. Custom Provisioning Files for JBoss EAP

Custom provisioning files are XML files with the file name **provisioning.xml** that are stored in the **galleon** subdirectory.

Using the **provisioning.xml** file is an alternative to the usage of **GALLEON\_PROVISION\_FEATURE\_PACKS** and **GALLEON\_PROVISION\_LAYERS** environment variables. During S2I build, the **provisioning.xml** file is used to provision the custom EAP server.



#### IMPORTANT

Do not create a custom provisioning file when using the **GALLEON\_PROVISION\_LAYERS** environment variable, because this environment variable configures the S2I build process to ignore the file.

The following code illustrates a custom provisioning file.

```
<?xml version="1.0" ?>
<installation xmlns="urn:jboss:galleon:provisioning:3.0">
  <feature-pack location="eap-s2i@maven(org.jboss.universe:s2i-universe)"> 1
    <default-configs inherit="false"/> 2
    <packages inherit="false"/> 3
  </feature-pack>
  <feature-pack location="org.example.mariadb:mariadb-galleon-pack:1.0-SNAPSHOT"> 4
    <default-configs inherit="false"/>
    <packages inherit="false"/>
  </feature-pack>
  <config model="standalone" name="standalone.xml"> 5
    <layers>
      <include name="jaxrs-server"/>
      <include name="mariadb-datasource"/>
    </layers>
  </config>
  <options> 6
    <option name="optional-packages" value="passive+"/>
  </options>
</installation>
```

- 1 This element instructs the provisioning process to provision the current eap-s2i feature-pack. Note that a builder image includes only one feature pack.
- 2 This element instructs the provisioning process to exclude default configurations.

- 3 This element instructs the provisioning process to exclude default packages.
- 4 This element instructs the provisioning process to provision the **org.example.mariadb:mariadb-galleon-pack:1.0-SNAPSHOT** feature pack. The child elements instruct the process to exclude default configurations and default packages.
- 5 This element instructs the provisioning process to create a custom standalone configuration. The configuration includes the **jaxrs-server** base layer and the **mariadb-datasource** custom layer from the **org.example.mariadb:mariadb-galleon-pack:1.0-SNAPSHOT** feature pack.
- 6 This element instructs the provisioning process to optimize provisioning of JBoss EAP modules.

### Additional resources

- For more information about using the **GALLEON\_PROVISION\_LAYERS** environment variable, see [Provision a Custom JBoss EAP server](#).

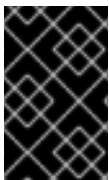
### 5.3.2. Configure Galleon by using advanced environment variables

You can use advanced custom Galleon feature pack environment variables to customize the location where you store your custom Galleon feature packs and layers during the S2I image build process. These advanced custom Galleon feature pack environment variables are as follows:

- **GALLEON\_DIR=<path>**, which overrides the default **<project\_root\_dir>/galleon** directory path to **<project\_root\_dir>/<GALLEON\_DIR>**.
- **GALLEON\_CUSTOM\_FEATURE\_PACKS\_MAVEN\_REPO=<path>**, which overrides the **<project root dir>/galleon/repository** directory path with an absolute path to a Maven local repository cache directory. This repository contains custom Galleon feature packs.

You must locate the Galleon feature pack archive files inside a sub-directory that is compliant with the Maven local-cache file system configuration. For example, locate the **org.examples:my-feature-pack:1.0.0.Final** feature pack inside the **path-to-repository/org/examples/my-feature-pack/1.0.0.Final/my-feature-pack-1.0.0.Final.zip** path.

You can configure your Maven project settings by creating a **settings.xml** file in the **<project\_root>/<GALLEON\_DIR>** directory. The default value for **GALLEON\_DIR** is **<project\_root\_dir>/galleon**. Maven uses the file to provision your custom Galleon feature packs for your application. If you do not create a **settings.xml** file, Maven uses a default **settings.xml** file that was created by the S2I image.



#### IMPORTANT

Do not specify a local Maven repository location in a **settings.xml** file, because the S2I builder image specifies a location to your local Maven repository. The S2I builder image uses this location during the S2I build process.

### Additional resources

- For more information about custom Galleon feature pack environment variables, see [custom Galleon feature pack environment variables](#).

### 5.3.3. Custom Galleon feature pack environment variables

You can use any of the following custom Galleon feature pack environment variables to customize how you use your JBoss EAP S2I image.

**Table 5.1. Descriptions of custom Galleon feature pack environment variables**

Environment variable	Description
GALLEON_DIR=<path>	<p>Where &lt;path&gt; is a directory relative to the root directory of your application project. Your &lt;path&gt; directory contains your optional Galleon custom content, such as the <b>settings.xml</b> file and local Maven repository cache. This cache contains the custom Galleon feature packs.</p> <p>Directory defaults to <b>galleon</b>.</p>
GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO=<path>	<p>&lt;path&gt; is the absolute path to a Maven local repository directory that contains custom feature packs. Directory defaults to <b>galleon/repository</b>.</p>
GALLEON_PROVISION_FEATURE_PACKS=<list_of_galleon_feature_packs>	<p>Where &lt;list_of_galleon_feature_packs&gt; is a comma-separated list of your custom Galleon feature packs identified by Maven coordinates. The listed feature packs must be compatible with the version of the JBoss EAP 7.4 server present in the builder image.</p> <p>You can use the <b>GALLEON_PROVISION_LAYERS</b> environment variable to set the Galleon layers, which were defined by your custom feature packs, for your server.</p>



## CHAPTER 6. TROUBLESHOOTING

### 6.1. TROUBLESHOOTING POD RESTARTS

Pods can restart for a number of reasons, but a common cause of JBoss EAP pod restarts might include OpenShift resource constraints, especially out-of-memory issues. See the OpenShift documentation for more information on [OpenShift pod eviction](#).

By default, JBoss EAP for OpenShift templates are configured to automatically restart affected containers when they encounter situations like out-of-memory issues. The following steps can help you diagnose and troubleshoot out-of-memory and other pod restart issues.

1. Get the name of the pod that has been having trouble.  
You can see pod names, as well as the number times each pod has restarted with the following command.

```
$ oc get pods
```

2. To diagnose why a pod has restarted, you can examine the JBoss EAP logs of the previous pod, or the OpenShift events.
  - a. To see the JBoss EAP logs of the previous pod, use the following command.

```
oc logs --previous POD_NAME
```

- b. To see the OpenShift events, use the following command.

```
$ oc get events
```

3. If a pod has restarted because of a resource issue, you can attempt to modify your OpenShift pod configuration to increase its [resource requests and limits](#). See the OpenShift documentation for more information on [configuring pod compute resources](#).

### 6.2. TROUBLESHOOTING USING THE JBOSS EAP MANAGEMENT CLI

The JBoss EAP management CLI, ***EAP\_HOME/bin/jboss-cli.sh***, is accessible from within a container for troubleshooting purposes.



#### IMPORTANT

It is not recommended to make configuration changes in a running pod using the JBoss EAP management CLI. Any configuration changes made using the management CLI in a running container will be lost when the container restarts.

To make configuration changes to JBoss EAP for OpenShift, see [Configuring the JBoss EAP for OpenShift Image for Your Java Application](#).

1. First open a remote shell session to the running pod.

```
$ oc rsh POD_NAME
```

2. Run the following command from the remote shell session to launch the JBoss EAP management CLI:

```
█ $ /opt/eap/bin/jboss-cli.sh
```

## CHAPTER 7. EAP OPERATOR FOR AUTOMATING APPLICATION DEPLOYMENT ON OPENSIFT

EAP operator is a JBoss EAP-specific controller that extends the OpenShift API. You can use the EAP operator to create, configure, manage, and seamlessly upgrade instances of complex stateful applications.

The EAP operator manages multiple JBoss EAP Java application instances across the cluster. It also ensures safe transaction recovery in your application cluster by verifying all transactions are completed before scaling down the replicas and marking a pod as **clean** for termination. The EAP operator uses **StatefulSet** for the appropriate handling of Jakarta Enterprise Beans remoting and transaction recovery processing. The **StatefulSet** ensures persistent storage and network hostname stability even after pods are restarted.

You must install the EAP operator using OperatorHub, which can be used by OpenShift cluster administrators to discover, install, and upgrade operators.

In OpenShift Container Platform 4, you can use the Operator Lifecycle Manager (OLM) to install, update, and manage the lifecycle of all operators and their associated services running across multiple clusters.

The OLM runs by default in OpenShift Container Platform 4. It aids cluster administrators in installing, upgrading, and granting access to operators running on their cluster. The OpenShift Container Platform web console provides management screens for cluster administrators to install operators, as well as grant specific projects access to use the catalog of operators available on the cluster.

For more information about operators and the OLM, see the [OpenShift documentation](#).

### 7.1. INSTALLING EAP OPERATOR USING THE WEB CONSOLE

As a JBoss EAP cluster administrator, you can install an EAP operator from Red Hat OperatorHub using the OpenShift Container Platform web console. You can then subscribe the EAP operator to one or more namespaces to make it available for developers on your cluster.

Here are a few points you must be aware of before installing the EAP operator using the web console:

- **Installation Mode:** Choose **All namespaces on the cluster (default)** to have the operator installed on all namespaces or choose individual namespaces, if available, to install the operator only on selected namespaces.
- **Update Channel:** If the EAP operator is available through multiple channels, you can choose which channel you want to subscribe to. For example, to deploy from the **stable** channel, if available, select it from the list.
- **Approval Strategy:** You can choose **automatic** or **manual** updates. If you choose automatic updates for the EAP operator, when a new version of the operator is available, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of EAP operator. If you choose manual updates, when a newer version of the operator is available, the OLM creates an update request. You must then manually approve the update request to have the operator updated to the new version.



## NOTE

The following procedure might change in accordance with the modifications in the OpenShift Container Platform web console. For the latest and most accurate procedure, see the [Installing from the OperatorHub using the web console](#) section in the latest version of the *Working with Operators in OpenShift Container Platform* guide.

## Prerequisites

- Access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

## Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **OperatorHub**.
2. Scroll down or type **EAP** into the **Filter by keyword** box to find the EAP operator.
3. Select JBoss EAP operator and click **Install**.
4. On the **Create Operator Subscription** page:
  - a. Select one of the following:
    - **All namespaces on the cluster (default)** installs the operator in the default **openshift-operators** namespace to watch and be made available to all namespaces in the cluster. This option is not always available.
    - **A specific namespace on the cluster** installs the operator in a specific, single namespace that you choose. The operator is made available for use only in this single namespace.
  - b. Select an **Update Channel**.
  - c. Select **Automatic** or **Manual** approval strategy, as described earlier.
5. Click **Subscribe** to make the EAP operator available to the selected namespaces on this OpenShift Container Platform cluster.
  - a. If you selected a manual approval strategy, the subscription's upgrade status remains **Upgrading** until you review and approve its install plan. After you approve the install plan on the **Install Plan** page, the subscription upgrade status moves to **Up to date**.
  - b. If you selected an automatic approval strategy, the upgrade status moves to **Up to date** without intervention.
6. After the subscription's upgrade status is **Up to date**, select **Operators** → **Installed Operators** to verify that the EAP ClusterServiceVersion (CSV) shows up and its **Status** changes to **InstallSucceeded** in the relevant namespace.



## NOTE

For the **All namespaces...** installation mode, the status displayed is **InstallSucceeded** in the **openshift-operators** namespace. In other namespaces the status displayed is **Copied**.

- If the **Status** field does not change to **InstallSucceeded**, check the logs in any pod in the **openshift-operators** project (or other relevant namespace if **A specific namespace...** installation mode was selected) on the **Workloads → Pods** page that are reporting issues to troubleshoot further.

## 7.2. INSTALLING EAP OPERATOR USING THE CLI

As a JBoss EAP cluster administrator, you can install an EAP operator from Red Hat OperatorHub using the OpenShift Container Platform CLI. You can then subscribe the EAP operator to one or more namespaces to make it available for developers on your cluster.

When installing the EAP operator from the OperatorHub using the CLI, use the **oc** command to create a **Subscription** object.

### Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- You have installed the **oc** tool in your local system.

