



Red Hat JBoss Enterprise Application Platform 8.0

Using JBoss EAP on OpenShift Container Platform

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

Red Hat JBoss Enterprise Application Platform 8.0 Using JBoss EAP on OpenShift Container Platform

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

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Guide to using Red Hat JBoss Enterprise Application Platform for OpenShift

Table of Contents

PROVIDING FEEDBACK ON JBOSS EAP DOCUMENTATION	6
MAKING OPEN SOURCE MORE INCLUSIVE	7
CHAPTER 1. WHAT IS RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM	8
1.1. HOW DOES JBOSS EAP WORK ON OPENSIFT?	8
1.2. COMPARISON: JBOSS EAP AND JBOSS EAP FOR OPENSIFT	8
1.3. VERSION COMPATIBILITY AND SUPPORT	9
1.3.1. OpenShift 4.x support	10
1.3.2. IBM Z Support	10
1.3.2.1. Upgrades from JBoss EAP 7.4 to JBoss EAP 8.0 on OpenShift	10
1.3.3. Deployment options	10
CHAPTER 2. PACKAGE NAMESPACE CHANGE FOR JBOSS EAP 8.0	12
2.1. JAVAX TO JAKARTA NAMESPACE CHANGE	12
CHAPTER 3. BUILDING AND RUNNING JBOSS EAP APPLICATIONS ON OPENSIFT CONTAINER PLATFORM	13
3.1. PREREQUISITES	13
3.2. PREPARING OPENSIFT TO DEPLOY AN APPLICATION	13
3.3. BUILDING APPLICATION IMAGES USING SOURCE-TO-IMAGE IN OPENSIFT	14
3.4. DEPLOYING A THIRD-PARTY APPLICATION ON OPENSIFT	15
3.4.1. Provisioning JBoss EAP servers with the default configuration	15
3.5. USING OPENID CONNECT TO SECURE JBOSS EAP APPLICATIONS ON OPENSIFT	17
3.5.1. OpenID Connect configuration in JBoss EAP	18
3.5.2. Creating an application secured with OpenID Connect	18
3.5.3. Deploying the application on OpenShift	24
3.5.4. Environment variable based configuration	26
3.6. SECURING APPLICATIONS BY USING SAML	28
3.6.1. Keycloak SAML adapter feature pack for securing applications by using SAML	28
3.6.2. Configuring Red Hat build of Keycloak as SAML provider for OpenShift	28
3.6.3. Creating an application secured with SAML	29
3.6.4. Building and deploying a SAML-secured application on OpenShift	35
3.6.5. Creating a SSO realm, users, and roles	37
3.6.6. Environment variables for configuring the SAML subsystem	38
3.6.7. Route discovery in JBoss EAP server	39
3.6.8. Additional resources	40
3.7. ADDITIONAL RESOURCES	40
CHAPTER 4. USING HELM CHARTS TO BUILD AND DEPLOY JBOSS EAP APPLICATIONS ON OPENSIFT ..	41
4.1. HELM CHART USE CASE	41
4.2. HELM CHART CUSTOMIZATION FOR JBOSS EAP ON OPENSIFT	41
4.3. PROVISIONING JBOSS EAP WITH S2I	41
4.4. BUILDING AND DEPLOYING JBOSS EAP APPLICATIONS USING HELM CHARTS	42
4.5. BUILDING YOUR APPLICATION IMAGE USING THE OPENSIFT DEVELOPMENT CONSOLE	42
4.6. DEPLOYING YOUR APPLICATION IMAGE	43
4.6.1. OpenShift volumes for persistent data storage in Helm chart	43
4.6.2. Mounting a volume with a Helm chart	43
CHAPTER 5. ENVIRONMENT VARIABLES AND MODEL EXPRESSION RESOLUTION	45
5.1. PREREQUISITES	45
5.2. ENVIRONMENT VARIABLES FOR RESOLVING MANAGEMENT MODEL EXPRESSIONS	45

System property to environment variable mapping	45
5.3. CONFIGURING ENVIRONMENT VARIABLES ON THE OPENSIFT CONTAINER PLATFORM	46
5.4. OVERRIDING MANAGEMENT ATTRIBUTES WITH ENVIRONMENT VARIABLES	47
CHAPTER 6. PROVISIONING A JBOSS EAP SERVER USING THE MAVEN PLUG-IN	49
6.1. JBOSS EAP MAVEN PLUG-IN	49
6.2. CREATING A JAKARTA EE 10 APPLICATION WITH THE MAVEN	49
6.3. USING THE MAVEN PLUG-IN TO PROVISION A JBOSS EAP SERVER	51
6.4. THE GALLEON PROVISIONING FILE	53
6.5. THE MAVEN PLUG-IN CONFIGURATION ATTRIBUTES	54
6.6. HOW TO ENABLE SUPPORT FOR EAP-DATASOURCES-GALLEON-PACK FOR JBOSS EAP 8.0	58
6.7. SUPPORTED DRIVERS AND DATA SOURCES	59
6.8. USING THE JBOSS EAP MAVEN PLUGIN TO PROVISION A SERVER WITH JDBC DRIVERS AND DATA SOURCES	59
CHAPTER 7. CONFIGURING YOUR JBOSS EAP SERVER AND APPLICATION	61
7.1. JVM DEFAULT MEMORY SETTINGS	61
7.2. JVM GARBAGE COLLECTION SETTINGS	62
7.3. JVM ENVIRONMENT VARIABLES	62
7.4. DEFAULT DATASOURCE	65
CHAPTER 8. CAPABILITY TRIMMING IN JBOSS EAP FOR OPENSIFT	67
8.1. AVAILABLE JBOSS EAP LAYERS	67
8.1.1. Base layers	67
datasources-web-server	67
jaxrs-server	68
cloud-server	69
cloud-default-config	69
ee-core-profile-server	69
8.1.2. Decorator layers	69
observability	69
web-clustering	70
8.2. PROVISIONING USER-DEVELOPED LAYERS IN JBOSS EAP	70
8.2.1. Building and using custom Galleon layers for JBoss EAP	70
8.2.1.1. Preparing the Maven project	70
8.2.1.2. Adding the feature-pack content	72
8.2.1.3. Using the custom Galleon feature-pack during S2I build	74
8.2.1.4. Importing the JBoss EAP 8 image stream	74
8.2.1.4.1. Creating an S2I build using the JBoss EAP maven plugin	75
8.2.1.4.2. Creating an S2I build using the legacy S2I provisioning capabilities	76
8.2.1.4.3. Starting the build	76
8.2.1.4.4. Creating a new deployment	77
8.2.2. Configure Galleon by using advanced environment variables	77
8.2.3. Custom Galleon feature pack environment variables	78
CHAPTER 9. DEPLOYING YOUR JBOSS EAP APPLICATION ON THE OPENSIFT CONTAINER PLATFORM	80
9.1. JBOSS EAP OPERATOR FOR AUTOMATING APPLICATION DEPLOYMENT ON OPENSIFT	80
9.1.1. Installing EAP operator using the web console	80
9.1.2. Installing EAP operator using the CLI	82
9.1.3. Deploying a Java application on OpenShift using the EAP operator	83
9.1.3.1. Creating a secret	86
9.1.3.2. Creating a configMap	86
9.1.3.3. Creating a configMap from a standalone.xml File	86

9.1.3.4. Configuring persistent storage for applications	87
9.1.4. Viewing metrics of an application using the EAP operator	87
9.1.5. Uninstalling EAP operator using web console	88
9.1.6. Uninstalling JBoss EAP operator using the CLI	88
9.1.7. JBoss EAP operator for safe transaction recovery	89
9.1.7.1. StatefulSets for stable network host names	90
9.1.7.2. Monitoring the scaledown process	90
9.1.7.2.1. Pod status during Scaledown	91
9.1.7.3. Scaling down during transactions with heuristic outcomes	91
9.1.7.4. Configuring the transactions subsystem to use the JDBC storage for transaction log	92
9.1.8. Automatically scaling pods with the horizontal pod autoscaler HPA	93
9.1.9. Jarkarta enterprise beans remoting on OpenShift	94
9.1.9.1. Jakarta Enterprise Beans remoting on openShift	94
9.1.9.1.1. Configuring Jakarta Enterprise Beans on OpenShift	95
CHAPTER 10. TROUBLESHOOTING	97
10.1. TROUBLESHOOTING POD RESTARTS	97
10.2. TROUBLESHOOTING USING THE JBOSS EAP MANAGEMENT CLI	97
10.3. TROUBLESHOOTING ERRORS WHEN UPDATING HELM CHART FROM VERSION 1.0.0 TO 1.1.0 ON JBOSS EAP 8	98
CHAPTER 11. REFERENCE INFORMATION FOR OPENSIFT CONTAINER PLATFORM	99
11.1. INFORMATION ENVIRONMENT VARIABLES	99
11.2. CONFIGURATION ENVIRONMENT VARIABLES	99
11.3. EXPOSED PORTS	102
11.4. DATASOURCES	102
11.4.1. JNDI mappings for datasources	103
11.4.1.1. Datasource Configuration Environment Variables	103
11.4.1.2. Examples	105
11.4.1.2.1. Single Mapping	105
11.4.1.2.2. Multiple Mappings	106
11.5. CLUSTERING	106
11.5.1. Configuring a JGroups Discovery Mechanism	106
11.5.1.1. Configuring KUBE_PING	106
11.5.1.2. Configuring DNS_PING	107
11.5.2. Configuring JGroups to Encrypt Cluster Traffic	108
11.5.2.1. Configuring SYM_ENCRYPT	109
11.5.2.2. Configuring ASYM_ENCRYPT	109
11.5.3. Considerations for scaling up pods	110
11.6. NATIVE HEALTH CHECKS	110
11.7. MESSAGING	111
11.7.1. Configuring External Red Hat AMQ Brokers	111
11.8. SECURITY DOMAINS	111
11.9. HTTPS ENVIRONMENT VARIABLES	111
11.10. ADMINISTRATION ENVIRONMENT VARIABLES	112
11.11. S2I	112
11.11.1. Custom Configuration	112
11.11.1.1. Custom Modules	113
11.11.2. Deployment Artifacts	113
11.11.3. Artifact repository mirrors	113
11.11.3.1. Secure artifact repository mirror URLs	114
11.11.4. Scripts	114
11.11.5. Custom Scripts	114

11.11.5.1. Mounting a configmap to execute custom scripts	114
11.11.5.2. Using install.sh to execute custom scripts	115
11.11.6. Environment variables	115
11.12. UNSUPPORTED TRANSACTION RECOVERY SCENARIOS	118
11.13. INCLUDED JBOSS MODULES	118
11.14. EAP OPERATOR: API INFORMATION	118
11.14.1. WildFlyServer	118
11.14.2. WildFlyServerList	119
11.14.3. WildFlyServerSpec	119
11.14.4. Resources	121
11.14.5. StorageSpec	121
11.14.6. StandaloneConfigMapSpec	122
11.14.7. WildFlyServerStatus	122
11.14.8. PodStatus	123

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. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. WHAT IS RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

Red Hat JBoss Enterprise Application Platform 8.0 (JBoss EAP) is a middleware platform built on open standards and compliant with the Jakarta EE 10 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.