### Procedure

- View the list of operators available to the cluster from the OperatorHub:

```
$ oc get packagemanifests -n openshift-marketplace | grep eap
NAME          CATALOG          AGE
...
eap           Red Hat Operators 43d
...
```

- Create a **Subscription** object YAML file (for example, **eap-operator-sub.yaml**) to subscribe a namespace to your EAP operator. The following is an example **Subscription** object YAML file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: eap
  namespace: openshift-operators
spec:
  channel: stable
  installPlanApproval: Automatic
  name: eap 1
  source: redhat-operators 2
  sourceNamespace: openshift-marketplace
```

- Name of the operator to subscribe to.
- The EAP operator is provided by the **redhat-operators** CatalogSource.

For information about channels and approval strategy, see the [web console](#) version of this procedure.

- Create the **Subscription** object from the YAML file:

```
$ oc apply -f eap-operator-sub.yaml
$ oc get csv -n openshift-operators
NAME          DISPLAY   VERSION  REPLACES  PHASE
eap-operator.v1.0.0  JBoss EAP  1.0.0      Succeeded
```

The EAP operator is successfully installed. At this point, the OLM is aware of the EAP operator. A ClusterServiceVersion (CSV) for the operator appears in the target namespace, and APIs provided by the EAP operator is available for creation.

## 7.3. THE EAP-S2I-BUILD TEMPLATE FOR CREATING APPLICATION IMAGES

Use the **eap-s2i-build** template to create your application images. The **eap-s2i-build** template adds several parameters to configure the location of the application source repository and the EAP S2I images to use to build your application.

The **APPLICATION\_IMAGE** parameter in the **eap-s2i-build** template specifies the name of the imagestream corresponding to the application image. For example, if you created an application image named **my-app** from the **eap-s2i-build** template, you can use the **my-app:latest** imagestreamtag from the **my-app** imagestream to deploy your application. For more information about the parameters used in the **eap-s2i-build** template, see [Building an application image using eap-s2i-build template](#).

With this template, the EAP operator can seamlessly upgrade your applications deployed on OpenShift. To enable seamless upgrades, you must configure a webhook in your GitHub repository and specify the webhook in the build configuration. The webhook notifies OpenShift when your repository is updated and a new build is triggered.

You can use this template to build an application image using an imagestream for any JBoss EAP version, such as JBoss EAP 7.4, JBoss EAP XP, or JBoss EAP CD.

### Additional resources

- [Building an application image using eap-s2i-build template](#).

## 7.4. BUILDING AN APPLICATION IMAGE USING EAP-S2I-BUILD TEMPLATE

The **eap-s2i-build** template adds several parameters to configure the location of your application source repository and the EAP S2I images to use to build the application. With this template, you can use an imagestream for any JBoss EAP version, such as JBoss EAP 7.4, JBoss EAP XP, or JBoss EAP CD.

### Procedure

1. Import EAP images in OpenShift. For more information, see [Importing the OpenShift image streams and templates for JBoss EAP XP](#).
2. Configure the imagestream to receive updates about the changes in the application imagestream and to trigger new builds. For more information, see [Configuring periodic importing of imagestreamtags](#).
3. Create the **eap-s2i-build** template for building the application image using EAP S2I images:

```
$ oc replace --force -f https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-templates/master/eap-s2i-build.yaml
```

This **eap-s2i-build** template creates two build configurations and two imagestreams corresponding to the intermediate build artifacts and the final application image.

4. Process the **eap-s2i-build** template with parameters to create the resources for the final application image. The following example creates an application image, **my-app**:

```
$ oc process eap-s2i-build \
  -p APPLICATION_IMAGE=my-app \ 1
  \
  -p EAP_IMAGE=jboss-eap-xp1-openjdk11-openshift:1.0 \ 2
  -p EAP_RUNTIME_IMAGE=jboss-eap-xp1-openjdk11-runtime-openshift:1.0 \ 3
  -p EAP_IMAGESTREAM_NAMESPACE=$(oc project -q) \ 4
  \
  -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts.git \ 5
  -p SOURCE_REPOSITORY_REF=xp-1.0.x \ 6
  -p CONTEXT_DIR=microprofile-config | oc create -f - 7
```

- 1 The name for the application imagestream. The application image is tagged with the **latest** tag.
- 2 The imagestreamtag for EAP builder image.
- 3 The imagestreamtag for EAP runtime image.
- 4 The namespace in which the imagestreams for Red Hat Middleware images are installed. If omitted, the **openshift** namespace is used. Modify this only if you have installed the imagestreams in a namespace other than **openshift**.
- 5 The Git source URL of your application.
- 6 The Git branch or tag reference
- 7 The path within the Git repository that contains the application to build.

5. Prepare the application image for deployment using the EAP operator.

- a. Configure the **WildFlyServer** resource:

```
$ cat > my-app.yaml<<EOF
apiVersion: wildfly.org/v1alpha1
kind: WildFlyServer
metadata:
  name: my-app
spec:
  applicationImage: 'my-app:latest'
  replicas: 1
EOF
```

- b. Apply the settings and let the EAP operator create a new **WildFlyServer** resource that references this application image:

```
$ oc apply -f my-app.yaml
```

- c. View the **WildFlyServer** resource with the following command:

```
$ oc get wfly my-app
```

#### Additional resources

- For more information about importing an application imagestream, see [Importing the latest OpenShift image streams and templates for JBoss EAP XP](#).
- For more information about periodic importing of imagestreams, see [Configuring periodic importing of imagestreamtags](#).

## 7.5. DEPLOYING A JAVA APPLICATION ON OPENSIFT USING THE EAP OPERATOR

The EAP operator helps automate Java application deployment on OpenShift. For information about the EAP operator APIs, see [EAP Operator: API Information](#).

#### Prerequisites

- You have installed EAP operator. For more information about installing the EAP operator, see [Installing EAP Operator Using the Webconsole](#) and [Installing EAP Operator Using the CLI](#).
- You have built a Docker image of the user application using JBoss EAP for OpenShift Source-to-Image (S2I) build image.
- The **APPLICATION\_IMAGE** parameter in your **eap-s2i-build** template has an imagestream, if you want to enable automatic upgrade of your application after it is deployed on OpenShift. For more information about building your application image using the **eap-s2i-build** template, see [Building an application image using eap-s2i-build template](#).
- You have created a **Secret** object, if your application's CustomResourceDefinition (CRD) file references one. For more information about creating a new **Secret** object, see [Creating a Secret](#).
- You have created a **ConfigMap**, if your application's CRD file references one. For information about creating a **ConfigMap**, see [Creating a ConfigMap](#).
- You have created a **ConfigMap** from the **standalone.xml** file, if you choose to do so. For information about creating a **ConfigMap** from the **standalone.xml** file, see [Creating a ConfigMap from a standalone.xml File](#).



#### NOTE

Providing a **standalone.xml** file from the **ConfigMap** is not supported in JBoss EAP 7.

#### Procedure

1. Open your web browser and log on to OperatorHub.



2. Select the **Project** or namespace you want to use for your Java application.
3. Navigate to **Installed Operator** and select **JBoss EAP operator**.
4. On the **Overview** tab, click the **Create Instance** link.
5. Specify the application image details.  
The application image specifies the Docker image that contains the Java application. The image must be built using the JBoss EAP for OpenShift Source-to-Image (S2I) build image. If the **applicationImage** field corresponds to an imagestreamtag, any change to the image triggers an automatic upgrade of the application.

You can provide any of the following references of the JBoss EAP for OpenShift application image:

- The name of the image: mycomp/myapp
  - A tag: mycomp/myapp:1.0
  - A digest:  
mycomp/myapp:@sha256:0af38bc38be93116b6a1d86a9c78bd14cd527121970899d719baf78e
  - An imagestreamtag: my-app:latest
6. Specify the size of the application. For example:

```
spec:
  replicas:2
```

7. Configure the application environment using the **env spec**. The [environment variables](#) can come directly from values, such as POSTGRESQL\_SERVICE\_HOST or from **Secret** objects, such as POSTGRESQL\_USER. For example:

```
spec:
  env:
    - name: POSTGRESQL_SERVICE_HOST
      value: postgresql
    - name: POSTGRESQL_SERVICE_PORT
      value: '5432'
    - name: POSTGRESQL_DATABASE
      valueFrom:
        secretKeyRef:
          key: database-name
          name: postgresql
    - name: POSTGRESQL_USER
      valueFrom:
        secretKeyRef:
          key: database-user
          name: postgresql
    - name: POSTGRESQL_PASSWORD
      valueFrom:
        secretKeyRef:
          key: database-password
          name: postgresql
```

8. Complete the following optional configurations that are relevant to your application deployment:

- Specify the storage requirements for the server data directory. For more information, see [Configuring Persistent Storage for Applications](#).
- Specify the name of the **Secret** you created in **WildFlyServerSpec** to mount it as a volume in the pods running the application. For example:

```
spec:
  secrets:
  - my-secret
```

The **Secret** is mounted at `/etc/secrets/<secret name>` and each key/value is stored as a file. The name of the file is the key and the content is the value. The **Secret** is mounted as a volume inside the pod. The following example demonstrates commands that you can use to find key values:

```
$ ls /etc/secrets/my-secret/
my-key my-password
$ cat /etc/secrets/my-secret/my-key
devuser
$ cat /etc/secrets/my-secret/my-password
my-very-secure-password
```



#### NOTE

Modifying a **Secret** object might lead to project inconsistencies. Instead of modifying an existing **Secret** object, Red Hat recommends creating a new object with the same content as that of the old one. You can then update the content as required and change the reference in operator custom resource (CR) from old to new. This is considered a new CR update and the pods are reloaded.

- Specify the name of the **ConfigMap** you created in **WildFlyServerSpec** to mount it as a volume in the pods running the application. For example:

```
spec:
  configMaps:
  - my-config
```

The **ConfigMap** is mounted at `/etc/configmaps/<configmap name>` and each key/value is stored as a file. The name of the file is the key and the content is the value. The **ConfigMap** is mounted as a volume inside the pod. To find the key values:

```
$ ls /etc/configmaps/my-config/
key1 key2
$ cat /etc/configmaps/my-config/key1
value1
$ cat /etc/configmaps/my-config/key2
value2
```

**NOTE**

Modifying a **ConfigMap** might lead to project inconsistencies. Instead of modifying an existing **ConfigMap**, Red Hat recommends creating a new **ConfigMap** with the same content as that of the old one. You can then update the content as required and change the reference in operator custom resource (CR) from old to new. This is considered a new CR update and the pods are reloaded.

- If you choose to have your own standalone **ConfigMap**, provide the name of the **ConfigMap** as well as the key for the **standalone.xml** file:

```
standaloneConfigMap:
  name: clusterbench-config-map
  key: standalone-openshift.xml
```

**NOTE**

Creating a **ConfigMap** from the **standalone.xml** file is not supported in JBoss EAP 7.

- If you want to disable the default HTTP route creation in OpenShift, set **disableHTTPRoute** to **true**:

```
spec:
  disableHTTPRoute: true
```

### 7.5.1. Creating a Secret

If your application's CustomResourceDefinition (CRD) file references a **Secret**, you must create the **Secret** before deploying your application on OpenShift using the EAP operator.

#### Procedure

- To create a **Secret**:

```
$ oc create secret generic my-secret --from-literal=my-key=devuser --from-literal=my-password='my-very-secure-pasword'
```

### 7.5.2. Creating a ConfigMap

If your application's CustomResourceDefinition (CRD) file references a ConfigMap in the **spec.ConfigMaps** field, you must create the ConfigMap before deploying your application on OpenShift using the EAP operator.

#### Procedure

- To create a configmap:

```
$ oc create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2
configmap/my-config created
```

### 7.5.3. Creating a ConfigMap from a standalone.xml File

You can create your own JBoss EAP standalone configuration instead of using the one in the application image that comes from JBoss EAP for OpenShift Source-to-Image (S2I). The **standalone.xml** file must be put in a **ConfigMap** that is accessible by the operator.



#### NOTE

NOTE: Providing a **standalone.xml** file from the **ConfigMap** is not supported in JBoss EAP 7.

#### Procedure

- To create a **ConfigMap** from the **standalone.xml** file:

```
$ oc create configmap clusterbench-config-map --from-file examples/clustering/config/standalone-openshift.xml
configmap/clusterbench-config-map created
```

### 7.5.4. Configuring Persistent Storage for Applications

If your application requires persistent storage for some data, such as, transaction or messaging logs that must persist across pod restarts, configure the storage spec. If the storage spec is empty, an **EmptyDir** volume is used by each pod of the application. However, this volume does not persist after its corresponding pod is stopped.

#### Procedure

- Specify **volumeClaimTemplate** to configure resources requirements to store the JBoss EAP standalone data directory. The name of the template is derived from the name of JBoss EAP. The corresponding volume is mounted in **ReadWriteOnce** access mode.

```
spec:
  storage:
    volumeClaimTemplate:
      spec:
        resources:
          requests:
            storage: 3Gi
```

The persistent volume that meets this storage requirement is mounted on the **/eap/standalone/data** directory.

## 7.6. DEPLOYING THE RED HAT SINGLE SIGN-ON-ENABLED IMAGE BY USING EAP OPERATOR

The EAP operator helps you to deploy an EAP application image with Red Hat Single Sign-On enabled on OpenShift. To deploy the application image, configure the [environment variables and secrets](#) listed in the table.

#### Prerequisites

- You have installed the EAP operator. For more information about installing the EAP operator, see [Installing EAP operator using the web console](#) and [Installing EAP operator using the CLI](#).
- You have built the EAP application image by using the **eap74-sso-s2i** template. For information about building the EAP application image, see [Building an application image](#).

## Procedure

1. Remove the **DeploymentConfig** file, created by the **eap74-sso-s2i** template, from the location where you have built the EAP application image.
2. In the **env** field of the EAP operator's **WildFlyServer** resource, configure all the [environment variables and secrets](#).

## Example configuration

```
$ cat > my-app.yaml<<EOF

apiVersion: wildfly.org/v1alpha1
kind: WildFlyServer
metadata:
  name: my-app
spec:
  applicationImage: 'my-app:latest'
  replicas: 1

  env:
  - name: SSO_URL
    value: https://secure-sso-sso-app-demo.openshift32.example.com/auth
  - name: SSO_REALM
    value: eap-demo
  - name: SSO_PUBLIC_KEY
    value: realm-public-key
  - name: SSO_USERNAME
    value: mySsoUser
  - name: SSO_PASSWORD
    value: 6fedmL3P
  - name: SSO_SAML_KEYSTORE
    value: /etc/secret/sso-app-secret/keystore.jks
  - name: SSO_SAML_KEYSTORE_PASSWORD
    value: mykeystorepass
  - name: SSO_SAML_CERTIFICATE_NAME
    value: jboss
  - name: SSO_BEARER_ONLY
    value: true
  - name: SSO_CLIENT
    value: module-name
  - name: SSO_ENABLE_CORS
    value: true
  - name: SSO_SECRET
    value: KZ1Qylq4
  - name: SSO_DISABLE_SSL_CERTIFICATE_VALIDATION
    value: true
  - name: SSO_SAML_KEYSTORE_SECRET
    value: sso-app-secret
  - name: HTTPS_SECRET
```

```
value: eap-ssl-secret
- name: SSO_TRUSTSTORE_SECRET
value: sso-app-secret
EOF
```



#### NOTE

- Ensure that all environment variables and secrets match the image configuration.
- The value of the parameter **SSO\_URL** varies depending on the user of the OpenShift cluster.
- The EAP operator mounts the secrets in the **/etc/secret** directory, whereas the **eap74-ssso** template mounts the secrets in the **/etc** directory.

3. Save the EAP operator's **WildFlyServer** resource configuration.

## 7.7. VIEWING METRICS OF AN APPLICATION USING THE EAP OPERATOR

You can view the metrics of an application deployed on OpenShift using the EAP operator.

When your cluster administrator enables metrics monitoring in your project, the EAP operator automatically displays the metrics on the OpenShift console.

#### Prerequisites

- Your cluster administrator has enabled monitoring for your project. For more information, see [Enabling monitoring for user-defined projects](#).

#### Procedure

1. In the OpenShift Container Platform web console, navigate to **Monitoring** → **Metrics**.
2. On the **Metrics** screen, type the name of your application in the text box to select your application. The metrics for your application appear on the screen.



#### NOTE

All metrics related to JBoss EAP application server are prefixed with **jboss**. For example, **jboss\_undertow\_request\_count\_total**.

## 7.8. UNINSTALLING EAP OPERATOR USING WEB CONSOLE

To delete, or uninstall, EAP operator from your cluster, you can delete the subscription to remove it from the subscribed namespace. You can also remove the EAP operator's ClusterServiceVersion (CSV) and deployment.



#### NOTE

To ensure data consistency and safety, scale down the number of pods in your cluster to 0 before uninstalling the EAP operator.

You can uninstall the EAP operator using the web console.



### WARNING

If you decide to delete the entire **wildflyserver** definition (**oc delete wildflyserver <deployment\_name>**), then no transaction recovery process is started and the pod is terminated regardless of unfinished transactions. The unfinished work that results from this operation might block the data changes that you later initiate. The data changes for other JBoss EAP instances involved in transactional enterprise bean remote calls with this **wildflyserver** might also be blocked.

### Procedure

1. From the **Operators**→ **Installed Operators** page, select **JBoss EAP**.
2. On the right-hand side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.
3. When prompted by the **Remove Operator Subscription** window, optionally select the **Also completely remove the Operator from the selected namespace** check box if you want all components related to the installation to be removed. This removes the CSV, which in turn removes the pods, deployments, custom resource definitions (CRDs), and custom resources (CRs) associated with the operator.
4. Click **Remove**. The EAP operator stops running and no longer receives updates.

## 7.9. UNINSTALLING EAP OPERATOR USING THE CLI

To delete, or uninstall, the EAP operator from your cluster, you can delete the subscription to remove it from the subscribed namespace. You can also remove the EAP operator's ClusterServiceVersion (CSV) and deployment.



### NOTE

To ensure data consistency and safety, scale down the number of pods in your cluster to 0 before uninstalling the EAP operator.

You can uninstall the EAP operator using the command line.

When using the command line, you uninstall the operator by deleting the subscription and CSV from the target namespace.

**WARNING**

If you decide to delete the entire **wildflyserver** definition (**oc delete wildflyserver <deployment\_name>**), then no transaction recovery process is started and the pod is terminated regardless of unfinished transactions. The unfinished work that results from this operation might block the data changes that you later initiate. The data changes for other JBoss EAP instances involved in transactional enterprise bean remote calls with this **wildflyserver** might also be blocked.

**Procedure**

1. Check the current version of the EAP operator subscription in the **currentCSV** field:

```
$ oc get subscription eap-operator -n openshift-operators -o yaml | grep currentCSV
currentCSV: eap-operator.v1.0.0
```

2. Delete the EAP operator's subscription:

```
$ oc delete subscription eap-operator -n openshift-operators
subscription.operators.coreos.com "eap-operator" deleted
```

3. Delete the CSV for the EAP operator in the target namespace using the **currentCSV** value from the previous step:

```
$ oc delete clusterserviceversion eap-operator.v1.0.0 -n openshift-operators
clusterserviceversion.operators.coreos.com "eap-operator.v1.0.0" deleted
```

**7.10. EAP OPERATOR FOR SAFE TRANSACTION RECOVERY**

For certain types of transactions, EAP operator ensures data consistency before terminating your application cluster by verifying that all transactions are completed before scaling down the replicas and marking a pod as **clean** for termination.

**NOTE**

Some scenarios are not supported. For more information about the unsupported scenarios, see [Unsupported Transaction Recovery Scenarios](#).

This means that if you want to remove the deployment safely without data inconsistencies, you must first scale down the number of pods to 0, wait until all pods are terminated, and only then delete the **wildflyserver** instance.



**WARNING**

If you decide to delete the entire **wildflyserver** definition (**oc delete wildflyserver <deployment\_name>**), then no transaction recovery process is started and the pod is terminated regardless of unfinished transactions. The unfinished work that results from this operation might block the data changes that you later initiate. The data changes for other JBoss EAP instances involved in transactional enterprise bean remote calls with this **wildflyserver** might also be blocked.

When the scaledown process begins the pod state (**oc get pod <pod\_name>**) is still marked as **Running**, because the pod must complete all the unfinished transactions, including the remote enterprise beans calls that target it.

If you want to monitor the state of the scaledown process, observe the status of the **wildflyserver** instance. For more information, see [Monitoring the Scaledown Process](#). For information about pod statuses during scaledown, see [Pod Status During Scaledown](#).

### 7.10.1. StatefulSets for Stable Network Host Names

The EAP operator that manages the wildflyserver creates a **StatefulSet** as an underlying object managing the JBoss EAP pods.

A **StatefulSet** is the workload API object that manages stateful applications. It manages the deployment and scaling of a set of pods, and provides guarantees about the ordering and uniqueness of these pods.

The **StatefulSet** ensures that the pods in a cluster are named in a predefined order. It also ensures that pod termination follows the same order. For example, let us say, pod-1 has a transaction with heuristic outcome, and so is in the state of **SCALING\_DOWN\_RECOVERY\_DIRTY**. Even if pod-0 is in the state of **SCALING\_DOWN\_CLEAN**, it is not terminated before pod-1. Until pod-1 is **clean** and is terminated, pod-0 remains in the **SCALING\_DOWN\_CLEAN** state. However, even if pod-0 is in the **SCALING\_DOWN\_CLEAN** state, it does not receive any new request and is practically idle.

**NOTE**

Decreasing the replica size of the **StatefulSet** or deleting the pod itself has no effect and such changes are reverted.

### 7.10.2. Monitoring the Scaledown Process

If you want to monitor the state of the scaledown process, you must observe the status of the **wildflyserver** instance. For more information about the different pod statuses during scaledown, see [Pod Status During Scaledown](#).

**Procedure**

- To observe the state of the scaledown process:

```
oc describe wildflyserver <name>
```

- The `WildFlyServer.Status.Scalingdown Pods` and `WildFlyServer.Status.Replicas` fields shows the overall state of the active and non-active pods.
- The `Scalingdown Pods` field shows the number of pods which are about to be terminated when all the unfinished transactions are complete.
- The `WildFlyServer.Status.Replicas` field shows the current number of running pods.
- The `WildFlyServer.Spec.Replicas` field shows the number of pods in ACTIVE state.
- If there are no pods in scaledown process the numbers of pods in the `WildFlyServer.Status.Replicas` and `WildFlyServer.Spec.Replicas` fields are equal.

### 7.10.2.1. Pod Status During Scaledown

The following table describes the different pod statuses during scaledown:

**Table 7.1. Pod Status Description**

Pod Status	Description
ACTIVE	The pod is active and processing requests.
SCALING_DOWN_RECOVERY_INVESTIGATION	The pod is about to be scaled down. The scale-down process is under investigation about the state of transactions in JBoss EAP.
SCALING_DOWN_RECOVERY_DIRTY	JBoss EAP contains some incomplete transactions. The pod cannot be terminated until they are cleaned. The transaction recovery process is periodically run at JBoss EAP and it waits until the transactions are completed
SCALING_DOWN_CLEAN	The pod is processed by transaction scaled down processing and is marked as <b>clean</b> to be removed from the cluster.

### 7.10.3. Scaling Down During Transactions with Heuristic Outcomes

When the outcome of a transaction is unknown, automatic transaction recovery is impossible. You must then manually recover your transactions.

#### Prerequisites

- The status of your pod is stuck at **SCALING\_DOWN\_RECOVERY\_DIRTY**.

#### Procedure

1. Access your JBoss EAP instance using CLI.
2. Resolve all the heuristics transaction records in the transaction object store. For more information, see [Recovering Heuristic Outcomes](#) in the *Managing Transactions on JBoss EAP*.

3. Remove all records from the enterprise bean client recovery folder.
  - a. Remove all files from the pod enterprise bean client recovery directory:

```
$JBOSS_HOME/standalone/data/ejb-xa-recovery
oc exec <podname> rm -rf $JBOSS_HOME/standalone/data/ejb-xa-recovery
```

4. The status of your pod changes to **SCALING\_DOWN\_CLEAN** and the pod is terminated.

### 7.10.4. Configuring the transactions subsystem to use the JDBC storage for transaction log

In cases where the system does not provide a file system to store **transaction logs**, use the JBoss EAP S2I image to configure the JDBC object store.



#### IMPORTANT

S2I environment variables are not usable when JBoss EAP is deployed as a bootable JAR. In this case, you must create a Galleon layer or configure a CLI script to make the necessary configuration changes.

The JDBC object store can be set up with the environment variable **TX\_DATABASE\_PREFIX\_MAPPING**. This variable has the same structure as **DB\_SERVICE\_PREFIX\_MAPPING**.

#### Prerequisite

- You have created a datasource based on the value of the environment variables.
- You have ensured consistent data reads and writes permissions exist between the database and the **transaction manager** communicating over the JDBC object store. For more information see [configuring JDBC data sources](#)

#### Procedure

- Set up and configure the JDBC object store through the S2I environment variable.

#### Example