By using the web-based management console and management command line interface (CLI), you can script and automate tasks and avoid having to edit XML configuration files. In addition, JBoss EAP includes APIs and development frameworks that you can use to develop, deploy, and run secure and scalable Jakarta EE applications. JBoss EAP 8.0 is a Jakarta EE 10 compatible implementation for Web Profile, Core Profile, and Full Platform specifications.

1.1. HOW DOES JBOSS EAP WORK ON OPENSHIFT?

Red Hat offers container images to build and run application images with JBoss EAP on OpenShift.



NOTE

Red Hat no longer offers images that contain JBoss EAP.

1.2. 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. The management CLI is accessible from within a pod for troubleshooting purposes.
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 ROOT.war .

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	Supported	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 EAP Operator for Safe Transaction Recovery .

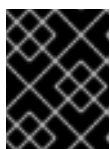
1.3. VERSION COMPATIBILITY AND SUPPORT

JBoss EAP for OpenShift provides images for OpenJDK 17.

Two variant of the image are available: an S2I builder image and a runtime image. The S2I Builder image contains all the required tools that will enable you provision a complete JBoss EAP Server 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 the images in JBoss EAP 8.0 for OpenShift.

- S2I builder image does not contain an installed JBoss EAP server and installs the JBoss EAP 8.0 server during S2I build.
- Configure the eap-maven-plugin in the application **pom** file during S2I build.
- Use existing JBoss EAP 7.4 application without any changes by setting **GALLEON_PROVISION_FEATURE_PACKS**, **GALLEON_PROVISION_LAYERS**, and **GALLEON_PROVISION_CHANNELS** environment variables during S2I build.
- The JBoss EAP provisioned server during S2I build contains a **standalone.xml** server configuration file customized for OpenShift.



IMPORTANT

The sever contains a **standalone.xml** configuration file, not the **standalone-openshift.xml** configuration file that was used with JBoss EAP 7.4.

- Inside the image, **JBOSS_HOME** value is **/opt/server**. The value of **JBOSS_HOME** was **/opt/eap** for JBoss EAP 7.4.
- **Jolokia agent** is no longer present in the image.
- **Prometheus agent** is not installed.
- **Python probes** are no more present.
- **SSO** adapters are no longer present in the image.

- **activemq.rar** is no more present.



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.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.

1.3.1. 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.

1.3.2. 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.

1.3.2.1. Upgrades from JBoss EAP 7.4 to JBoss EAP 8.0 on OpenShift

The file **standalone.xml** installed with JBoss EAP 7.4 on OpenShift is not compatible with JBoss EAP 8.0 and later. You must modify and rename the file to **standalone.xml** before starting a JBoss EAP 8.0 or later container for OpenShift.

Additional resources

- [Updates to **standalone.xml** when upgrading JBoss EAP 7.1 to JBoss EAP 8.0 on OpenShift](#) .

1.3.3. Deployment options

You can deploy the JBoss EAP Java applications on OpenShift using 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.

Additional resources

- For more information about the EAP operator, see [EAP Operator for Automating Application Deployment on OpenShift](#).

CHAPTER 2. PACKAGE NAMESPACE CHANGE FOR JBOSS EAP 8.0

This section provides additional information for the package namespace changes in JBoss EAP 8.0. JBoss EAP 8.0 provides full support for Jakarta EE 10 and many other implementations of the Jakarta EE 10 APIs. An important change supported by Jakarta EE 10 for JBoss EAP 8.0 is the package namespace change.

2.1. JAVAX TO JAKARTA NAMESPACE CHANGE

A key difference between Jakarta EE 8 and EE 10 is the renaming of the EE API Java packages from **javax.*** to **jakarta.***. This follows the move of Java EE to the Eclipse Foundation and the establishment of Jakarta EE.

Adapting to this namespace change is the biggest task of migrating an application from JBoss EAP 7 to JBoss EAP 8. To migrate applications to Jakarta EE 10, you must complete the following steps:

- Update any import statements or other source code uses of EE API classes from the **javax** package to the **jakarta** package.
- Update the names of any EE-specified system properties or other configuration properties that begin with **javax** to begin with **jakarta**.
- For any application-provided implementations of EE interfaces or abstract classes that are bootstrapped using the **java.util.ServiceLoader** mechanism, change the name of the resource that identifies the implementation class from **META-INF/services/javax.[rest_of_name]** to **META-INF/services/jakarta.[rest_of_name]**.



NOTE

The Red Hat Migration Toolkit can assist in updating the namespaces in the application source code. For more information, see [How to use Red Hat Migration Toolkit for Auto-Migration of an Application to the Jakarta EE 10 Namespace](#). In cases where source code migration is not an option, the Open Source [Eclipse Transformer](#) project provides bytecode transformation tooling to transform existing Java archives from the **javax** namespace to the **jakarta** namespace.



NOTE

This change does not affect **javax** packages that are part of Java SE.

Additional resources

- For more information, see [The javax to jakarta Package Namespace Change](#).

CHAPTER 3. BUILDING AND RUNNING JBOSS EAP APPLICATIONS ON OPENSIFT CONTAINER PLATFORM

You can follow the source-to-image (S2I) process to build and run a Java application on the JBoss EAP for OpenShift image.

3.1. PREREQUISITES

- You have an OpenShift instance installed and operational.

3.2. PREPARING OPENSIFT TO DEPLOY AN APPLICATION

As a JBoss EAP application developer, you can deploy your applications on OpenShift. In the following example, note that the **kitchensink** quickstart 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 JBoss EAP 8.0 **kitchensink** quickstart for more information. Deploy your application by following the procedures below.

Procedure

- Log in to your OpenShift instance using the **oc login** command.
- Create a project in OpenShift.
Create a project using the following command. With a project, you can organize and manage content separately from other groups.

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

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

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

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



NOTE

You must create a keystore and a secret if you use any HTTPS-enabled features in your OpenShift project.

- Use the Java **keytool** command to generate a keystore:



WARNING

The following commands generate a self-signed certificate, but for production environments, use your own SSL certificate from a verified certificate authority (CA) for SSL-encrypted connections (HTTPS).

```
$ 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. Use the following command to create a secret from your new keystore:

```
$ 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 eap-app-secret --from-file=keystore.jks
```

Additional resources

- [ImageStreams and Pods fail to pull images when Dev Portal generated secret is added in the namespace](#)

3.3. BUILDING APPLICATION IMAGES USING SOURCE-TO-IMAGE IN OPENSIFT

Follow the source-to-image (S2I) workflow to build reproducible container images for a JBoss EAP application. These generated container images include the application deployment and ready-to-run JBoss EAP servers.

The S2I workflow takes source code from a Git repository and injects it into a container that's based on the language and framework you want to use. After the S2I workflow is completed, the **src** code is compiled, the application is packaged and is deployed to the JBoss EAP server.

For more information, see [Legacy server provisioning for JBoss EAP S2I](#).



NOTE

In JBoss EAP, you can use S2I images only if you develop your application using Jakarta EE 10.

Prerequisites

- You have an active Red Hat customer account.
- You have a Registry Service Account. Follow the instructions on the Red Hat Customer Portal to [create an authentication token using a registry service account](#).
- You have downloaded the OpenShift secret YAML file, which you can use to pull images from Red Hat Ecosystem Catalog. For more information, see [OpenShift Secret](#).
- You used the **oc login** command to log in to OpenShift.
- You have installed Helm. For more information, see [Installing Helm](#).

- You have installed the repository for the JBoss EAP Helm charts by entering this command in the management CLI:

```
$ helm repo add jboss-eap https://jbossas.github.io/eap-charts/
```

Procedure

1. Create a file named **helm.yaml** using the following YAML content:

```
build:
  uri: https://github.com/jboss-developer/jboss-eap-quickstarts.git
  ref: EAP_8.0.0.GA
  contextDir: helloworld
deploy:
  replicas: 1
```

2. Use the following command to deploy your JBoss EAP application on OpenShift.

```
$ helm install helloworld -f helm.yaml jboss-eap/eap8
```

Verification

- Access the application using **curl**.

```
$ curl https://$(oc get route helloworld --template='{{ .spec.host }}')/HelloWorld
```

You get the output **Hello World!** confirming that the application is deployed.

3.4. DEPLOYING A THIRD-PARTY APPLICATION ON OPENSIFT

You can create application images for OpenShift deployments by using compiled WAR files or EAR archives. Use a Dockerfile to deploy these archives onto JBoss EAP server, along with an updated and comprehensive runtime stack that includes the operating system, Java, and JBoss EAP components.



NOTE

Red Hat do not provide pre-built JBoss EAP server images.

3.4.1. Provisioning JBoss EAP servers with the default configuration

You can install and configure a JBoss EAP server with its default configuration on OpenShift by using the builder image. For seamless deployment, follow the procedure to provision the server, transfer the application files, and make any necessary customization.

Prerequisites

- You have access to the supported Red Hat JBoss Enterprise Application Platform container images. For example:
 - **registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8**
 - **registry.redhat.io/jboss-eap-8/eap8-openjdk17-runtime-openshift-rhel8**

- You have podman installed on your system. Use the latest podman version available on supported RHEL. For more information, see [Red Hat JBoss Enterprise Application Platform 8.0 Supported Configurations](#).

Procedure

- Copy the following Dockerfile contents as provided:

```
# Use EAP 8 Builder image to create a JBoss EAP 8 server
# with its default configuration

FROM registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest AS
builder

# Set up environment variables for provisioning. 1
ENV GALLEON_PROVISION_FEATURE_PACKS org.jboss.eap:wildfly-ee-galleon-
pack,org.jboss.eap.cloud:eap-cloud-galleon-pack
ENV GALLEON_PROVISION_LAYERS cloud-default-config
# Specify the JBoss EAP version 2
ENV GALLEON_PROVISION_CHANNELS org.jboss.eap.channels:eap-8.0

# Run the assemble script to provision the server.
RUN /usr/local/s2i/assemble

# Copy the JBoss EAP 8 server from the builder image to the runtime image.
FROM registry.redhat.io/jboss-eap-8/eap8-openjdk17-runtime-openshift-rhel8:latest AS
runtime

# Set appropriate ownership and permissions.
COPY --from=builder --chown=jboss:root $JBOSS_HOME $JBOSS_HOME

# Steps to add:
# (1) COPY the WAR/EAR to $JBOSS_HOME/standalone/deployments
# with the jboss:root user. For example:
# COPY --chown=jboss:root my-app.war $JBOSS_HOME/standalone/deployments 3
# (2) (optional) server modification. You can modify EAP server configuration:
#
# * invoke management operations. For example
#
# RUN $JBOSS_HOME/bin/jboss-cli.sh --commands="embed-server,/system-
property=Foo:add(value=Bar)"
#
# First operation must always be embed-server.
#
# * copy a modified standalone.xml in $JBOSS_HOME/standalone/configuration/
# for example
#
# COPY --chown=jboss:root standalone.xml $JBOSS_HOME/standalone/configuration

# Ensure appropriate permissions for the copied files.
RUN chmod -R ug+rwX $JBOSS_HOME
```

- 1 You can specify the **MAVEN_MIRROR_URL** environment variable, which is used by the JBoss EAP Maven plugin internally within the image. For more information, see [Artifact](#)
- 2 You do not need to update this Dockerfile for any of the minor releases. Specify the JBoss EAP version in the **GALLEON_PROVISION_CHANNELS** environment variable if you want to use a specific version. For more information, see [Environment variables](#).
- 3 Modify the copied Dockerfile to include your WAR file in the container. For example:

```
COPY --chown=jboss:root <my-app.war> $JBOSS_HOME/standalone/deployments
```

Replace <myapp.war> with the path to the Web archive you want to add to the image.

2. Build the application image using podman:

```
$ podman build -t my-app .
```

After the command is executed, the **my-app** container image is ready to be deployed on OpenShift.

3. Upload your container image to one of the following options:
 - Your internal registry that is accessible from OpenShift.
 - The OpenShift registry by pushing the image directly from the machine where it was built. For more information, see [How to push a container image into the image registry in RHOCP 4](#).
4. When deploying your image from the registry, use deployment strategies such as Helm charts, Operator, or Deployment. Select your preferred method and use either the full image URL or ImageStreams based on your requirements. For more information, see [Using Helm charts to build and deploy JBoss EAP applications on OpenShift](#).

3.5. USING OPENID CONNECT TO SECURE JBOSS EAP APPLICATIONS ON OPENSIFT

Use the JBoss EAP native OpenID Connect (OIDC) client to delegate authentication using an external OpenID provider. OIDC is an identity layer that enables clients, such as JBoss EAP, to verify a user's identity based on the authentication performed by an OpenID provider.

The **elytron-oidc-client** subsystem and **elytron-oidc-client** Galleon layer provides a native OIDC client in JBoss EAP to connect with OpenID providers. JBoss EAP automatically creates a virtual security domain for your application, based on your OpenID provider configurations.

You can configure the **elytron-oidc-client** subsystem in three different ways:

- Adding an **oidc.json** into your deployment.
- Running a CLI script to configure the **elytron-oidc-client** subsystem.
- Defining environment variables to configure an **elytron-oidc-client** subsystem on start of JBoss EAP server on OpenShift.



NOTE

This procedure explains how you can configure an **elytron-oidc-client** subsystem using the environment variables to secure application with OIDC.

3.5.1. OpenID Connect configuration in JBoss EAP

When you secure your applications using an OpenID provider, you do not need to configure any security domain resources locally. The **elytron-oidc-client** subsystem provides a native OpenID Connect (OIDC) client in JBoss EAP to connect with OpenID providers. JBoss EAP automatically creates a virtual security domain for your application, based on your OpenID provider configurations.



IMPORTANT

Use the OIDC client with Red Hat build of Keycloak. You can use other OpenID providers if they can be configured to use access tokens that are JSON Web Tokens (JWTs) and can be configured to use the RS256, RS384, RS512, ES256, ES384, or ES512 signature algorithm.

To enable the use of OIDC, you can configure either the **elytron-oidc-client** subsystem or an application itself. JBoss EAP activates the OIDC authentication as follows:

- When you deploy an application to JBoss EAP, the **elytron-oidc-client** subsystem scans the deployment to detect if the OIDC authentication mechanism is required.
- If the subsystem detects OIDC configuration for the deployment in either the **elytron-oidc-client** subsystem or the application deployment descriptor, JBoss EAP enables the OIDC authentication mechanism for the application.
- If the subsystem detects OIDC configuration in both places, the configuration in the **elytron-oidc-client** subsystem **secure-deployment** attribute takes precedence over the configuration in the application deployment descriptor.

Additional resources

- [OpenID Connect specification](#)
- [OpenID Connect Libraries](#)
- [Securing applications using OpenID Connect with Red Hat build of Keycloak](#)

3.5.2. Creating an application secured with OpenID Connect

For creating a web-application, create a Maven project with the required dependencies and the directory structure. Create a web application containing a servlet that returns the user name obtained from the logged-in user's principal and attributes. If there is no logged-in user, the servlet returns the text "NO AUTHENTICATED USER".

Prerequisites

- You have installed Maven. For more information, see [Downloading Apache Maven](#).

Procedure

1. Set up a Maven project using the **mvn** command. The command creates the directory structure for the project and the **pom.xml** configuration file.

Syntax

```
$ mvn archetype:generate \
-DgroupId=${group-to-which-your-application-belongs} \
-DartifactId=${name-of-your-application} \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

Example

```
$ mvn archetype:generate \
-DgroupId=com.example.app \
-DartifactId=simple-webapp-example \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

2. Navigate to the application root directory:

Syntax

```
$ cd <name-of-your-application>
```

Example

```
$ cd simple-webapp-example
```

3. Replace the content of the generated **pom.xml** file with the following text:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example.app</groupId>
  <artifactId>simple-webapp-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>simple-webapp-example Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
<version.maven.war.plugin>3.3.2</version.maven.war.plugin>
<version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>
<version.server>8.0.0.GA-redhat-00009</version.server>
<version.bom.ee>${version.server}</version.bom.ee>
</properties>

<repositories>
  <repository>
    <id>jboss</id>
    <url>https://maven.repository.redhat.com/ga/</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </repository>
</repositories>

<pluginRepositories>
  <pluginRepository>
    <id>jboss</id>
    <url>https://maven.repository.redhat.com/ga/</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </pluginRepository>
</pluginRepositories>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee-with-tools</artifactId>
      <version>${version.bom.ee}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.wildfly.security</groupId>
    <artifactId>wildfly-elytron-auth-server</artifactId>
  </dependency>
</dependencies>

<build>
  <finalName>${project.artifactId}</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
```



```

    <artifactId>maven-war-plugin</artifactId>
    <version>${version.maven.war.plugin}</version>
</plugin>
<plugin>
  <groupId>org.jboss.eap.plugins</groupId>
  <artifactId>eap-maven-plugin</artifactId>
  <version>${version.eap.plugin}</version>
  <configuration>
    <channels>
      <channel>
        <manifest>
          <groupId>org.jboss.eap.channels</groupId>
          <artifactId>eap-8.0</artifactId>
        </manifest>
      </channel>
    </channels>
    <feature-packs>
      <feature-pack>
        <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
      </feature-pack>
      <feature-pack>
        <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
      </feature-pack>
    </feature-packs>
    <layers>
      <layer>cloud-server</layer>
      <layer>elytron-oidc-client</layer>
    </layers>
    <galleon-options>
      <jboss-fork-embedded>>true</jboss-fork-embedded>
    </galleon-options>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>package</goal>
      </goals>
    </execution>
  </executions>
</plugin>
</plugins>
</build>
</project>

```



NOTE

- `<version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>` is an example version of JBoss EAP Maven plugin. See the Red Hat Maven repository for more information on JBoss EAP Maven plugin releases: <https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/eap-maven-plugin/>.

4. Create a directory to store the Java files.

Syntax

```
$ mkdir -p src/main/java/<path_based_on_artifactID>
```

Example

```
$ mkdir -p src/main/java/com/example/app
```

5. Navigate to the new directory.

Syntax

```
$ cd src/main/java/<path_based_on_artifactID>
```

Example

```
$ cd src/main/java/com/example/app
```

6. Create a file **SecuredServlet.java** with the following content:

```
package com.example.app;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.Principal;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.wildfly.security.auth.server.SecurityDomain;
import org.wildfly.security.auth.server.SecurityIdentity;
import org.wildfly.security.authz.Attributes;
import org.wildfly.security.authz.Attributes.Entry;
/**
 * A simple secured HTTP servlet. It returns the user name and
 * attributes obtained from the logged-in user's Principal. If
 * there is no logged-in user, it returns the text
 * "NO AUTHENTICATED USER".
 */
@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {

            Principal user = req.getUserPrincipal();
```

```

SecurityIdentity identity = SecurityDomain.getCurrent().getCurrentSecurityIdentity();
Attributes identityAttributes = identity.getAttributes();
Set <String> keys = identityAttributes.keySet();
String attributes = "<ul>";

for (String attr : keys) {
    attributes += "<li> " + attr + " : " + identityAttributes.get(attr).toString() + "</li>";
}

attributes+="</ul>";
writer.println("<html>");
writer.println(" <head><title>Secured Servlet</title></head>");
writer.println(" <body>");
writer.println(" <h1>Secured Servlet</h1>");
writer.println(" <p>");
writer.print(" Current Principal ");
writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
writer.print("");
writer.print(user != null ? "\n" + attributes : "");
writer.println(" </p>");
writer.println(" </body>");
writer.println("</html>");
}
}
}

```

7. Configure the application's **web.xml** to protect the application resources.

Example

```

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    metadata-complete="false">

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>secured</web-resource-name>
            <url-pattern>/secured</url-pattern>
        </web-resource-collection>

        <auth-constraint>
            <role-name>Users</role-name>
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>OIDC</auth-method>
    </login-config>

    <security-role>

```

```

    <role-name>*</role-name>
  </security-role>
</web-app>

```

In this example, only the users with the role **Users** can access the application.

3.5.3. Deploying the application on OpenShift

As a JBoss EAP application developer, you can deploy your applications on OpenShift that uses the OpenID Connect subsystem and integrate it with a Red Hat build of Keycloak server. Deploy your application by following the procedures below.

Prerequisites

You have configured the Red Hat build of Keycloak server in your OpenShift with the following configuration. For more information, see [Red Hat build of Keycloak Operator](#).

- Create a realm called **JBossEAP**.
- Create a user called **demo**.
- Set a password for the user called **demo**. Toggle **Temporary** to **OFF** and click **Set Password**. In the confirmation prompt, click **Set password**.
- Create a role called **Users**.
- Assign the role **Users** to the user **demo**.
- In the **Client Roles** field, select the **realm-management** you configured for JBoss EAP.
- Assign the role **create-client** to the client **realm-management**.