```
# Narayana JDBC objectstore configuration via s2i env variables
- name: TX_DATABASE_PREFIX_MAPPING
  value: 'PostgresJdbcObjectStore-postgresql=PG_OBJECTSTORE'
- name: POSTGRESJDBCObjectSTORE_POSTGRESQL_SERVICE_HOST
  value: 'postgresql'
- name: POSTGRESJDBCObjectSTORE_POSTGRESQL_SERVICE_PORT
  value: '5432'
- name: PG_OBJECTSTORE_JNDI
  value: 'java:jboss/datasources/PostgresJdbc'
- name: PG_OBJECTSTORE_DRIVER
  value: 'postgresql'
- name: PG_OBJECTSTORE_DATABASE
  value: 'sampledb'
- name: PG_OBJECTSTORE_USERNAME
```

```
value: 'admin'
- name: PG_OBJECTSTORE_PASSWORD
value: 'admin'
```

## Verification

- You can verify both the datasource configuration and transaction subsystem configuration by checking the **standalone-openshift.xml** configuration file **oc rsh <podname> cat /opt/eap/standalone/configuration/standalone-openshift.xml**.

Expected output:

```
<datasource jta="false" jndi-name="java:jboss/datasources/PostgresJdbcObjectStore" pool-
name="postgresjdbcobjectstore_postgresqlObjectStorePool"
  enabled="true" use-java-context="true" statistics-enabled="{wildfly.datasources.statistics-
enabled:${wildfly.statistics-enabled:false}}">
  <connection-url>jdbc:postgresql://postgresql:5432/sampledb</connection-url>
  <driver>postgresql</driver>
  <security>
    <user-name>admin</user-name>
    <password>admin</password>
  </security>
</datasource>

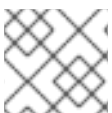
<!-- under subsystem urn:jboss:domain:transactions -->
<jdbc-store datasource-jndi-name="java:jboss/datasources/PostgresJdbcObjectStore">
  <!-- the pod name was named transactions-xa-0 -->
  <action table-prefix="ostransactionsxa0"/>
  <communication table-prefix="ostransactionsxa0"/>
  <state table-prefix="ostransactionsxa0"/>
</jdbc-store>
```

## Additional resources

- For more information about creating datasources by using either the management console or the management CLI, see [Creating Datasources](#) in the JBoss EAP *Configuration Guide*.

## 7.11. AUTOMATICALLY SCALING PODS WITH THE HORIZONTAL POD AUTOSCALER HPA

With EAP operator, you can use a horizontal pod autoscaler HPA to automatically increase or decrease the scale of an EAP application based on metrics collected from the pods that belong to that EAP application.



### NOTE

Using HPA ensures that transaction recovery is still handled when a pod is scaled down.

## Procedure

- Configure the resources:

```
apiVersion: wildfly.org/v1alpha1
kind: WildFlyServer
```

```

metadata:
  name: eap-helloworld
spec:
  applicationImage: 'eap-helloworld:latest'
  replicas: 1
  resources:
    limits:
      cpu: 500m
      memory: 2Gi
    requests:
      cpu: 100m
      memory: 1Gi

```



### IMPORTANT

You must specify the resource limits and requests for containers in a pod for autoscaling to work as expected.

2. Create the Horizontal pod autoscaler:

```
oc autoscale wildflyserver/eap-helloworld --cpu-percent=50 --min=1 --max=10
```

### Verification

- You can verify the HPA behavior by checking the replicas. The number of replicas increase or decrease depending on the increase or decrease of the workload.

```
oc get hpa -w
```

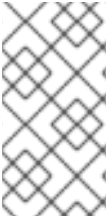
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
eap-helloworld	WildFlyServer/eap-helloworld	217%/50%	1	10	1	4s
eap-helloworld	WildFlyServer/eap-helloworld	217%/50%	1	10	4	17s
eap-helloworld	WildFlyServer/eap-helloworld	133%/50%	1	10	8	32s
eap-helloworld	WildFlyServer/eap-helloworld	133%/50%	1	10	10	47s
eap-helloworld	WildFlyServer/eap-helloworld	139%/50%	1	10	10	62s
eap-helloworld	WildFlyServer/eap-helloworld	180%/50%	1	10	10	92s
eap-helloworld	WildFlyServer/eap-helloworld	133%/50%	1	10	10	2m2s

### Additional resources

- [https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.10/html-single/nodes/index#nodes-pods-autoscaling](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.10/html-single/nodes/index#nodes-pods-autoscaling)

## 7.12. JAKARTA ENTERPRISE BEANS REMOTING ON OPENSIFT

For JBoss EAP to work correctly with enterprise bean remoting calls between different JBoss EAP clusters on OpenShift, you must understand the enterprise bean remoting configuration options on OpenShift.

**NOTE**

When deploying on OpenShift, consider the use of the EAP operator. The EAP operator uses **StatefulSet** for the appropriate handling of enterprise bean remoting and transaction recovery processing. The **StatefulSet** ensures persistent storage and network hostname stability even after pods are restarted.

Network hostname stability is required when the JBoss EAP instance is contacted using an enterprise bean remote call with transaction propagation. The JBoss EAP instance must be reachable under the same hostname even if the pod restarts. The transaction manager, which is a stateful component, binds the persisted transaction data to a particular JBoss EAP instance. Because the transaction log is bound to a specific JBoss EAP instance, it must be completed in the same instance.

To prevent data loss when the JDBC transaction log store is used, make sure your database provides data-consistent reads and writes. Consistent data reads and writes are important when the database is scaled horizontally with multiple instances.

An enterprise bean remote caller has two options to configure the remote calls:

- Define a remote outbound connection. For more information, see [Configuring a Remote Outbound Connection](#).
- Use a programmatic JNDI lookup for the bean at the remote server. For more information, see [Using Remote Jakarta Enterprise Beans Clients](#).

You must reconfigure the value representing the address of the target node depending on the enterprise bean remote call configuration method.

**NOTE**

The name of the target enterprise bean for the remote call must be the DNS address of the first pod.

The **StatefulSet** behaviour depends on the ordering of the pods. The pods are named in a predefined order. For example, if you scale your application to three replicas, your pods have names such as **eap-server-0**, **eap-server-1**, and **eap-server-2**.

The EAP operator also uses a [headless service](#) that ensures a specific DNS hostname is assigned to the pod. If the application uses the EAP operator, a headless service is created with a name such as **eap-server-headless**. In this case, the DNS name of the first pod is **eap-server-0.eap-server-headless**.

The use of the hostname **eap-server-0.eap-server-headless** ensures that the enterprise bean call reaches any EAP instance connected to the cluster. A bootstrap connection is used to initialize the Jakarta Enterprise Beans client, which gathers the structure of the EAP cluster as the next step.

### 7.12.1. Configuring Jakarta Enterprise Beans on OpenShift

You must configure the JBoss EAP servers that act as callers for enterprise bean remoting. The target server must configure a user with permission to receive the enterprise bean remote calls.

#### Prerequisites

- You have used the EAP operator and the supported JBoss EAP for OpenShift S2I image for deploying and managing the JBoss EAP application instances on OpenShift.

- The clustering is set correctly. For more information about JBoss EAP clustering, see the [Clustering](#) section.

## Procedure

1. Create a user in the target server with permission to receive the enterprise bean remote calls:

```
$JBOSS_HOME/bin/add-user.sh
```

2. Configure the caller JBoss EAP application server.
  - a. Create the **eap-config.xml** file in **\$JBOSS\_HOME/standalone/configuration** using the custom configuration functionality. For more information, see [Custom Configuration](#).
  - b. Configure the caller JBoss EAP application server with the **wildfly.config.url** property:

```
JAVA_OPTS_APPEND="-
Dwildfly.config.url=$JBOSS_HOME/standalone/configuration/eap-config.xml"
```



### NOTE

If you use the following example for your configuration, replace the **>>PASTE\_...\_HERE<<** with username and password you configured.

## Example Configuration