Procedure

1. Deploy your application code to Git Repository.
2. Create a secret containing the OIDC configuration.
 - a. Create a file named **oidc-secret.yaml** using the following content:

```

apiVersion: v1
kind: Secret
metadata:
  name: oidc-secret
type: Opaque
stringData:
  OIDC_PROVIDER_NAME: rh-sso
  OIDC_USER_NAME: demo
  OIDC_USER_PASSWORD: demo
  OIDC_SECURE_DEPLOYMENT_SECRET: mysecret

```

- b. Use the following command to create a secret:

```
$ oc apply -f oidc-secret.yaml
```

3. Create a file named **helm.yaml** using the following content:

```

build:
  uri: [URL TO YOUR GIT REPOSITORY]
deploy:
  envFrom:
  - secretRef:
    name: oidc-secret

```

4. Deploy the example application using JBoss EAP Helm charts:

```
$ helm install eap-oidc-test-app -f helm.yaml jboss-eap/eap8
```

5. Add the environment variables to the **oidc-secret.yaml** file to configure the OIDC provider URL and application hostname.

```

yaml
stringData:
  ...
  OIDC_HOSTNAME_HTTPS: <host of the application>
  OIDC_PROVIDER_URL: https://<host of the SSO provider>/realms/JBossEAP

```

The value for **OIDC_HOSTNAME_HTTPS** corresponds to the following output:

```
echo $(oc get route eap-oidc-test-app --template='{{ .spec.host }}')
```

The value for **OIDC_PROVIDER_URL** corresponds to the following output:

```
echo https://$(oc get route sso --template='{{ .spec.host }}')/realms/JBossEAP
```

A route discovery attempt is made if **OIDC_HOSTNAME_HTTP(S)** is not set. To enable route discovery, the OpenShift user must be able to list the **route** resources. For example, to create and associate the **routeview** role with the **view** user, use the following **oc** command:

```

$ oc create role <role-name> --verb=list --resource=route

$ oc adm policy add-role-to-user <role-name> <user-name> --role-namespace=<your namespace>

```

6. Update the secret with **oc apply -f oidc-secret.yaml**.
7. Deploy the application again to ensure OpenShift uses the new environment variables:

```
$ oc rollout restart deploy eap-oidc-test-app
```

Verification

1. In your browser, navigate to **https://<eap-oidc-test-app route>/**. You will be redirected to Red Hat build of Keycloak login page.
2. Access the secured servlet.
3. Log in with the following credentials:

```
username: demo
password: demo
```

A page appears that contains the Principal ID.

3.5.4. Environment variable based configuration

Use these environment variables to configure JBoss EAP OIDC support on OpenShift image.

Table 3.1. Environment Variables

Environment variable	Legacy SSO environment variable	Description	Required	Default Value
OIDC_PROVIDER_NAME	NONE. When SSO_* environment variable are used, "rh-sso" name is internally set.	You must set to rh-sso when using <code>OIDC_PROVIDER_NAME</code> variable.	Yes	
OIDC_PROVIDER_URL	\$\$\$SO_URL/realms/\$\$\$SO_REALM	The URL of the provider.	Yes	
OIDC_USERNAME	SSO_USERNAME	Dynamic client registration requires the username to receive a token.	Yes	
OIDC_USER_PASSWORD	SSO_PASSWORD	Dynamic client registration requires the user password to receive a token.	Yes	
OIDC_SECURE_DEPLOYMENT_SECRET	SSO_SECRET	It is known to both the secure-deployment subsystem and the authentication server client.	No	
OIDC_SECURE_DEPLOYMENT_PRINCIPAL_ATTRIBUTE	SSO_PRINCIPAL_ATTRIBUTE	Configure the value of the principal name.	No	Defaults to sub (ID token) for rh-sso . Typical value: preferred_username.
OIDC_SECURE_DEPLOYMENT_ENABLE_CORS	SSO_ENABLE_CORS	Enable CORS for Single Sign-On applications.	No	Defaults to False .

Environment variable	Legacy SSO environment variable	Description	Required	Default Value
OIDC_SECURE_DEPLOYMENT_BEARER_ONLY	SSO_BEARER_ONLY	Deployment that accepts only bearer token and does not support logging.	No	Defaults to False .
OIDC_PROVIDER_SSL_REQUIRED	NONE	Defaults to external, such as private and local address, but does not support https.	No	External
OIDC_PROVIDER_TRUSTSTORE	SSO_TRUSTSTORE	Specify the realm truststore file. If it is not set, the adapter cannot use a trust manager when processing HTTPS requests.	No	
OIDC_PROVIDER_TRUSTSTORE_DIR	SSO_TRUSTSTORE_DIR	Directory to find the realm truststore . If it is not set, the adapter cannot use a trust manager when processing HTTPS requests.	No	
OIDC_PROVIDER_TRUSTSTORE_PASSWORD	SSO_TRUSTSTORE_PASSWORD	Specify the realm truststore password. If it is not set, the adapter cannot use a trust manager when processing HTTPS requests.	No	
OIDC_PROVIDER_TRUSTSTORE_CERTIFICATE_ALIAS	SSO_TRUSTSTORE_CERTIFICATE_ALIAS	Specify the realm truststore alias. It is required to interact with the authentication server to register a client.	No	
OIDC_DISABLE_SSL_CERTIFICATE_VALIDATION	SSO_DISABLE_SSL_CERTIFICATE_VALIDATION	Disable certificate validation when interacting with the authentication server to register a client.	No	
OIDC_HOSTNAME_HTTP	HOSTNAME_HTTP	Hostname used for unsecure routes.	No	Routes are discovered.
OIDC_HOSTNAME_HTTPS	HOSTNAME_HTTPS	Hostname used for secured routes.	No	Secured routes are discovered.

Environment variable	Legacy SSO environment variable	Description	Required	Default Value
NONE	SSO_PUBLIC_KEY	Public key of the Single Sign-On realm. This option is not used, public key is automatically retrieved by the OIDC subsystem.	No	If set, a warning is displayed that this option is being ignored.

3.6. SECURING APPLICATIONS BY USING SAML

The Security Assertion Markup Language (SAML) serves as a data format and protocol that enables the exchange of authentication and authorization information between two parties. These two parties typically include an identity provider and a service provider. This information takes the form of SAML tokens containing assertions. Identity providers issue these SAML tokens to subjects to enable these subjects to authenticate with service providers. Subjects can reuse SAML tokens with multiple service providers, which enables browser-based Single Sign-On in SAML v2.

You can secure web applications by using the Galleon layers that the Keycloak SAML adapter feature pack provides.

For information about the Keycloak SAML adapter feature pack, see [Keycloak SAML adapter feature pack for securing applications by using SAML](#).

3.6.1. Keycloak SAML adapter feature pack for securing applications by using SAML

Keycloak SAML adapter Galleon pack is a Galleon feature pack that includes the **keycloak-saml** layer. Use the **keycloak-saml** layer in the feature pack to install the necessary modules and configurations in JBoss EAP. These modules and configurations are required if you want to use Red Hat build of Keycloak as an identity provider for Single Sign-On (SSO) when using SAML. When using the **keycloak-saml** SAML adapter Galleon layer for source-to-image (S2I), you can optionally use the SAML client feature that enables automatic registration with an Identity Service Provider (IDP), such as Red Hat build of Keycloak.

3.6.2. Configuring Red Hat build of Keycloak as SAML provider for OpenShift

Red Hat build of Keycloak is an identity and access management provider for securing web applications with Single Sign-On (SSO). It supports OpenID Connect, which is an extension to OAuth 2.0, and SAML.

The following procedure outlines the essential steps needed to secure applications with SAML. For more information, see [Red Hat build of Keycloak documentation](#).

Prerequisites

- You have administrator access to Red Hat build of Keycloak.
- Red Hat build of Keycloak is running. For more information, see [Red Hat build of Keycloak Operator](#).
- You used the **oc** login command to log in to OpenShift.

Procedure

1. [Create a Single Sign-On realm, users, and roles](#) .
2. Generate the key and certificate by using the Java **keytool** command:

```
keytool -genkeypair -alias saml-app -storetype PKCS12 -keyalg RSA -keysize 2048 -
keystore keystore.p12 -storepass password -dname "CN=saml-basic-auth,OU=EAP SAML
Client,O=Red Hat EAP QE,L=MB,S=Milan,C=IT" -ext ku:c=dig,keyEncipherment -validity 365
```

3. Import the keystore into a Java KeyStore (JKS) format:

```
keytool -importkeystore -deststorepass password -destkeystore keystore.jks -srckeystore
keystore.p12 -srcstoretype PKCS12 -srcstorepass password
```

4. Create a secret in OpenShift for the keystore:

```
$ oc create secret generic saml-app-secret --from-file=keystore.jks=./keystore.jks --
type=opaque
```



NOTE

These steps are only necessary when using the automatic SAML client registration feature. When JBoss EAP registers a new SAML client into Red Hat build of Keycloak as the **client-admin** user, JBoss EAP must store the certificate of the new SAML client in the Red Hat build of Keycloak client configuration. This allows JBoss EAP to retain the private key while only storing the public certificate in Red Hat build of Keycloak, which establishes an authenticated client for communication with Red Hat build of Keycloak.

3.6.3. Creating an application secured with SAML

You can enhance web application security by using the Security Assertion Markup Language (SAML). SAML provides effective user authentication and authorization, along with Single Sign-On (SSO) capabilities, making it a dependable choice for strengthening web applications.

Prerequisites

- You have installed Maven. For more information, see [Downloading Apache Maven](#).

Procedure

1. Set up a Maven project by using the **mvn** command. This command creates both the directory structure for the project and the **pom.xml** configuration file.

Syntax

```
$ mvn archetype:generate \
-DgroupId=${group-to-which-your-application-belongs} \
-DartifactId=${name-of-your-application} \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

Example

```
$ mvn archetype:generate \
-DgroupId=com.example.app \
-DartifactId=simple-webapp-example \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

2. Navigate to the application root directory:

Syntax

```
$ cd <name-of-your-application>
```

Example

```
$ cd simple-webapp-example
```

3. Replace the content of the generated **pom.xml** file with the following text:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example.app</groupId>
  <artifactId>simple-webapp-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>simple-webapp-example Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <version.maven.war.plugin>3.3.2</version.maven.war.plugin>
    <version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>
    <version.server>8.0.0.GA-redhat-00009</version.server>
    <version.bom.ee>${version.server}</version.bom.ee>
  </properties>

  <repositories>
    <repository>
      <id>jboss</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
    </repository>
  </repositories>
```

```

<pluginRepositories>
  <pluginRepository>
    <id>jboss</id>
    <url>https://maven.repository.redhat.com/ga/</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </pluginRepository>
</pluginRepositories>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee-with-tools</artifactId>
      <version>${version.bom.ee}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.wildfly.security</groupId>
    <artifactId>wildfly-elytron-auth-server</artifactId>
  </dependency>
</dependencies>

<build>
  <finalName>${project.artifactId}</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>${version.maven.war.plugin}</version>
    </plugin>
    <plugin>
      <groupId>org.jboss.eap.plugins</groupId>
      <artifactId>eap-maven-plugin</artifactId>
      <version>${version.eap.plugin}</version>
      <configuration>
        <channels>
          <channel>
            <manifest>
              <groupId>org.jboss.eap.channels</groupId>
              <artifactId>eap-8.0</artifactId>
            </manifest>
          </channel>
        </channels>
      </configuration>
    </plugin>
  </plugins>
</build>

```

```

<feature-packs>
  <feature-pack>
    <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.keycloak:keycloak-saml-adapter-galleon-pack</location>
  </feature-pack>
</feature-packs>
<layers>
  <layer>cloud-server</layer>
  <layer>keycloak-saml</layer>
</layers>
<galleon-options>
  <jboss-fork-embedded>true</jboss-fork-embedded>
</galleon-options>
</configuration>
<executions>
  <execution>
    <goals>
      <goal>package</goal>
    </goals>
  </execution>
</executions>
</plugin>
</plugins>
</build>
</project>

```



NOTE

- `<version.eap.plugin>1.0.0.Final-redhat-00014</version.eap.plugin>` is an example version of JBoss EAP Maven plugin. See the Red Hat Maven repository for more information on JBoss EAP Maven plugin releases: <https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/eap-maven-plugin/>.

4. Create a directory to store the Java files.

Syntax

```
$ mkdir -p src/main/java/<path_based_on_artifactID>
```

Example

```
$ mkdir -p src/main/java/com/example/app
```

5. Navigate to the new directory.

Syntax

```
$ cd src/main/java/<path_based_on_artifactID>
```

Example

```
$ cd src/main/java/com/example/app
```

6. Create a file named **SecuredServlet.java** that contains the following settings:

```
package com.example.app;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.Principal;
import java.util.Set;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.wildfly.security.auth.server.SecurityDomain;
import org.wildfly.security.auth.server.SecurityIdentity;
import org.wildfly.security.authz.Attributes;
/**
 * A simple secured HTTP servlet. It returns the user name and
 * attributes obtained from the logged-in user's Principal. If
 * there is no logged-in user, it returns the text
 * "NO AUTHENTICATED USER".
 */

@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {

            Principal user = req.getUserPrincipal();
            SecurityIdentity identity = SecurityDomain.getCurrent().getCurrentSecurityIdentity();
            Attributes identityAttributes = identity.getAttributes();
            Set <String> keys = identityAttributes.keySet();
            String attributes = "<ul>";

            for (String attr : keys) {
                attributes += "<li> " + attr + " : " + identityAttributes.get(attr).toString() + "</li>";
            }

            attributes+="</ul>";
            writer.println("<html>");
            writer.println(" <head><title>Secured Servlet</title></head>");
            writer.println(" <body>");
            writer.println(" <h1>Secured Servlet</h1>");
            writer.println(" <p>");
            writer.print(" Current Principal ");
            writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
```

```

writer.print("");
writer.print(user != null ? "\n" + attributes : "");
writer.println(" </p>");
writer.println(" </body>");
writer.println("</html>");
}
}
}

```

7. Create the directory structure for the **web.xml** file:

```

mkdir -p src/main/webapp/WEB-INF
cd src/main/webapp/WEB-INF

```

8. Configure the application's **web.xml** file to protect the application resources.

Example

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  metadata-complete="false">

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secured</web-resource-name>
      <url-pattern>/secured</url-pattern>
    </web-resource-collection>

    <auth-constraint>
      <role-name>user</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>KEYCLOAK-SAML</auth-method>
  </login-config>

  <security-role>
    <role-name>user</role-name>
  </security-role>
</web-app>

```

In this example, only users with the **user** role can access the application.

Verification

After creating the application, commit it to a remote Git repository.

1. Create a Git repository such as <https://github.com/your-username/simple-webapp-example>. For more information about remote repositories and Git, see [Getting started with Git - About remote repositories](#).

- From the root folder of the application, run the following Git commands:

```
git init -b main
git add pom.xml src
git commit -m "First commit"
git remote add origin git@github.com:your-username/simple-webapp-example.git
git remote -v
git push -u origin main
```

These steps commit your application to the remote repository, making it accessible online.

3.6.4. Building and deploying a SAML-secured application on OpenShift

You can build and deploy your application secured with SAML on OpenShift by using the JBoss EAP and Single Sign-On (SSO) Galleon layers.

Prerequisites

- You have installed Helm. For more information, see [Installing Helm](#).
- You have created the SAML application project and made it accessible in a Git repository.
- You have installed the repository for the JBoss EAP Helm charts by entering this command in the management CLI:

```
$ helm repo add jboss-eap https://jbossas.github.io/eap-charts/
```

Procedure

- Deploy your application code to the Git Repository.
- Create an OpenShift secret containing the required environment variables:

```
apiVersion: v1
kind: Secret
metadata:
  name: saml-secret
type: Opaque
stringData:
  SSO_REALM: "saml-basic-auth"
  SSO_USERNAME: "client-admin"
  SSO_PASSWORD: "client-admin"
  SSO_SAML_CERTIFICATE_NAME: "saml-app"
  SSO_SAML_KEYSTORE: "keystore.jks"
  SSO_SAML_KEYSTORE_PASSWORD: "password"
  SSO_SAML_KEYSTORE_DIR: "/etc/sso-saml-secret-volume"
  SSO_SAML_LOGOUT_PAGE: "/simple-webapp-example"
  SSO_DISABLE_SSL_CERTIFICATE_VALIDATION: "true"
```

- Save the provided YAML content to a file, such as **saml-secret.yaml**.
- Apply the saved YAML file by using the following command:

```
oc apply -f saml-secret.yaml
```

5. Create a file named **helm.yaml** that contains the following settings:

```
build:
  uri: [WEB ADDRESS TO YOUR GIT REPOSITORY]
deploy:
  volumes:
    - name: saml-keystore-volume
      secret:
        secretName: saml-app-secret
  volumeMounts:
    - name: saml-keystore-volume
      mountPath: /etc/sso-saml-secret-volume
      readOnly: true
envFrom:
  - secretRef:
      name: saml-secret
```



NOTE

Specify the web address in the HTTP format, such as **http://www.redhat.com**. If you are using a maven mirror, specify the web address as follows:

```
build:
  uri: [WEB ADDRESS TO YOUR GIT REPOSITORY]
env:
  - name: "MAVEN_MIRROR_URL"
    value: "http://..."
```

6. Deploy the example application by using JBoss EAP Helm charts:

```
$ helm install saml-app -f helm.yaml jboss-eap/eap8
```

7. Add the environment variables to the **saml-secret.yaml** file to configure the Keycloak server URL and application route:

```
stringData:
  ...
  HOSTNAME_HTTPS: <saml-app application route>
  SSO_URL: https://<host of the Keycloak server>
```

Replace **<saml-app application route>** and **<host of the Keycloak server>** with the appropriate values.

The value for **HOSTNAME_HTTPS** corresponds to the following output:

```
echo $(oc get route saml-app --template='{{ .spec.host }}')
```

The value for **SSO_URL** corresponds to the following output:

```
echo https://$(oc get route sso --template='{{ .spec.host }}')
```


**NOTE**

If you cannot use this command, use **oc get routes** to list the available routes and select the route to your Red Hat build of Keycloak instance.

8. Update the secret with **oc apply -f saml-secret.yaml**.

Verification

1. Deploy the application again to ensure that OpenShift uses the new environment variables:

```
$ oc rollout restart deploy saml-app
```

2. In a browser, navigate to the application URL. For example, **https://<saml-app route>/simple-webapp-example**.

You are redirected to the Red Hat build of Keycloak login page.

3. To get the web address, use the following command to access the secured servlet:

```
echo https://$(oc get route saml-app --template='{{ .spec.host }}')/simple-webapp-example/secured
```

4. Log in with the following credentials:

```
username: demo
password: demo
```

A page is displayed that contains the Principal ID.

Your application is now secured using SAML.

3.6.5. Creating a SSO realm, users, and roles

You can configure a Single Sign-On (SSO) realm, define user roles, and manage access control in your Red Hat build of Keycloak environment. These actions enable you to enhance security and simplify user access management, ensuring a streamlined authentication experience. This is essential for optimizing your SSO setup and improving user authentication processes.

Prerequisites

- You have administrator access to Red Hat build of Keycloak.
- Red Hat build of Keycloak is running.

Procedure

1. Log in to the Red Hat build of Keycloak admin console using the URL: **https://<SSO route>/**.
2. Create a realm in Red Hat build of Keycloak; for example, **saml-basic-auth**. You can subsequently use this realm to create the required users, roles, and a client. For more information, see [Creating a realm](#).
3. Create a role within the **saml-basic-auth** realm. For example, **user**. For more information, see [Creating a realm role](#).

4. Create a user. For example, **demo**.
For more information, see [Creating users](#).
5. Create a password for the user. For example, **demo**.
Ensure that the password is not temporary. For more information, see [Setting a password for a user](#).
6. Assign the **user** role to the **demo** user for login access.
For more information, see [Assigning role mappings](#).
7. Create a user. For example, **client-admin**.
To create the SAML client in the Keycloak server when the JBoss EAP server starts, you can use the **client-admin** user, which requires additional privileges. For more information, see [Creating users](#).
8. Create a password for the user. For example, **client-admin**.
Ensure that the password is not temporary. For more information, see [Setting a password for a user](#).
9. Select **realm-management** from the **Client Roles** drop down list.
10. Assign the roles **create-client**, **manage-clients**, and **manage-realm** to the **client-admin** user.
For more information, see [Assigning role mappings](#).

3.6.6. Environment variables for configuring the SAML subsystem

You can optimize the integration of the Keycloak server within your environment by understanding and using the following variables. This ensures a seamless and secure Keycloak setup for your application.

Table 3.2. Environment variables

Environment variable	Description	Required
APPLICATION_NAME	Used as a prefix for the client name, derived from the deployment name.	Optional
HOSTNAME_HTTP	Custom hostname for the HTTP OpenShift route. If not set, route discovery is performed.	Optional
HOSTNAME_HTTPS	Custom hostname for the HTTPS OpenShift route. If not set, route discovery is performed.	Optional
SSO_DISABLE_SSL_CERTIFICATE_VALIDATION	Choose between true or false to enable or disable validation of the Keycloak server certificate. Consider setting this to true when the SSO server generates a self-signed certificate.	Optional
SSO_PASSWORD	The password for a user with privileges to interact with the Keycloak realm and to create and register clients. For example, client-admin .	True
SSO_REALM	The SSO realm for associating application clients. For example, saml-basic-auth .	Optional

Environment variable	Description	Required
SSO_SAML_CERTIFICATE_NAME	Alias of private key and certificate in the SAML client keystore. For example, saml-app .	True
SSO_SAML_KEYSTORE	Name of the keystore file. For example, keystore.jks .	True
SSO_SAML_KEYSTORE_DIR	Directory that contains the client keystore. For example, /etc/sso-saml-secret-volume .	True
SSO_SAML_KEYSTORE_PASSWORD	Keystore password. For example, password .	True
SSO_SAML_LOGOUT_PAGE	Logout page. For example, simple-webapp-example .	True
SSO_SAML_VALIDATE_SIGNATURE	Specify true to validate the signature or false to not validate it. True by default.	Optional
SSO_SECURITY_DOMAIN	The name of the security domain used to secure undertow and ejb subsystems. The default is keycloak .	Optional
SSO_TRUSTSTORE	The truststore file name containing the server certificate.	Optional
SSO_TRUSTSTORE_CERTIFICATE_ALIAS	Certificate alias within the truststore.	Optional
SSO_TRUSTSTORE_DIR	Directory that contains the truststore.	Optional
SSO_TRUSTSTORE_PASSWORD	The password for the truststore and certificate . For example, mykeystorepass .	Optional
SSO_URL	The URL for the SSO server. For example, <SSO server accessible route> .	True
SSO_USERNAME	The username of a user with privileges to interact with the Keycloak realm and to create and register clients. For example, client-admin .	True

3.6.7. Route discovery in JBoss EAP server

You can optimize your server's performance and simplify route configurations in your specified namespace by using the route discovery feature in the JBoss EAP server. This feature is essential for improving server efficiency to provide a smoother operational experience, particularly when the **HOSTNAME_HTTPS** variable is unspecified.