```
<configuration>
  <authentication-client xmlns="urn:elytron:1.0">
    <authentication-rules>
      <rule use-configuration="jta">
        <match-abstract-type name="jta" authority="jboss" />
      </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="jta">
        <sasl-mechanism-selector selector="DIGEST-MD5" />
        <providers>
          <use-service-loader />
        </providers>
        <set-user-name name="PASTE_USER_NAME_HERE" />
        <credentials>
          <clear-password password="PASTE_PASSWORD_HERE" />
        </credentials>
        <set-mechanism-realm name="ApplicationRealm" />
      </configuration>
    </authentication-configurations>
  </authentication-client>
</configuration>
```

## CHAPTER 8. REFERENCE INFORMATION



### NOTE

The content in this section is derived from the engineering documentation for this image. It is provided for reference as it can be useful for development purposes and for testing beyond the scope of the product documentation.

### 8.1. PERSISTENT TEMPLATES

The JBoss EAP database templates, which deploy JBoss EAP and database pods, have both ephemeral and persistent variations.

Persistent templates include an environment variable to provision a persistent volume claim, which binds with an available persistent volume to be used as a storage volume for the JBoss EAP for OpenShift deployment. Information, such as timer schema, log handling, or data updates, is stored on the storage volume, rather than in ephemeral container memory. This information persists if the pod goes down for any reason, such as project upgrade, deployment rollback, or an unexpected error.

Without a persistent storage volume for the deployment, this information is stored in the container memory only, and is lost if the pod goes down for any reason.

For example, an EE timer backed by persistent storage continues to run if the pod is restarted. Any events triggered by the timer during the restart process are enacted when the application is running again.

Conversely, if the EE timer is running in the container memory, the timer status is lost if the pod is restarted, and starts from the beginning when the pod is running again.

### 8.2. INFORMATION ENVIRONMENT VARIABLES

The following environment variables are designed to provide information to the image and should not be modified by the user:

**Table 8.1. Information Environment Variables**

Variable Name	Description and Value
JBOSS_IMAGE_NAME	<p>The image names.</p> <p>Values:</p> <ul style="list-style-type: none"> <li>● <b>jboss-eap-7/eap74-openjdk8-openshift-rhel7</b> (JDK 8 / RHEL 7)</li> <li>● <b>jboss-eap-7/eap74-openjdk11-openshift-rhel8</b> (JDK 11 / RHEL 8)</li> </ul>



Variable Name	Description and Value
JBOSS_IMAGE_VERSION	<p>The image version.</p> <p>Value: This is the image version number. See the Red Hat Container Catalog for the latest values:</p> <ul style="list-style-type: none"> <li>• <a href="#">JDK 8 / RHEL 7</a></li> <li>• <a href="#">JDK 11 / RHEL 8</a></li> </ul>
JBOSS_MODULES_SYSTEM_PKGS	<p>A comma-separated list of JBoss EAP system modules packages that are available to applications.</p> <p>Value: <b>jdk.nashorn.api</b></p>
STI_BUILDER	<p>Provides OpenShift S2I support for <b>jee</b> project types.</p> <p>Value: <b>jee</b></p>

### 8.3. CONFIGURATION ENVIRONMENT VARIABLES

You can configure the following environment variables to adjust the image without requiring a rebuild.



#### NOTE

See the [JBoss EAP documentation](#) for other environment variables that are not listed here.

Table 8.2. Configuration environment variables

Variable Name	Description
AB_JOLOKIA_AUTH_OPENSIFT	<p>Switch on client authentication for OpenShift TLS communication. The value of this parameter can be <b>true</b>, <b>false</b>, or a relative distinguished name, which must be contained in a presented client's certificate. The default CA cert is set to <b>/var/run/secrets/kubernetes.io/serviceaccount/ca.crt</b>.</p> <ul style="list-style-type: none"> <li>• Set to <b>false</b> to disable client authentication for OpenShift TLS communication.</li> <li>• Set to <b>true</b> to enable client authentication for OpenShift TLS communication using the default CA certificate and client principal.</li> <li>• Set to a relative distinguished name, for example <b>cn=someSystem</b>, to enable client authentication for OpenShift TLS communication but override the client principal. This distinguished name must be contained in a presented client's certificate.</li> </ul>

Variable Name	Description
AB_JOLOKIA_CONFIG	<p>If set, uses this fully qualified file path for the Jolokia JVM agent properties, which are described in the <a href="#">Jolokia reference documentation</a>. If you set your own Jolokia properties config file, the rest of the Jolokia settings in this document are ignored.</p> <p>If not set, <code>/opt/jolokia/etc/jolokia.properties</code> is created using the settings as defined in the Jolokia reference documentation.</p> <p>Example value: <code>/opt/jolokia/custom.properties</code></p>
AB_JOLOKIA_DISCOVERY_ENABLED	<p>Enable Jolokia discovery.</p> <p>Defaults to <b>false</b>.</p>
AB_JOLOKIA_HOST	<p>Host address to bind to.</p> <p>Defaults to <b>0.0.0.0</b>.</p> <p>Example value: <b>127.0.0.1</b></p>
AB_JOLOKIA_HTTPS	<p>Switch on secure communication with HTTPS.</p> <p>By default self-signed server certificates are generated if no <b>serverCert</b> configuration is given in <b>AB_JOLOKIA_OPTS</b>.</p> <p>Example value: <b>true</b></p>
AB_JOLOKIA_ID	<p>Agent ID to use.</p> <p>The default value is the <b>\$HOSTNAME</b>, which is the container id.</p> <p>Example value: <b>openjdk-app-1-xqlsj</b></p>
AB_JOLOKIA_OFF	<p>If set to <b>true</b>, disables activation of Jolokia, which echos an empty value.</p> <p>Jolokia is enabled by default.</p>
AB_JOLOKIA_OPTS	<p>Additional options to be appended to the agent configuration. They should be given in the format <b>key=value, key=value, ...</b>.</p> <p>Example value: <b>backlog=20</b></p>
AB_JOLOKIA_PASSWORD	<p>The password for basic authentication.</p> <p>By default, authentication is switched off.</p> <p>Example value: <b>mypassword</b></p>

Variable Name	Description
AB_JOLOKIA_PASSWORD_RANDOM	<p>Determines if a random <b>AB_JOLOKIA_PASSWORD</b> should be generated.</p> <p>Set to <b>true</b> to generate a random password. The generated value is saved in the <code>/opt/jolokia/etc/jolokia.pw</code> file.</p>
AB_JOLOKIA_PORT	<p>The port to listen to.</p> <p>Defaults to <b>8778</b>.</p> <p>Example value: <b>5432</b></p>
AB_JOLOKIA_USER	<p>The name of the user to use for basic authentication.</p> <p>Defaults to <b>jolokia</b>.</p> <p>Example value: <b>myusername</b></p>
AB_PROMETHEUS_ENABLE	<p>If set to <b>true</b>, this variable activates the <b>jmx-exporter</b> java agent that exposes Prometheus format metrics. Default is set to <b>false</b>.</p> <div data-bbox="687 1043 798 1301" style="float: left; margin-right: 10px;">  </div> <p><b>NOTE</b></p> <p>The MicroProfile Metrics subsystem is the preferred method to expose data in the Prometheus format. For more information about the MicroProfile Metrics subsystem, see <a href="#">Eclipse MicroProfile</a> in the <i>Configuration Guide</i> for JBoss EAP.</p>
AB_PROMETHEUS_JMX_EXPORTER_CONFIG	<p>The path within the container to a user-specified <b>configuration.yaml</b> for the <b>jmx-exporter</b> agent to use instead of the default <b>configuration.yaml</b> file. To find out more about the <b>S2I</b> mechanism to incorporate additional configuration files, see <a href="#">S2I Artifacts</a>.</p>
AB_PROMETHEUS_JMX_EXPORTER_PORT	<p>The port on which the <b>jmx-exporter</b> agent listens for scrapes from the Prometheus server. Default is <b>9799</b>. The agent listens on <b>localhost</b>. Metrics can be made available outside of the container by configuring the <b>DeploymentConfig</b> API for the application to include the service exposing this endpoint.</p>
CLI_GRACEFUL_SHUTDOWN	<p>If set to any non-zero length value, the image will prevent shutdown with the <b>TERM</b> signal and will require execution of the <b>shutdown</b> command using the JBoss EAP management CLI.</p> <p>Example value: <b>true</b></p>

Variable Name	Description
CONTAINER_HEAP_PERCENT	<p>Set the maximum Java heap size, as a percentage of available container memory.</p> <p>Example value: <b>0.5</b></p>
CUSTOM_INSTALL_DIRECTORIES	<p>A list of comma-separated directories used for installation and configuration of artifacts for the image during the S2I process.</p> <p>Example value: <b>custom,shared</b></p>
DEFAULT_JMS_CONNECTION_FACTORY	<p>This value is used to specify the default JNDI binding for the Jakarta Messaging connection factory, for example <b>jms-connection-factory='java:jboss/DefaultJMSConnectionFactory'</b>.</p> <p>Example value: <b>java:jboss/DefaultJMSConnectionFactory</b></p>
DISABLE_EMBEDDED_JMS_BROKER	<p>The use of an embedded messaging broker in OpenShift containers is deprecated. Support for an embedded broker will be removed in a future release.</p> <p>If the following conditions are true, a warning is logged.</p> <ul style="list-style-type: none"> <li>● A container is configured to use an embedded messaging broker.</li> <li>● A remote broker is not configured for the container.</li> <li>● This variable is not set or is set with a value of <b>false</b>.</li> </ul> <p>If this variable is included with the value set to <b>true</b>, the embedded messaging broker is disabled, and no warning is logged.</p> <p>Include this variable set to <b>true</b> for any container that is not configured with remote messaging destinations.</p>
ENABLE_ACCESS_LOG	<p>Enable logging of access messages to the standard output channel.</p> <p>Logging of access messages is implemented using following methods:</p> <ul style="list-style-type: none"> <li>● The JBoss EAP 6.4 OpenShift image uses a custom JBoss Web Access Log Valve.</li> <li>● The JBoss EAP for OpenShift image uses the Undertow <a href="#">AccessLogHandler</a>.</li> </ul> <p>Defaults to <b>false</b>.</p>

Variable Name	Description
INITIAL_HEAP_PERCENT	<p>Set the initial Java heap size, as a percentage of the maximum heap size.</p> <p>Example value: <b>0.5</b></p>
JAVA_OPTS_APPEND	<p>Server startup options.</p> <p>Example value: <b>-Dfoo=bar</b></p>
JBOSS_MODULES_SYSTEM_PKGS_APPEND	<p>A comma-separated list of package names that will be appended to the <b>JBOSS_MODULES_SYSTEM_PKGS</b> environment variable.</p> <p>Example value: <b>org.jboss.byteman</b></p>
JGROUPS_CLUSTER_PASSWORD	<p>Password used to authenticate the node so it is allowed to join the JGroups cluster. <b>Required</b>, when using <b>ASYM_ENCRYPT</b> JGroups cluster traffic encryption protocol. If not set, authentication is disabled, cluster communication is not encrypted and a warning is issued. Optional, when using <b>SYM_ENCRYPT</b> JGroups cluster traffic encryption protocol.</p> <p>Example value: <b>mypassword</b></p>
JGROUPS_ENCRYPT_KEYSTORE	<p>Name of the keystore file within the secret specified via <b>JGROUPS_ENCRYPT_SECRET</b> variable, when using <b>SYM_ENCRYPT</b> JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.</p> <p>Example value: <b>jgroups.jceks</b></p>
JGROUPS_ENCRYPT_KEYSTORE_DIR	<p>Directory path of the keystore file within the secret specified via <b>JGROUPS_ENCRYPT_SECRET</b> variable, when using <b>SYM_ENCRYPT</b> JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.</p> <p>Example value: <b>/etc/jgroups-encrypt-secret-volume</b></p>
JGROUPS_ENCRYPT_NAME	<p>Name associated with the server's certificate, when using <b>SYM_ENCRYPT</b> JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.</p> <p>Example value: <b>jgroups</b></p>

Variable Name	Description
JGROUPS_ENCRYPT_PASSWORD	<p>Password used to access the keystore and the certificate, when using <b>SYM_ENCRYPT</b> JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.</p> <p>Example value: <b>mypassword</b></p>
JGROUPS_ENCRYPT_PROTOCOL	<p>JGroups protocol to use for encryption of cluster traffic. Can be either <b>SYM_ENCRYPT</b> or <b>ASYM_ENCRYPT</b>.</p> <p>Defaults to <b>SYM_ENCRYPT</b>.</p> <p>Example value: <b>ASYM_ENCRYPT</b></p>
JGROUPS_ENCRYPT_SECRET	<p>Name of the secret that contains the <b>JGroups keystore</b> file used for securing the JGroups communications when using <b>SYM_ENCRYPT</b> JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.</p> <p>Example value: <b>eap7-app-secret</b></p>
JGROUPS_PING_PROTOCOL	<p>JGroups protocol to use for node discovery. Can be either <b>dns.DNS_PING</b> or <b>kubernetes.KUBE_PING</b>.</p>
MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION	<p>For backwards compatibility, set to <b>true</b> to use <b>MyQueue</b> and <b>MyTopic</b> as physical destination name defaults instead of <b>queue/MyQueue</b> and <b>topic/MyTopic</b>.</p>
OPENSIFT_DNS_PING_SERVICE_NAME	<p>Name of the service exposing the ping port on the servers for the DNS discovery mechanism.</p> <p>Example value: <b>eap-app-ping</b></p>
OPENSIFT_DNS_PING_SERVICE_PORT	<p>The port number of the ping port for the DNS discovery mechanism. If not specified, an attempt is made to discover the port number from the SRV records for the service, otherwise the default <b>8888</b> is used.</p> <p>Defaults to <b>8888</b>.</p>
OPENSIFT_KUBE_PING_LABELS	<p>Clustering labels selector for the Kubernetes discovery mechanism.</p> <p>Example value: <b>app=eap-app</b></p>

Variable Name	Description
OPENSIFT_KUBE_PING_NAMESPACE	Clustering project namespace for the Kubernetes discovery mechanism.  Example value: <b>myproject</b>
SCRIPT_DEBUG	If set to <b>true</b> , ensures that the Bash scripts are executed with the <b>-x</b> option, printing the commands and their arguments as they are executed.

## 8.4. APPLICATION TEMPLATES

Table 8.3. Application Templates

Variable Name	Description
AUTO_DEPLOY_EXPLODED	Controls whether exploded deployment content should be automatically deployed.  Example value: <b>false</b>

## 8.5. EXPOSED PORTS

Table 8.4. Exposed Ports

Port Number	Description
8443	HTTPS
8778	Jolokia Monitoring

## 8.6. DATASOURCES

Datasources are automatically created based on the value of some of the environment variables.

The most important environment variable is **DB\_SERVICE\_PREFIX\_MAPPING**, as it 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.

### 8.6.1. JNDI Mappings for Datasources

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.



#### NOTE

The first part (before the equal sign) of the **DB\_SERVICE\_PREFIX\_MAPPING** should be lowercase.

The **DATABASETYPE** determines the driver for the datasource.

For more information about configuring a driver, see [Modules, Drivers, and Generic Deployments](#). The JDK 8 image has drivers for **postgresql** and **mysql** configured by default.



#### WARNING

Do not use any special characters for the **POOLNAME** parameter.



#### DATABASE DRIVERS

Support for using the Red Hat-provided internal datasource drivers with the JBoss EAP for OpenShift image is now deprecated. Red Hat recommends that you use JDBC drivers obtained from your database vendor for your JBoss EAP applications.

The following internal datasources are no longer provided with the JBoss EAP for OpenShift image:

- MySQL
- PostgreSQL

For more information about installing drivers, see [Modules, Drivers, and Generic Deployments](#).

For more information on configuring JDBC drivers with JBoss EAP, see [JDBC drivers](#) in the JBoss EAP *Configuration Guide*.

Note that you can also create a custom layer to install these drivers and datasources if you want to add them to a provisioned server.

#### 8.6.1.1. Datasource Configuration Environment Variables

To configure other datasource properties, use the following environment variables.



#### IMPORTANT

Be sure to replace the values for **POOLNAME**, **DATABASETYPE**, and **PREFIX** in the following variable names with the appropriate values. These replaceable values are described in this section and in the [Datasources](#) section.



Variable Name	Description
<code>POOLNAME_DATABASETYPE_SERVICE_HOST</code>	<p>Defines the database server's host name or IP address to be used in the datasource's <b>connection-url</b> property.</p> <p>Example value: <b>192.168.1.3</b></p>
<code>POOLNAME_DATABASETYPE_SERVICE_PORT</code>	<p>Defines the database server's port for the datasource.</p> <p>Example value: <b>5432</b></p>
<code>PREFIX_BACKGROUND_VALIDATION</code>	<p>When set to <b>true</b> database connections are validated periodically in a background thread prior to use. Defaults to <b>false</b>, meaning the <b>validate-on-match</b> method is enabled by default instead.</p>
<code>PREFIX_BACKGROUND_VALIDATION_MILLIS</code>	<p>Specifies frequency of the validation, in milliseconds, when the <b>background-validation</b> database connection validation mechanism is enabled (<b>PREFIX_BACKGROUND_VALIDATION</b> variable is set to <b>true</b>). Defaults to <b>10000</b>.</p>
<code>PREFIX_CONNECTION_CHECKER</code>	<p>Specifies a connection checker class that is used to validate connections for the particular database in use.</p> <p>Example value: <b>org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker</b></p>
<code>PREFIX_DATABASE</code>	<p>Defines the database name for the datasource.</p> <p>Example value: <b>myDatabase</b></p>
<code>PREFIX_DRIVER</code>	<p>Defines Java database driver for the datasource.</p> <p>Example value: <b>postgresql</b></p>
<code>PREFIX_EXCEPTION_SORTER</code>	<p>Specifies the exception sorter class that is used to properly detect and clean up after fatal database connection exceptions.</p> <p>Example value: <b>org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter</b></p>
<code>PREFIX_JNDI</code>	<p>Defines the JNDI name for the datasource. Defaults to <b>java:jboss/datasources/<i>POOLNAME_DATABASETYPE</i></b>, where <b>POOLNAME</b> and <b>DATABASETYPE</b> are taken from the triplet described above. This setting is useful if you want to override the default generated JNDI name.</p> <p>Example value: <b>java:jboss/datasources/test-postgresql</b></p>

Variable Name	Description
<i>PREFIX_JTA</i>	Defines Jakarta Transactions option for the non-XA datasource. The XA datasources are already Jakarta Transactions capable by default.  Defaults to <b>true</b> .
<i>PREFIX_MAX_POOL_SIZE</i>	Defines the maximum pool size option for the datasource.  Example value: <b>20</b>
<i>PREFIX_MIN_POOL_SIZE</i>	Defines the minimum pool size option for the datasource.  Example value: <b>1</b>
<i>PREFIX_NONXA</i>	Defines the datasource as a non-XA datasource. Defaults to <b>false</b> .
<i>PREFIX_PASSWORD</i>	Defines the password for the datasource.  Example value: <b>password</b>
<i>PREFIX_TX_ISOLATION</i>	Defines the java.sql.Connection transaction isolation level for the datasource.  Example value: <b>TRANSACTION_READ_UNCOMMITTED</b>
<i>PREFIX_URL</i>	Defines connection URL for the datasource.  Example value: <b>jdbc:postgresql://localhost:5432/postgresdb</b>
<i>PREFIX_USERNAME</i>	Defines the username for the datasource.  Example value: <b>admin</b>

When running this image in OpenShift, the ***POOLNAME\_DATABASETYPESERVICE\_HOST*** and ***POOLNAME\_DATABASETYPESERVICE\_PORT*** environment variables are set up automatically from the database service definition in the OpenShift application template, while the others are configured in the template directly as **env** entries in container definitions under each pod template.

### 8.6.1.2. Examples

These examples show how value of the **DB\_SERVICE\_PREFIX\_MAPPING** environment variable influences datasource creation.

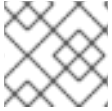
#### 8.6.1.2.1. Single Mapping

Consider value **test-postgresql=TEST**.

This creates a datasource with **java:jboss/datasources/test\_postgresql** name. Additionally, all the required settings like password and username are expected to be provided as environment variables with the **TEST\_** prefix, for example **TEST\_USERNAME** and **TEST\_PASSWORD**.

### 8.6.1.2.2. Multiple Mappings

You can specify multiple datasource mappings.



#### NOTE

Always separate multiple datasource mappings with a comma.

Consider the following value for the **DB\_SERVICE\_PREFIX\_MAPPING** environment variable: **cloud-postgresql=CLOUD,test-mysql=TEST\_MYSQL**.

This creates the following two datasources:

1. **java:jboss/datasources/test\_mysql**
2. **java:jboss/datasources/cloud\_postgresql**

Then you can use **TEST\_MYSQL** prefix for configuring things like the username and password for the MySQL datasource, for example **TEST\_MYSQL\_USERNAME**. And for the PostgreSQL datasource, use the **CLOUD\_** prefix, for example **CLOUD\_USERNAME**.

## 8.7. CLUSTERING

### 8.7.1. Configuring a JGroups Discovery Mechanism

To enable JBoss EAP clustering on OpenShift, configure the JGroups protocol stack in your JBoss EAP configuration to use either the **kubernetes.KUBE\_PING** or the **dns.DNS\_PING** discovery mechanism.

Although you can use a custom **standalone-openshift.xml** configuration file, it is recommended that you [use environment variables](#) to configure JGroups in your image build.

The instructions below use environment variables to configure the discovery mechanism for the JBoss EAP for OpenShift image.



#### IMPORTANT

If you use one of the available application templates to deploy an application on top of the JBoss EAP for OpenShift image, the default discovery mechanism is **dns.DNS\_PING**.

The **dns.DNS\_PING** and **kubernetes.KUBE\_PING** discovery mechanisms are not compatible with each other. It is not possible to form a supercluster out of two independent child clusters, with one using the **dns.DNS\_PING** mechanism for discovery and the other using the **kubernetes.KUBE\_PING** mechanism. Similarly, when performing a rolling upgrade, the discovery mechanism needs to be identical for both the source and the target clusters.

#### 8.7.1.1. Configuring KUBE\_PING

To use the **KUBE\_PING** JGroups discovery mechanism:

1. The JGroups protocol stack must be configured to use **KUBE\_PING** as the discovery mechanism.  
You can do this by setting the **JGROUPS\_PING\_PROTOCOL** environment variable to **kubernetes.KUBE\_PING**:

```
JGROUPS_PING_PROTOCOL=kubernetes.KUBE_PING
```

2. The **KUBERNETES\_NAMESPACE** environment variable must be set to your OpenShift project name. If not set, the server behaves as a single-node cluster (a "cluster of one"). For example:

```
KUBERNETES_NAMESPACE=PROJECT_NAME
```

3. The **KUBERNETES\_LABELS** environment variable should be set. This should match the [label set at the service level](#). If not set, pods outside of your application (albeit in your namespace) will try to join. For example:

```
KUBERNETES_LABELS=application=APP_NAME
```

4. Authorization must be granted to the service account the pod is running under to be allowed to access Kubernetes' REST API. This is done using the OpenShift CLI. The following example uses the **default** service account in the current project's namespace:

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default -n $(oc project -q)
```

Using the **eap-service-account** in the project namespace:

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):eap-service-account -n $(oc project -q)
```



#### NOTE

See [Prepare OpenShift for Application Deployment](#) for more information on adding policies to service accounts.

### 8.7.1.2. Configuring DNS\_PING

To use the **DNS\_PING** JGroups discovery mechanism:

1. The JGroups protocol stack must be configured to use **DNS\_PING** as the discovery mechanism.  
You can do this by setting the **JGROUPS\_PING\_PROTOCOL** environment variable to **dns.DNS\_PING**:

```
JGROUPS_PING_PROTOCOL=dns.DNS_PING
```

2. The **OPENSIFT\_DNS\_PING\_SERVICE\_NAME** environment variable must be set to the name of the ping service for the cluster.

```
OPENSIFT_DNS_PING_SERVICE_NAME=PING_SERVICE_NAME
```

3. The **OPENSIFT\_DNS\_PING\_SERVICE\_PORT** environment variable should be set to the port number on which the ping service is exposed. The **DNS\_PING** protocol attempts to discern the port from the SRV records, otherwise it defaults to **8888**.

```
OPENSIFT_DNS_PING_SERVICE_PORT=PING_PORT
```

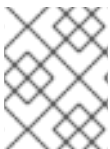
4. A ping service which exposes the ping port must be defined. This service should be headless (ClusterIP=None) and must have the following:
  - a. The port must be named.
  - b. The service must be annotated with the **service.alpha.kubernetes.io/tolerate-unready-endpoints** and the **publishNotReadyAddresses** properties, both set to **true**.



#### NOTE

- Use both the **service.alpha.kubernetes.io/tolerate-unready-endpoints** and the **publishNotReadyAddresses** properties to ensure that the ping service works in both the older and newer OpenShift releases.
- Omitting these annotations result in each node forming its own "cluster of one" during startup. Each node then merges its cluster into the other nodes' clusters after startup, because the other nodes are not detected until after they have started.

```
kind: Service
apiVersion: v1
spec:
  publishNotReadyAddresses: true
  clusterIP: None
  ports:
  - name: ping
    port: 8888
  selector:
    deploymentConfig: eap-app
metadata:
  name: eap-app-ping
  annotations:
    service.alpha.kubernetes.io/tolerate-unready-endpoints: "true"
    description: "The JGroups ping port for clustering."
```



#### NOTE

**DNS\_PING** does not require any modifications to the service account and works using the default permissions.

### 8.7.2. Configuring JGroups to Encrypt Cluster Traffic

To encrypt cluster traffic for JBoss EAP on OpenShift, you must configure the JGroups protocol stack in your JBoss EAP configuration to use either the **SYM\_ENCRYPT** or **ASYM\_ENCRYPT** protocol.

Although you can use a custom **standalone-openshift.xml** configuration file, it is recommended that you [use environment variables](#) to configure JGroups in your image build.

The instructions below use environment variables to configure the protocol for cluster traffic encryption for the JBoss EAP for OpenShift image.



## IMPORTANT

The **SYM\_ENCRYPT** and **ASYM\_ENCRYPT** protocols are not compatible with each other. It is not possible to form a supercluster out of two independent child clusters, with one using the **SYM\_ENCRYPT** protocol for the encryption of cluster traffic and the other using the **ASYM\_ENCRYPT** protocol. Similarly, when performing a rolling upgrade, the protocol needs to be identical for both the source and the target clusters.

### 8.7.2.1. Configuring SYM\_ENCRYPT

To use the **SYM\_ENCRYPT** protocol to encrypt JGroups cluster traffic:

1. The JGroups protocol stack must be configured to use **SYM\_ENCRYPT** as the encryption protocol.

You can do this by setting the **JGROUPS\_ENCRYPT\_PROTOCOL** environment variable to **SYM\_ENCRYPT**:

```
JGROUPS_ENCRYPT_PROTOCOL=SYM_ENCRYPT
```

2. The **JGROUPS\_ENCRYPT\_SECRET** environment variable must be set to the name of the secret containing the JGroups keystore file used for securing the JGroups communications. If not set, cluster communication is not encrypted and a warning is issued. For example:

```
JGROUPS_ENCRYPT_SECRET=eap7-app-secret
```

3. The **JGROUPS\_ENCRYPT\_KEYSTORE\_DIR** environment variable must be set to the directory path of the keystore file within the secret specified via **JGROUPS\_ENCRYPT\_SECRET** variable. If not set, cluster communication is not encrypted and a warning is issued. For example:

```
JGROUPS_ENCRYPT_KEYSTORE_DIR=/etc/jgroups-encrypt-secret-volume
```

4. The **JGROUPS\_ENCRYPT\_KEYSTORE** environment variable must be set to the name of the keystore file within the secret specified via **JGROUPS\_ENCRYPT\_SECRET** variable. If not set, cluster communication is not encrypted and a warning is issued. For example:

```
JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
```

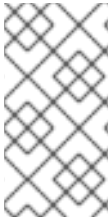
5. The **JGROUPS\_ENCRYPT\_NAME** environment variable must be set to the name associated with the server's certificate. If not set, cluster communication is not encrypted and a warning is issued. For example:

```
JGROUPS_ENCRYPT_NAME=jgroups
```

6. The **JGROUPS\_ENCRYPT\_PASSWORD** environment variable must be set to the password used to access the keystore and the certificate. If not set, cluster communication is not encrypted and a warning is issued. For example:

```
JGROUPS_ENCRYPT_PASSWORD=mypassword
```

### 8.7.2.2. Configuring ASYM\_ENCRYPT



#### NOTE

JBoss EAP 7.4 includes a new version of the **ASYM\_ENCRYPT** protocol. The previous version of the protocol is deprecated. If you specify the **JGROUPS\_CLUSTER\_PASSWORD** environment variable, the deprecated version of the protocol is used and a warning is printed in the pod log.

To use the **ASYM\_ENCRYPT** protocol to encrypt JGroups cluster traffic, specify **ASYM\_ENCRYPT** as the encryption protocol, and configure it to use a keystore configured in the **elytron** subsystem.