If the **HOSTNAME_HTTPS** variable is not set, the JBoss EAP server automatically attempts route discovery. To enable route discovery, you must create the required permissions:

```
oc create role routeview --verb=list --resource=route -n YOUR_NAME_SPACE
oc policy add-role-to-user routeview system:serviceaccount:YOUR_NAME_SPACE:default --role-namespace=YOUR_NAME_SPACE -n YOUR_NAME_SPACE
```

3.6.8. Additional resources

- [Red Hat build of Keycloak Server Administration Guide](#)

3.7. ADDITIONAL RESOURCES

- [OpenShift Container Platform Getting Started](#)

CHAPTER 4. USING HELM CHARTS TO BUILD AND DEPLOY JBOSS EAP APPLICATIONS ON OPENSHIFT

Helm is an open-source package manager that enables you to build, deploy, and maintain your JBoss EAP applications on OpenShift. In JBoss EAP 8.0, Helm charts replace the OpenShift templates.

4.1. HELM CHART USE CASE

You can use Helm charts with JBoss EAP 8.0 to:

- Build your application from a Maven project hosted on a Git repository using OpenShift Source-to-Image (S2I).
- Deploy an application image on OpenShift with deep integration with the OpenShift cluster (TLS configuration, public route to expose the application, and so on).
- Build your application image with Helm chart and use the JBoss EAP operator to deploy the image.
- Build an application image for JBoss EAP using other methods and use the Helm chart to deploy the application image.

4.2. HELM CHART CUSTOMIZATION FOR JBOSS EAP ON OPENSHIFT

You can customize Helm chart for your JBoss EAP application by modifying the **YAML** file that contains specific settings for your application.

In the **YAML** file, there are two main sections:

- The **build** configuration.
- The **deploy** configuration.

By selecting configure via **YAML view**, You can edit the **Values files** file directly on your OpenShift Development Console to upgrade your Helm release with an updated configuration.

Additional resources

- [Helm chart on JBoss EAP 8](#)
- [Value file examples](#)

4.3. PROVISIONING JBOSS EAP WITH S2I

Use the **eap-maven-plugin** from the application **pom.xml** to provision your JBoss EAP server.



NOTE

The **build.s2i.featurePacks**, **build.s2i.galleonLayers** and **build.s2i.channels** fields have been deprecated.

4.4. BUILDING AND DEPLOYING JBOSS EAP APPLICATIONS USING HELM CHARTS

You can build your JBoss EAP application using Helms chart by configuring the **build** and **deploy** values. You must provide a URL to the Git repository that hosts your application code in your **build** configuration, the output is an **ImageStreamTag** resource that contains the built application image.

To deploy your application, you must provide an **ImageStreamTag** resource that contains your built application image. The output is your deployed application and other related resources you can use to access your application from inside and outside OpenShift.

Prerequisites

- You have logged into the OpenShift Development Console.
- You have JBoss EAP application hosted in a Git repository.
- Your application is a Maven project
- You have configured your application to use the **org.jboss.eap.plugins:eap-maven-plugin** to provision your JBoss EAP 8.0 server.

Procedure

1. Build your application image from the source repository:

```
build:
  uri: <git repository URL of your application>
```

2. Optional: Enter the secret in the **build** section:

```
build:
  sourceSecret: <name of secret login to your Git repository>
```

Verification

- If your application has been successfully deployed, you should see a deployed badge next to the Helm release on OpenShift Development Console.

Additional resources

- [Provisioning a JBoss EAP server using the Maven plug-in](#)

4.5. BUILDING YOUR APPLICATION IMAGE USING THE OPENSIFT DEVELOPMENT CONSOLE

You can build your JBoss EAP application image using Helm chart by configuring the **build** section on the OpenShift Development Console.



NOTE

If the application image has been built by another mechanism, you can skip the building part of the Helm chart by setting the **build.enabled** field to **false**.



IMPORTANT

You must specify the **build.url** field with the Git URL that references your Git repository.

Additional resources

- [Helm chart for JBoss EAP 8.0.](#)
- [Building the application image.](#)

4.6. DEPLOYING YOUR APPLICATION IMAGE

You can deploy your JBoss EAP application using Helm chart by configuring the **deploy** setting on the OpenShift Development Console.



NOTE

If you built your application image using another mechanism, you can skip the deployment configuration of the Helm chart by setting the **build.deploy** field to **false**.

Additional resources

- [OpenShift Development Console Quickstarts.](#)
- [Deploying the application image.](#)

4.6.1. OpenShift volumes for persistent data storage in Helm chart

OpenShift volumes enable containers to store and share data from various sources, including cloud storage, network file systems (NFS), or host machines. You can use Helm chart, an OpenShift package manager, to deploy applications in a consistent and reproducible manner. By adding a volume mount to a Helm chart, you can enable your application to persist data across deployments.

4.6.2. Mounting a volume with a Helm chart

This procedure explains how to mount a **secret** as a volume using a Helm chart on JBoss EAP 8.0. Additionally, you can also use it to mount a **ConfigMap**. This action enables the application to securely access and use the data, protecting it from unauthorized access or tampering.

For example, by mounting a **secret** as a volume, the sensitive data that you store in the secret appear as a file in the POD running the deployment where the secret has been mounted.

Prerequisites

- You have created a **secret**. For example, you have created a secret named **eap-app-secret** that refers to a file like **keystore.jks**.
- You have identified a location where to mount the secret in the container's file system. For example, the directory **/etc/jgroups-encrypt-secre-secret-volume** is where the secret file, such as **keystore.jks** is mounted.

Procedure

1. Specify a **volume** in the **deploy.volumes** field and configure the secret to be used. You must provide the **name** of the volume and the **secretName** of the secret:

```
volumes:  
  - name: eap-jgroups-keystore-volume  
    secret:  
      secretName: eap-app-secret
```

2. Mount the volume on the file system using the **deploy.volumeMounts** in the deployment configuration:

```
volumeMounts:  
  - name: eap-jgroups-keystore-volume  
    mountPath: /etc/jgroups-encrypt-secret-volume  
    readOnly: true
```

When the pod starts, the container mounts the **keystore.jks** file at **/etc/jgroups-encrypt-secret-volume/keystore.jks** location.

Additional resources

- [Volumes \(Kubernetes documentation\)](#)

CHAPTER 5. ENVIRONMENT VARIABLES AND MODEL EXPRESSION RESOLUTION

5.1. PREREQUISITES

- You have some basic knowledge of how to configure environment variables on an operating system.
- For configuring environment variables on the OpenShift Container Platform, you must meet the following prerequisites:
 - You have already installed OpenShift and set up the OpenShift CLI ("oc"). For more information about the oc, see [Getting Started with the OpenShift CLI](#).
 - You have deployed your application to OpenShift using a Helm chart. For more information about Helm charts, see [Helm Charts for JBoss EAP](#).

5.2. ENVIRONMENT VARIABLES FOR RESOLVING MANAGEMENT MODEL EXPRESSIONS

To resolve management model expressions and to start your JBoss EAP 8.0 server on the OpenShift Container Platform, you can either add environment variables or set Java system properties in the management command-line interface (CLI). If you use both, JBoss EAP observes and uses the Java system property rather than the environment variable to resolve the management model expression.

System property to environment variable mapping

Imagine that you have this management expression: `${my.example-expr}`. When your JBoss EAP server tries to resolve it, it checks for a system property named `my.example-expr`.

- If your server finds this property, it uses its value to resolve the expression.
- If it doesn't find this property, your server continues searching.

Next, assuming that your server does not find system property `my.example-expr`, it automatically changes `my.example-expr` to all uppercase letters and replaces all characters that aren't alphanumeric with underscores (`_`): `MY_EXAMPLE_EXPR`. JBoss EAP then checks for an environment variable with that name.

- If your server finds this variable, it uses its value to resolve the expression.
- If it doesn't find this variable, your server continues searching.

TIP

If your original expression starts with the prefix `env.`, JBoss EAP resolves the environment variable by removing the prefix, then looking for only the environment variable name. For example, for the expression `env.example`, JBoss EAP looks for an `example` environment variable.

If none of these checks finds a property or variable to resolve your original expression, JBoss EAP looks for whether the expression has a default value. If it does, that default value resolves the expression. If not, then JBoss EAP can't resolve the expression.

Example with two servers

Suppose that, on one server, JBoss EAP defines this management resource: **<socket-binding-group name="standard-sockets" default-interface="public" port-offset="\${jboss.socket.binding.port-offset:0}">**. To run a second server with a different port offset, instead of editing the configuration file, do one of the following:

- Set the **jboss.socket.binding.port-offset** Java system property to resolve the value on the second server: **./standalone.sh -Djboss.socket.binding.port-offset=100.**
- Set the **JBOSS_SOCKET_BINDING_PORT_OFFSET** environment variable to resolve the value on the second server: **JBOSS_SOCKET_BINDING_PORT_OFFSET=100 ./standalone.sh.**

5.3. CONFIGURING ENVIRONMENT VARIABLES ON THE OPENSIFT CONTAINER PLATFORM

With JBoss EAP 8.0, you can configure environment variables to resolve management model expressions. You can also use environment variables to adapt the configuration of the JBoss EAP server you're running on OpenShift.

Set environment variables and options on a resource that uses a pod template:

```
$ oc set env <object-selection> KEY_1=VAL_1 ... KEY_N=VAL_N [<set-env-options>] [<common-options>]
```

Option	Description
-e, --env=<KEY>=<VAL>	Set given key-value pairs of environment variables.
--overwrite	Confirm update of existing environment variables.

NOTE

Kubernetes workload resources that use pod templates include the following:

- **Deployment**
- **ReplicaSet**
- **StatefulSet**
- **DaemonSet**
- **Job**
- **CronJob**

After you configure your environment variables, the JBoss EAP management console should display them in the details for their related pods.