```
-e JGROUPS_ENCRYPT_PROTOCOL="ASYM_ENCRYPT" \
-e JGROUPS_ENCRYPT_SECRET="encrypt_secret" \
-e JGROUPS_ENCRYPT_NAME="encrypt_name" \
-e JGROUPS_ENCRYPT_PASSWORD="encrypt_password" \
-e JGROUPS_ENCRYPT_KEYSTORE="encrypt_keystore" \
-e JGROUPS_CLUSTER_PASSWORD="cluster_password"
```

### 8.7.3. Considerations for scaling up pods

Based on the discovery mechanism in JGroups, a starting node searches for an existing cluster coordinator node. If no coordinator node is found within a given timeout, the starting node assumes that it is the first member and takes up the coordinator status.

When multiple nodes start concurrently, they all make this assumption of being the first member, leading to the creation of a split cluster with multiple partitions. For example, scaling up from 0 to 2 pods using the **DeploymentConfig** API may lead to the creation of a split cluster. To avoid this situation, you need to start the first pod and then scale up to the required number of pods.



#### NOTE

By default, the EAP Operator uses the **StatefulSet** API, which starts pods in order, that is, one by one, preventing the creation of split clusters.

## 8.8. HEALTH CHECKS

The JBoss EAP for OpenShift image utilizes the [liveness and readiness probes](#) included in OpenShift by default. In addition, this image includes [Eclipse MicroProfile Health](#), as discussed in the *Configuration Guide*.

The following table demonstrates the values necessary for these health checks to pass. If the status is anything other than the values found below, then the check is failed and the image is restarted per the image's restart policy.

**Table 8.5. Liveness and Readiness Checks**

Performed Test	Liveness	Readiness
Server Status	Any status	Running
Boot Errors	None	None

Performed Test	Liveness	Readiness
Deployment Status <sup>[a]</sup>	N/A or no <b>failed</b> entries	N/A or no <b>failed</b> entries
Eclipse MicroProfile Health <sup>[b]</sup>	N/A or <b>UP</b>	N/A or <b>UP</b>
<p>[a] N/A is only a valid state when no deployments are present.</p> <p>[b] N/A is only a valid state when the <b>microprofile-health-smallrye</b> subsystem has been disabled.</p>		

## 8.9. MESSAGING

### 8.9.1. Configuring External Red Hat AMQ Brokers

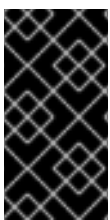
You can configure the JBoss EAP for OpenShift image with environment variables to connect to external Red Hat AMQ brokers.

#### Example OpenShift Application Definition

The following example uses a template to create a JBoss EAP application connected to an external Red Hat AMQ 7 broker.

#### Example: JDK 8

```
oc new-app eap74-amq-s2i \
-p EAP_IMAGE_NAME=jboss-eap74-openjdk8-openshift:7.4.0 \
-p EAP_RUNTIME_IMAGE_NAME=jboss-eap74-openjdk8-runtime-openshift:7.4.0 \
-p APPLICATION_NAME=eap74-mq \
-p MQ_USERNAME=MY_USERNAME \
-p MQ_PASSWORD=MY_PASSWORD
```



#### IMPORTANT

The template used in this example provides valid default values for the required parameters. If you do not use a template and provide your own parameters, be aware that the **MQ\_SERVICE\_PREFIX\_MAPPING** name must match the **APPLICATION\_NAME** name, appended with "-amq7=MQ".

## 8.10. SECURITY DOMAINS

To configure a new Security Domain, the user must define the **SECDOMAIN\_NAME** environment variable.

This results in the creation of a security domain named after the environment variable. The user may also define the following environment variables to customize the domain:

#### Table 8.6. Security Domains



Variable name	Description
SECDOMAIN_NAME	Defines an additional security domain.  Example value: <b>myDomain</b>
SECDOMAIN_PASSWORD_STACKING	If defined, the <b>password-stacking</b> module option is enabled and set to the value <b>useFirstPass</b> .  Example value: <b>true</b>
SECDOMAIN_LOGIN_MODULE	The login module to be used.  Defaults to <b>UsersRoles</b>
SECDOMAIN_USERS_PROPERTIES	The name of the properties file containing user definitions.  Defaults to <b>users.properties</b>
SECDOMAIN_ROLES_PROPERTIES	The name of the properties file containing role definitions.  Defaults to <b>roles.properties</b>

## 8.11. HTTPS ENVIRONMENT VARIABLES

Variable name	Description
HTTPS_NAME	If defined along with <b>HTTPS_PASSWORD</b> and <b>HTTPS_KEYSTORE</b> , enables HTTPS and sets the SSL name.  This should be the value specified as the alias name of your keystore if you created it with the <b>keytool -genkey</b> command.  Example value: <b>example.com</b>
HTTPS_PASSWORD	If defined along with <b>HTTPS_NAME</b> and <b>HTTPS_KEYSTORE</b> , enables HTTPS and sets the SSL key password.  Example value: <b>passw0rd</b>
HTTPS_KEYSTORE	If defined along with <b>HTTPS_PASSWORD</b> and <b>HTTPS_NAME</b> , enables HTTPS and sets the SSL certificate key file to a relative path under <b>EAP_HOME/standalone/configuration</b>  Example value: <b>ssl.key</b>

## 8.12. ADMINISTRATION ENVIRONMENT VARIABLES

Table 8.7. Administration Environment Variables

Variable name	Description
ADMIN_USERNAME	If both this and <b>ADMIN_PASSWORD</b> are defined, used for the JBoss EAP management user name.  Example value: <b>eapadmin</b>
ADMIN_PASSWORD	The password for the specified <b>ADMIN_USERNAME</b> .  Example value: <b>passw0rd</b>

## 8.13. S2I

The image includes S2I scripts and Maven.

Maven is currently only supported as a build tool for applications that are supposed to be deployed on JBoss EAP-based containers (or related/descendant images) on OpenShift.

Only WAR deployments are supported at this time.

### 8.13.1. Custom Configuration

It is possible to add custom configuration files for the image. All files put into **configuration/** directory will be copied into **EAP\_HOME/standalone/configuration/**. For example to override the default configuration used in the image, just add a custom **standalone-openshift.xml** into the **configuration/** directory. [See example](#) for such a deployment.

#### 8.13.1.1. Custom Modules

It is possible to add custom modules. All files from the **modules/** directory will be copied into **EAP\_HOME/modules/**. [See example](#) for such a deployment.

### 8.13.2. Deployment Artifacts

By default, artifacts from the source **target** directory will be deployed. To deploy from different directories set the **ARTIFACT\_DIR** environment variable in the BuildConfig definition. **ARTIFACT\_DIR** is a comma-delimited list. For example: **ARTIFACT\_DIR=app1/target,app2/target,app3/target**

### 8.13.3. Artifact Repository Mirrors

A repository in Maven holds build artifacts and dependencies of various types, for example, all of the project JARs, library JARs, plug-ins, or any other project specific artifacts. It also specifies locations from where to download artifacts while performing the S2I build. Besides using central repositories, it is a common practice for organizations to deploy a local custom mirror repository.

Benefits of using a mirror are:

- Availability of a synchronized mirror, which is geographically closer and faster.
- Ability to have 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.

Often, a repository manager can serve as local cache to a mirror. Assuming that the repository manager is already deployed and reachable externally at **https://10.0.0.1:8443/repository/internal/**, the S2I build can then use this manager by supplying the **MAVEN\_MIRROR\_URL** environment variable to the build configuration of the application as follows:

1. Identify the name of the build configuration to apply **MAVEN\_MIRROR\_URL** variable against.

```
oc get bc -o name
buildconfig/eap
```

2. Update build configuration of **eap** with a **MAVEN\_MIRROR\_URL** environment variable.

```
oc env bc/eap MAVEN_MIRROR_URL="https://10.0.0.1:8443/repository/internal/"
buildconfig "eap" updated
```

3. Verify the setting.

```
oc env bc/eap --list
# buildconfigs eap
MAVEN_MIRROR_URL=https://10.0.0.1:8443/repository/internal/
```

4. Schedule new build of the application.



#### NOTE

During application build, you will notice that Maven dependencies are pulled from the repository manager, instead of the default public repositories. Also, after the build is finished, you will see that the mirror is filled with all the dependencies that were retrieved and used during the build.

#### 8.13.3.1. Secure Artifact Repository Mirror URLs

To prevent "man-in-the-middle" attacks through the Maven repository, JBoss EAP requires the use of secure URLs for artifact repository mirror URLs.

The URL should specify a secure http ("https") and a secure port.

By default, if you specify an unsecure URL, an error will be returned. You can override this behavior using the the property **-Dinsecure.repositories=WARN**.

#### 8.13.4. Scripts

##### run

This script uses the **openshift-launch.sh** script that configures and starts JBoss EAP with the **standalone-openshift.xml** configuration.

##### assemble

This script uses Maven to build the source, create a package (WAR), and move it to the **EAP\_HOME/standalone/deployments** directory.

#### 8.13.5. Custom Scripts

You can add custom scripts to run when starting a pod, before JBoss EAP is started.

You can add any script valid to run when starting a pod, including CLI scripts.

Two options are available for including scripts when starting JBoss EAP from an image:

- Mount a configmap to be executed as `postconfigure.sh`
- Add an `install.sh` script in the nominated installation directory

### 8.13.5.1. Mounting a configmap to execute custom scripts

Mount a configmap when you want to mount a custom script at runtime to an existing image (in other words, an image that has already been built).

To mount a configmap:

1. Create a configmap with content you want to include in the `postconfigure.sh`. For example, create a directory called **extensions** in the project root directory to include the scripts **postconfigure.sh** and **extensions.cli** and run the following command:

```
$ oc create configmap jboss-cli --from-file=postconfigure.sh=extensions/postconfigure.sh --from-file=extensions.cli=extensions/extensions.cli
```

2. Mount the configmap into the pods via the deployment controller (dc).

```
$ oc set volume dc/eap-app --add --name=jboss-cli -m /opt/eap/extensions -t configmap --configmap-name=jboss-cli --default-mode='0755' --overwrite
```

#### Example `postconfigure.sh`

```
#!/usr/bin/env bash
set -x
echo "Executing postconfigure.sh"
$JBOSS_HOME/bin/jboss-cli.sh --file=$JBOSS_HOME/extensions/extensions.cli
```

#### Example `extensions.cli`

```
embed-server --std-out=echo --server-config=standalone-openshift.xml
:whoami
quit
```

### 8.13.5.2. Using `install.sh` to execute custom scripts

Use `install.sh` when you want to include the script as part of the image when it is built.

To execute custom scripts using `install.sh`:

1. In the git repository of the project that will be used during `s2i` build, create a directory called **.s2i**.
2. Inside the **s2i** directory, add a file called `environment`, with the following content:

```
$ cat .s2i/environment
CUSTOM_INSTALL_DIRECTORIES=extensions
```

3. Create a directory called **extensions**.
4. In the **extensions** directory, create the file `postconfigure.sh` with contents similar to the following (replace placeholder code with appropriate code for your environment):

```
$ cat extensions/postconfigure.sh
#!/usr/bin/env bash
echo "Executing patch.cli"
$JBOSS_HOME/bin/jboss-cli.sh --file=$JBOSS_HOME/extensions/some-cli-example.cli
```

5. In the `extensions` directory, create the file `install.sh` with contents similar to the following (replace placeholder code with appropriate code for your environment):

```
$ cat extensions/install.sh
#!/usr/bin/env bash
set -x
echo "Running $PWD/install.sh"
injected_dir=$1
# copy any needed files into the target build.
cp -rf ${injected_dir} $JBOSS_HOME/extensions
```

### 8.13.6. Environment Variables

You can influence the way the build is executed by supplying environment variables to the **s2i build** command. The environment variables that can be supplied are:

**Table 8.8. s2i Environment Variables**

Variable name	Description
ARTIFACT_DIR	The <b>.war</b> , <b>.ear</b> , and <b>.jar</b> files from this directory will be copied into the <b>deployments/</b> directory.  Example value: <b>target</b>
ENABLE_GENERATE_DEFAULT_DATASOURCE	Optional. When included with the value <b>true</b> , the server is provisioned with the default datasource. Otherwise, the default datasource is not included.
GALLEON_PROVISION_DEFAULT_FAT_SERVER	Optional. When included with the value <b>true</b> , and no galleon layers have been set, a default JBoss EAP server is provisioned.
GALLEON_PROVISION_LAYERS	Optional. Instructs the S2I process to provision the specified layers. The value is a comma-separated list of layers to provision, including one base layer and any number of decorator layers.  Example value: <b>jaxrs, sso</b>

Variable name	Description
HTTP_PROXY_HOST	Host name or IP address of a HTTP proxy for Maven to use.  Example value: <b>192.168.1.1</b>
HTTP_PROXY_PORT	TCP Port of a HTTP proxy for Maven to use.  Example value: <b>8080</b>
HTTP_PROXY_USERNAME	If supplied with <b>HTTP_PROXY_PASSWORD</b> , use credentials for HTTP proxy.  Example value: <b>myusername</b>
HTTP_PROXY_PASSWORD	If supplied with <b>HTTP_PROXY_USERNAME</b> , use credentials for HTTP proxy.  Example value: <b>mypassword</b>
HTTP_PROXY_NONPROXYHOSTS	If supplied, a configured HTTP proxy will ignore these hosts.  Example value: <b>some.example.org *.example.net</b>
MAVEN_ARGS	Overrides the arguments supplied to Maven during build.  Example value: <b>-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package</b>
MAVEN_ARGS_APPEND	Appends user arguments supplied to Maven during build.  Example value: <b>-Dfoo=bar</b>
MAVEN_MIRROR_URL	URL of a Maven Mirror/repository manager to configure.  Example value: <b>https://10.0.0.1:8443/repository/internal/</b>  Note that the specified URL should be secure. For details see <a href="#">Section 8.13.3.1, "Secure Artifact Repository Mirror URLs"</a> .
MAVEN_CLEAR_REPO	Optionally clear the local Maven repository after the build.  If the server present in the image is strongly coupled to the local cache, the cache is not deleted and a warning is printed.  Example value: <b>true</b>
APP_DATADIR	If defined, directory in the source from where data files are copied.  Example value: <b>mydata</b>

Variable name	Description
DATA_DIR	Directory in the image where data from <b>\$APP_DATADIR</b> will be copied.  Example value: <b>EAP_HOME/data</b>

**NOTE**

For more information, see [Build and Run a Java Application on the JBoss EAP for OpenShift Image](#), which uses Maven and the S2I scripts included in the JBoss EAP for OpenShift image.

## 8.14. SINGLE SIGN-ON IMAGE

This image includes the Red Hat Single Sign-On-enabled applications.

For more information on deploying the Red Hat Single Sign-On for OpenShift image with the JBoss EAP for OpenShift image, see [Deploy the Red Hat Single Sign-On-enabled JBoss EAP Image](#) on the *Red Hat Single Sign-On for OpenShift* guide.

**Table 8.9. Single Sign-On environment variables**

Variable name	Description
<b>SSO_URL</b>	URL of the Single Sign-On server.
<b>SSO_REALM</b>	Single Sign-On realm for the deployed applications.
<b>SSO_PUBLIC_KEY</b>	Public key of the Single Sign-On realm. This field is optional but if omitted can leave the applications vulnerable to man-in-middle attacks.
<b>SSO_USERNAME</b>	Single Sign-On user required to access the Single Sign-On REST API.  Example value: <b>mySsoUser</b>
<b>SSO_PASSWORD</b>	Password for the Single Sign-On user defined by the <b>SSO_USERNAME</b> variable.  Example value: <b>6fedmL3P</b>
<b>SSO_SAML_KEYSTORE</b>	Keystore location for SAML. Defaults to <b>/etc/sso-saml-secret-volume/keystore.jks</b> .
<b>SSO_SAML_KEYSTORE_PASSWORD</b>	Keystore password for SAML. Defaults to <b>mykeystorepass</b> .

Variable name	Description
<b>SSO_SAML_CERTIFICATE_NAME</b>	Alias for keys and certificates to use for SAML. Defaults to <b>jboss</b> .
<b>SSO_BEARER_ONLY</b>	Single Sign-On client access type. (Optional)  Example value: <b>true</b>
<b>SSO_CLIENT</b>	Path for Single Sign-On redirects back to the application. Defaults to match <b>module-name</b> .
<b>SSO_ENABLE_CORS</b>	If <b>true</b> , enable Cross-Origin Resource Sharing (CORS) for Single Sign-On applications. (Optional)
<b>SSO_SECRET</b>	The Single Sign-On client secret for confidential access.  Example value: <b>KZ1Qylq4</b>
<b>SSO_DISABLE_SSL_CERTIFICATE_VALIDATION</b>	If <b>true</b> the SSL/TLS communication between JBoss EAP and the Red Hat Single Sign-On server is unsecure, for example, the certificate validation is disabled with <b>curl</b> . Not set by default.  Example value: <b>true</b>

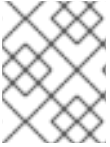
Table 8.10. Secrets

Variable name	Description
<b>SSO_SAML_KEYSTORE_SECRET</b>	Secret to use for access to SAML keystore. The default value is <b>sso-app-secret</b> .
<b>HTTPS_SECRET</b>	The name of the secret containing the keystore file.  Example value: <b>eap-ssl-secret</b>
<b>SSO_TRUSTSTORE_SECRET</b>	The name of the secret containing the truststore file. Used for <b>sso-truststore-volume</b> volume.  Example value: <b>sso-app-secret</b>

## 8.15. UNSUPPORTED TRANSACTION RECOVERY SCENARIOS

- JTS transactions are not supported in OpenShift.
- XTS transactions are not supported in OpenShift.
- The [XATerminator](#) interface that some third parties use for transaction completion and crash recovery flows is not supported in OpenShift.
- Transactions propagated over [JBoss Remoting](#) is unsupported with OpenShift 3.



**NOTE**

Transactions propagated over [JBoss Remoting](#) is supported with OpenShift 4 and the EAP operator.

## 8.16. INCLUDED JBOSS MODULES

The table below lists included JBoss Modules in the JBoss EAP for OpenShift image.

**Table 8.11. Included JBoss Modules**

JBoss Module
org.jboss.as.clustering.common
org.jboss.as.clustering.jgroups
org.jboss.as.ee
org.jgroups
org.openshift.ping
net.oauth.core

## 8.17. EAP OPERATOR: API INFORMATION

The EAP operator introduces the following APIs:

### 8.17.1. WildFlyServer

**WildFlyServer** defines a custom JBoss EAP resource.

**Table 8.12. WildFlyServer**

Field	Description	Scheme	Required
<b>metadata</b>	Standard object's metadata	<a href="#">ObjectMeta v1 meta</a>	false
<b>spec</b>	<a href="#">Specification</a> of the desired behaviour of the JBoss EAP deployment.	<a href="#">WildFlyServerSpec</a>	true
<b>status</b>	Most recent observed <a href="#">status</a> of the JBoss EAP deployment. Read-only.	<a href="#">WildFlyServerStatus</a>	false

### 8.17.2. WildFlyServerList

**WildFlyServerList** defines a list of JBoss EAP deployments.

Table 8.13. Table

Field	Description	Scheme	Required
<b>metadata</b>	Standard list's metadata	<a href="#">metav1.ListMeta</a>	false
<b>items</b>	List of <b>WildFlyServer</b>	<a href="#">WildFlyServer</a>	true

### 8.17.3. WildFlyServerSpec

**WildFlyServerSpec** is a specification of the desired behavior of the JBoss EAP resource.

It uses a **StatefulSet** with a pod spec that mounts the volume specified by storage on `/opt/jboss/wildfly/standalone/data`.

Table 8.14. WildFlyServerSpec

Field	Description	Scheme	Required
<b>applicationImage</b>	Name of the application image to be deployed	string	false
<b>replicas</b>	the desired number of replicas for the application	int32]	true
<b>standaloneConfigMap</b>	Spec to specify how a standalone configuration can be read from a <b>ConfigMap</b> .	<a href="#">StandaloneConfigMapSpec</a>	false
<b>resources</b>	<b>Resources</b> spec to specify the request or limits of the Stateful Set. If omitted, the namespace defaults are used.	<a href="#">Resources</a>	false
<b>SecurityContext</b>	<b>SecurityContext</b> spec to define privilege and access control settings for the pod containers created by the Stateful Set. If omitted, default privileges are used. For additional information see <a href="#">securityContext</a> .	<b>*corev1.SecurityContext</b>	false

Field	Description	Scheme	Required
<b>storage</b>	Storage spec to specify how storage should be used. If omitted, an <b>EmptyDir</b> is used (that does not persist data across pod restart)	<a href="#">StorageSpec</a>	false
<b>serviceAccountName</b>	Name of the ServiceAccount to use to run the JBoss EAP pods	string	false
<b>envFrom</b>	List of environment variables present in the containers from <b>configMap</b> or secret	<a href="#">corev1.EnvFromSource</a>	false
<b>env</b>	List of environment variable present in the containers	<a href="#">corev1.EnvVar</a>	false
<b>secrets</b>	List of secret names to mount as volumes in the containers. Each secret is mounted as a read-only volume at <b>/etc/secrets/&lt;secret name&gt;</b>	string	false
<b>configMaps</b>	List of <b>ConfigMap</b> names to mount as volumes in the containers. Each <b>ConfigMap</b> is mounted as a read-only volume under <b>/etc/configmaps/&lt;config map name&gt;</b>	string	false
<b>disableHTTPRoute</b>	Disable the creation a route to the HTTP port of the application service (false if omitted)	boolean	false

Field	Description	Scheme	Required
<b>sessionAffinity</b>	If connections from the same client IP are passed to the same JBoss EAP instance/pod each time (false if omitted)	boolean	false

#### 8.17.4. Resources

**Resources** defines the configured resources for a **WildflyServer** resource. If the **Resources** field is not defined or **Request** or **Limits** is empty, this resource is removed from the **StatefulSet**. The description of this resource is a standard **Container** resource and uses the scheme for [corev1.ResourceRequirements](#).

#### 8.17.5. StorageSpec

**StorageSpec** defines the configured storage for a **WildFlyServer** resource. If neither an **EmptyDir** nor a **volumeClaimTemplate** is defined, a default **EmptyDir** is used.

The EAP Operator configures the **StatefulSet** using information from this **StorageSpec** to mount a volume dedicated to the standalone/data directory used by JBoss EAP to persist its own data. For example, transaction log). If an **EmptyDir** is used, the data does not survive a pod restart. If the application deployed on JBoss EAP relies on transaction, specify a **volumeClaimTemplate**, so that the same persistent volume can be reused upon pod restarts.

Table 8.15. Table

Field	Description	Scheme	Required
<b>emptyDir</b>	<b>EmptyDirVolumeSource</b> to be used by the JBoss EAP <b>StatefulSet</b>	<a href="#">corev1.EmptyDirVolumeSource</a>	false
<b>volumeClaimTemplate</b>	A <b>PersistentVolumeClaim</b> spec to configure <b>Resources</b> requirements to store JBoss EAP standalone data directory. The name of the template is derived from the <b>WildFlyServer</b> name. The corresponding volume is mounted in <b>ReadWriteOnce</b> access mode.	<a href="#">corev1.PersistentVolumeClaim</a>	false

#### 8.17.6. StandaloneConfigMapSpec

**StandaloneConfigMapSpec** defines how JBoss EAP standalone configuration can be read from a **ConfigMap**. If omitted, JBoss EAP uses its **standalone.xml** configuration from its image.

Table 8.16. StandaloneConfigMapSpec

Field	Description	Scheme	Required
<b>name</b>	Name of the <b>ConfigMap</b> containing the standalone configuration XML file.	string	true
key	Key of the <b>ConfigMap</b> whose value is the standalone configuration XML file. If omitted, the spec finds the <b>standalone.xml</b> key.	string	false

### 8.17.7. WildFlyServerStatus

**WildFlyServerStatus** is the most recent observed status of the JBoss EAP deployment. Read-only.

Table 8.17. WildFlyServerStatus

Field	Description	Scheme	Required
<b>replicas</b>	The actual number of replicas for the application	int32	true
<b>selector</b>	selector for pods, used by HorizontalPodAutoscaler	string	true
<b>hosts</b>	Hosts that route to the application HTTP service	string	true
<b>pods</b>	Status of the pods	<a href="#">PodStatus</a>	true
<b>scalingdownPods</b>	Number of pods that are under scale down cleaning process	int32	true

### 8.17.8. PodStatus

**PodStatus** is the most recent observed status of a pod running the JBoss EAP application.

Table 8.18. PodStatus

Field	Description	Scheme	Required
<b>name</b>	Name of the pod	string	true
<b>podIP</b>	IP address allocated to the pod	string	true
<b>state</b>	State of the pod in the scale down process. The state is ACTIVE by default, which means it serves requests.	string	false

Revised on 2024-02-08 08:02:02 UTC