Additional resources

- [About the OpenShift CLI](#)

- [Red Hat JBoss Enterprise Application Platform Configuration Guide](#)

5.4. OVERRIDING MANAGEMENT ATTRIBUTES WITH ENVIRONMENT VARIABLES

You know that you can use a Java system property or an environment variable to resolve a management attribute that's defined with an expression, but you can also modify other attributes, even if they don't use expressions.

To more easily adapt your JBoss EAP server configuration to your server environment, you can use an environment variable to override the value of any management attribute, without ever having to edit your configuration file. This feature, which is available starting with the JBoss EAP version 8.0, is useful for the following reasons:

- JBoss EAP provides expressions for only its most common management attributes. Now, you can change the value of an attribute that has no defined expression.
- Some management attributes connect your JBoss EAP server with other services, such as a database, whose values you can't know in advance, or whose values you can't store in a configuration; for example, in database credentials. By using environment variables, you can defer the configuration of such attributes while your JBoss EAP server is running.



IMPORTANT

This feature is enabled by default, starting with JBoss EAP version 8.0 OpenShift runtime image. To enable it on other platforms, you must set the **WILDFLY_OVERRIDING_ENV_VARS** environment variable to any value; for example, **export WILDFLY_OVERRIDING_ENV_VARS=1**.



NOTE

You can't override management attributes whose **type** is **LIST**, **OBJECT**, or **PROPERTY**.

Prerequisites

- You must have defined a management attribute that you now want to override.

Procedure

To override a management attribute with an environment variable, complete the following steps:

1. Identify the path of the resource and attribute you want to change. For example, set the value of the **proxy-address-forwarding** attribute to **true** for the resource **/subsystem=undertow/server=default-server/http-listener=default**.
2. Create the name of the environment variable to override this attribute by mapping the resource address and the management attribute, as follows:
 - a. Remove the first slash (/) from the resource address:
/subsystem=undertow/server=default-server/http-listener=default becomes **subsystem=undertow/server=default-server/http-listener=default**.
 - b. Append two underscores (__) and the name of the attribute; for example:
subsystem=undertow/server=default-server/http-listener=default__proxy-address-forwarding.

- c. Replace all non-alphanumeric characters with an underscore (_), and put the entire line of code in all capital letters:

SUBSYSTEM_UNDERTOW_SERVER_DEFAULT_SERVER_HTTP_LISTENER_DEFAULT_PROXY_ADDRESS_FORWARDING.

3. Set the environment value:

SUBSYSTEM_UNDERTOW_SERVER_DEFAULT_SERVER_HTTP_LISTENER_DEFAULT_PROXY_ADDRESS_FORWARDING=true.



NOTE

These values are examples that you must replace with your actual configuration values.

CHAPTER 6. PROVISIONING A JBOSS EAP SERVER USING THE MAVEN PLUG-IN

Using JBoss EAP Maven plug-in, you can configure a server according to your requirements by including only those Galleon layers that provide the capabilities that you need, in your server.

6.1. JBOSS EAP MAVEN PLUG-IN

The JBoss EAP Maven plug-in uses Galleon trimming capability to reduce the size and memory footprint of the server. The JBoss EAP Maven plug-in supports the execution of JBoss EAP CLI script files to customize your server configuration. A CLI script includes a list of CLI commands for configuring the server.

You can retrieve the latest Maven plug-in version from the Maven repository, which is available at [Index of /ga/org/jboss/eap/plugins/eap-maven-plugin](https://index.of/ga/org/jboss/eap/plugins/eap-maven-plugin). In a Maven project, the **pom.xml** file contains the configuration of the JBoss EAP Maven plug-in.

The JBoss EAP Maven plug-in provisions the server and deploys the packaged application, such as WAR, to the provisioned server during the Maven execution. The provisioned server on which your application is deployed is located in **target/server** directory. The JBoss EAP Maven plug-in also provides the following functionality:



NOTE

The server in **target/server** is not supported and is available only for debugging or development purposes.

- Uses the **org.jboss.eap:wildfly-ee-galleon-pack** and **org.jboss.eap.cloud:eap-cloud-galleon-pack** Galleon **feature-pack** and some of its layers for customizing the server configuration file.
- Applies CLI script commands to the server.
- Supports the addition of extra files into the server installation, such as a **keystore** file.

6.2. CREATING A JAKARTA EE 10 APPLICATION WITH THE MAVEN

Create an application that prints “Hello World!” when you access it.

Prerequisites

- You have installed JDK 17.
- You have installed the Maven 3.6 or later version. For more information, see [Downloading Apache Maven](#).

Procedure

1. Set up the Maven project.

```
$ mvn archetype:generate \
-DgroupId=GROUP_ID \
-DartifactId=ARTIFACT_ID \
```

```
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

Where *GROUP_ID* is the **groupId** of your project and *ARTIFACT_ID* is the **artifactId** of your project.

- To configure the Maven to automatically manage versions for the Jakarta EE artifacts in the **jboss-eap-ee** BOM, add the BOM to the **<dependencyManagement>** section of the project **pom.xml** file. For example:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>8.0.0.GA-redhat-00009</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```



NOTE

- <version>A.B.C-redhat-XXXXX</version>** Where **A.B.C** is the release number and **XXXXX** is build number of your JBoss EAP instance. See the Red Hat Maven repository for version details about JBoss EAP releases. The release and build numbers are available for all JBoss EAP releases. <https://maven.repository.redhat.com/earlyaccess/all/org/jboss/bom/jboss-eap-ee/>.

- Add the servlet API artifact, which is managed by the BOM, to the **<dependencies>** section of the project **pom.xml** file, as shown in the following example:

```
<dependency>
  <groupId>jakarta.servlet</groupId>
  <artifactId>jakarta.servlet-api</artifactId>
</dependency>
```

- Create a Java file **TestServlet.java** with the following content and save the file in the **APPLICATION_ROOT/src/main/java/com/example/simple/** directory.

```
package com.example.simple;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
@WebServlet(urlPatterns = "/hello")
public class TestServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
```

```

IOException {
    PrintWriter writer = resp.getWriter();
    writer.println("Hello World!");
    writer.close();
}
}

```

You can now deploy this application on JBoss EAP or update this application to package it with and deploy it on a custom provisioned JBoss EAP server using the Maven plug-in.

6.3. USING THE MAVEN PLUG-IN TO PROVISION A JBOSS EAP SERVER

Update the **pom.xml** of an application to package it with and deploy on a custom provisioned JBoss EAP server using the Maven plug-in. You can then deploy the application running on the custom-provisioned JBoss EAP server on OpenShift.

Prerequisites

- Ensure that the JBoss EAP Maven plug-in and the JBoss EAP Maven artifact are accessible from either your local or remote Maven repositories.
- You have installed JDK 17.
- You have installed Maven. For more information, see [Downloading Apache Maven](#).



NOTE

If you are using JDK 17 and Maven 3.8.5 or previous Maven version, use the latest Maven WAR plugin.

- You have created a Maven project for Jakarta EE 10 application. For more information, see [Creating a Jakarta EE 10 application with the Maven](#).

Procedure

1. Configure Maven to retrieve the JBoss EAP BOM and JBoss EAP Maven plug-in from a remote repository by adding the following content to the **pom.xml** file:

```

<repositories>
  <repository>
    <id>jboss</id>
    <url>https://maven.repository.redhat.com/ga/</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>jboss</id>
    <url>https://maven.repository.redhat.com/ga/</url>
    <snapshots>
      <enabled>>false</enabled>

```

```

</snapshots>
</pluginRepository>
</pluginRepositories>

```

2. Add the following content to the **<build>** element of the **pom.xml** file. You must specify the latest version of the JBoss EAP Maven plug-in. For example:

```

<plugins>
  <plugin>
    <groupId>org.jboss.eap.plugins</groupId>
    <artifactId>eap-maven-plugin</artifactId>
    <version>1.0.0.Final-redhat-00014</version> 1
    <configuration>
      <channels>
        <channel>
          <manifest>
            <groupId>org.jboss.eap.channels</groupId> 2
            <artifactId>eap-8.0</artifactId>
          </manifest>
        </channel>
      </channels>
      <feature-packs>
        <feature-pack>
          <location>org.jboss.eap:wildfly-ee-galleon-pack</location> 3
        </feature-pack>
        <feature-pack>
          <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location> 4
        </feature-pack>
      </feature-packs>
      <layers>
        <layer>cloud-server</layer> 5
      </layers>
      <runtime-name>ROOT.war</runtime-name> 6
    </configuration>
    <executions>
      <execution>
        <goals>
          <goal>package</goal> 7
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>

```

- 1** **<version>1.0.0.Final-redhat-00014</version>** is an example version of JBoss EAP Maven plugin. See the Red Hat Maven repository for more information on JBoss EAP Maven plugin releases:
<https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/plugins/eap-maven-plugin/>.
- 2** This specifies the JBoss EAP 8.0 channel in which the JBoss EAP server artifacts are defined.
- 3** You can retrieve the version of this feature pack from the JBoss EAP channel. The Galleon **feature-pack** includes Galleon layers such as **cloud-server** for provisioning trimmed JBoss

EAP servers.

- 4 This feature pack adjusts the server Galleon layers for the cloud. It is necessary to use this feature pack to build applications for OpenShift.
- 5 This Galleon layer provisions a server with features that are necessary when running JBoss EAP applications in the cloud.
- 6 With this configuration option, you can register your deployment in the HTTP root context.
- 7 With this plug-in goal, you can provision the server, deploy your application, apply custom configured CLI scripts, and copy custom content into the server installation.

3. Package the application.

```
$ mvn package
```

The directory **target/server** contains a server and application that are ready for use for debugging or development purposes. In the JBoss EAP S2I build context, the server provisioned by the JBoss EAP maven-plugin is installed in the JBoss EAP image at the **/opt/server** location. For more information see [Building Applications Images using Source-to-Image \(S2I\) in OpenShift](#).



NOTE

If you use the **mvn package** command with debugging enabled (**-X** option), include the property **-Dorg.slf4j.simpleLogger.log.com.networknt.schema=off** to prevent excessive debug logging during schema validation.

Verification

- You can check the generated server configuration file **target/server/standalone/configuration/standalone.xml** that contains the provisioned subsystems and application deployment.

The JBoss EAP server that contains your deployment has been provisioned.

Additional resources

- [Available JBoss EAP layers](#).
- [Use the Maven Repository](#).
- [Introduction to the Standard Directory Layout in the Apache Maven documentation](#) .

6.4. THE GALLEON PROVISIONING FILE

Provisioning files are XML files with the name **provisioning.xml** that you can store in the **galleon** subdirectory. Using them is an alternative to configuring feature packs and layers in the JBoss EAP Maven plug-in. You can configure **provisioning.xml** file to fine-tune the provisioning process.

The following code demonstrates a provisioning file content that you can use to provision JBoss EAP server based on the **cloud-server** layer.



NOTE

The JBoss EAP feature packs don't have versions, versions are retrieved from the configured channel in the Maven plug-in.

```
<?xml version="1.0" ?>
<installation xmlns="urn:jboss:galleon:provisioning:3.0">
  <feature-pack location="org.jboss.eap:wildfly-ee-galleon-pack:"> 1
    <default-configs inherit="false"/> 2
    <packages inherit="false"/> 3
  </feature-pack>
  <feature-pack location="org.jboss.eap.cloud:eap-cloud-galleon-pack:
"> 4
    <default-configs inherit="false"/>
    <packages inherit="false"/>
  </feature-pack>
  <config model="standalone" name="standalone.xml"> 5
    <layers>
      <include name="cloud-server"/>
    </layers>
  </config>
  <options> 6
    <option name="optional-packages" value="passive+"/>
  </options>
</installation>
```

- 1 This element instructs the provisioning process to provision the JBoss EAP feature pack retrieved from the JBoss EAP channel.
- 2 This element instructs the provisioning process to exclude default configurations. You can retrieve default configurations in JBoss EAP server installation, such as **standalone.xml** and **standalone-ha.xml**. When you are provisioning JBoss EAP server from the JBoss EAP Maven plugin, generate a single server configuration based on the configured Galleon users. Setting the option to **false** prevents the generation of any additional server configurations. Setting **inherit=true** is not supported for both **default-configs** and **packages**.
- 3 This element instructs the provisioning process to exclude default packages.
- 4 This element instructs the provisioning process to provision the JBoss EAP cloud 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 **cloud-server** base layer defined in the JBoss EAP feature pack and tuned for OpenShift by the JBoss EAP cloud feature pack.
- 6 This element instructs the provisioning process to optimize provisioning of JBoss EAP modules.

6.5. THE MAVEN PLUG-IN CONFIGURATION ATTRIBUTES

You can configure the **eap-maven-plugin** Maven plug-in by setting the following list of configuration parameters.

Table 6.1. The Maven plug-in configuration attributes

Name	Type	Description
channels	List	<p>A list of channel YAML file references. A channel file contains the versions of the JBoss EAP server artifacts. There are two ways to identify a channel YAML file.</p> <ul style="list-style-type: none"> If you deploy the channel YAML file artifact in a Maven repository with the channels classifier, then you can identify it using its Maven coordinates: groupid, artifactId and optional version. If version is not set, it uses the latest channel version. For example: <pre><channels> <channel> <manifest> <groupid>org.jboss.eap.channels</groupid> <artifactId>eap-8.0</artifactId> </manifest> </channel> </channels></pre> <ul style="list-style-type: none"> You can retrieve the channel YAML file by using the URL. For example: <pre><channels> <channel> <manifest> <url>file:///foo/my-manifest.yaml</url> </manifest> </channel> </channels></pre>
excluded-layers	List	<p>A list of Galleon layers to exclude. You can use it when feature-pack-location or feature packs are set. Use the system property wildfly.provisioning.layers.excluded to provide a comma-separated list of layers to exclude.</p>
extra-server-content-dirs	List	<p>A list of directories from which content is copied to the provisioned server. You can use either the absolute path to the directory or the relative path. The relative path must be relative to the project base directory.</p>
feature-packs	List	<p>A list of feature pack configurations to install, which you can combine with layers. Use the system property wildfly.provisioning.feature-packs to provide a comma-separated list of feature packs.</p>

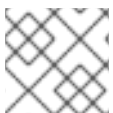
Name	Type	Description
filename	String	The file name of the application to deploy. The default value is <code>\${project.build.finalName}.\${project.packaging}</code> . In an exception case, ejb packaging results in .jar extension. For example, the value of <code>[project.packaging]</code> during war packaging is war and the value of <code>[project.packaging]</code> during ejb packaging is ejb , which is not a valid jar extension. These cases require the .jar extension.
galleon-options	Map	When provisioning the server, you can set specific Galleon options. If you are building a large number of servers in the same Maven session, you must set jboss-fork-embedded option to true to fork Galleon provisioning and CLI scripts execution. For example: <pre><galleon-options> <jboss-fork-embedded>true</jboss-fork-embedded> </galleon-options></pre>
layers	List	A list of Galleon layers to provision. You can use it when feature-pack-location or feature packs are set. Use the system property wildfly.provisioning.layers to provide a comma-separated list of layers.
layers-configuration-file-name	String	A name of the configuration file generated from layers. The default value is standalone.xml . You cannot set this parameter if layers are not configured.
log-provisioning-time	boolean	Specifies whether to log the provisioning time at the end of the provisioning. The default value is false .
name	String	A name used for the deployment.
offline-provisioning	boolean	Specifies whether to use offline mode when the plug-in resolves an artifact. In offline mode, the plug-in uses the local Maven repository for artifact resolution. The default value is false .
overwrite-provisioned-server	boolean	If you want to delete the existing server referenced from the provisioningDir and provision a new one, set it to true . If not, set it to false . The default value is false .

Name	Type	Description
packaging-scripts	List	<p>A list of CLI scripts and commands to execute. If a script file is not absolute, it must be relative to the project base directory. Configure the CLI executions in the following way:</p> <pre> <packaging-scripts> <packaging-script> <scripts> <script>../scripts/script1.cli</script> </scripts> <commands> <command>/system- property=foo:add(value=bar)</command> </commands> <properties-files> <property-file>my- properties.properties</property-file> </properties-files> <java-opts> <java-opt>-Xmx256m</java-opt> </java-opts> <!-- Expressions resolved during server execution --> <resolve-expressions>>false</resolve- expressions> </packaging-script> </packaging-scripts> </pre>
provisioning-dir	String	<p>Path to the directory in which to provision the server. It can be an absolute path or a path relative to the buildDir. By default, the server is provisioned into the target/server directory. The default value is server.</p>
provisioning-file	File	<p>The path to the provisioning.xml file to use. You cannot use it when feature packs configuration item and layers configuration item are set. If the provisioning file path is not absolute, it must be relative to the project base directory. The default value is \${project.basedir}/galleon/provisioning.xml.</p>
record-provisioning-state	boolean	<p>Specifies whether to record the provisioning state in .galleon directory. The default value is false.</p>
runtime-name	String	<p>The runtime-name of the deployment. The default value is the deployment file name, such as myapp.war. You can set this argument to ROOT.war to get the deployment registered in the HTTP root context.</p>

Name	Type	Description
server-config	String	The name of the server configuration to use during deployment. The deployment is deployed inside the configuration referenced from layers-configuration-file-name if layers-configuration-file-name is set. The default value is standalone.xml .
skip	boolean	If you want the goal to be skipped, set it to true . If not, set it to false . The default value is false .
stdout	String	Indicates how stdout and stderr are handled for the created CLI processes. stderr is redirected to stdout if the value is defined unless the value is none. By default the stdout and stderr streams are inherited from the current process. You can change the setting to one from the following options: <ul style="list-style-type: none"> ● None indicates that stderr and stdout should not be used. ● System.out or System.err to redirect to the current processes. ● Any other value is assumed to be the path to a file and the stdout and stderr will be written to it.

6.6. HOW TO ENABLE SUPPORT FOR EAP-DATASOURCES-GALLEON-PACK FOR JBOSS EAP 8.0

The **eap-data-sources-galleon-pack** Galleon feature pack allows you to provision a JBoss EAP 8.0 server that can connect to your databases.



NOTE

Not all databases are supported.

In addition, this feature pack provides JDBC drivers and data sources for various databases that can be provisioned along with the JBoss EAP 8.0 Galleon feature packs. Galleon layers defined in this feature pack are decorator layers. This means that they need to be provisioned in addition to a JBoss EAP base layer.



IMPORTANT

The **datasources-web-server** base layer, is the minimal base layer to use when provisioning Galleon layers that are feature pack defined.

Additional resources

- [JBoss EAP 8.0 defined layer documentation](#)

6.7. SUPPORTED DRIVERS AND DATA SOURCES

For each database the Galleon feature pack supports, it provides Galleon layers that build upon each other, these are:

- **postgresql-driver**
- **postgresql-datasource**
- **mssqlserver-datasource**
- **mssqlserver-driver**
- **oracle-datasource**
- **oracle-driver**

Table 6.2. Supported drivers and data sources

Layers	Description
postgresql-driver	This installs a JBoss EAP module for the driver and adds a driver resource to the data sources subsystem in the server configuration.
postgresql-datasource	This is built upon the postgresql-driver Galleon layer to add a data source.



NOTE

- No specific driver version is included in the feature pack. Before you provision your server, you have to specify the driver version.

Example

```
POSTGRESQL_DRIVER_VERSION="42.2.19"
```

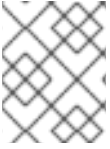
- The driver-specific environment variables are defined within their specific driver documentation.

Additional resources

- [Environment variables to configure Microsoft SQL Server driver and data source](#)
- [Environment variables to configure Oracle driver and data source](#)
- [Environment variables to configure PostgreSQL driver and data source](#)

6.8. USING THE JBOSS EAP MAVEN PLUGIN TO PROVISION A SERVER WITH JDBC DRIVERS AND DATA SOURCES

You can use the Galleon feature pack to provision your JBoss EAP server on OpenShift.



NOTE

This procedure only demonstrates how to provision your JBoss EAP server on OpenShift for JBoss EAP 8.0.

Prerequisites

- You have already installed OpenShift and set up the OpenShift CLI ("oc"). For more information, see [Getting Started with the OpenShift CLI](#).
- You have basic knowledge of how to use the JBoss EAP maven plugin. For more information, see [JBoss EAP Maven plug-in](#).

Procedure

- Add the Maven coordinates (GroupId and artifactId) of the data sources feature pack to the JBoss EAP maven plugin configuration.

```
<channels>
  <channel>
    <groupId>org.jboss.eap.channels</groupId>
    <artifactId>eap-8.0</artifactId>
  </channel>
</channels>
<feature-packs>
  <feature-pack>
    <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap:eap-datasources-galleon-pack</location>
  </feature-pack>
</feature-packs>
<layers>
  <!-- Base layer -->
  <layer>jaxrs-server</layer>
  <!-- The postgresql datasource layer -->
  <layer>postgresql-datasource</layer>
</layers>
```

Additional resources

- [JBoss EAP 8 Beta Maven Plugin configuration for the todo-backend quickstart](#)

CHAPTER 7. CONFIGURING YOUR JBOSS EAP SERVER AND 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 and set environment variables in Helm charts to tune the JVM.



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 environment variables, to make any configuration changes to the JBoss EAP instance inside the JBoss EAP for OpenShift image.

7.1. JVM DEFAULT MEMORY SETTINGS

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 when a valid container memory limit is defined.

Environment variables	Description
JAVA_INITIAL_MEM_RATIO	<p>This environment variable is now deprecated. Corresponds to the JVM argument -XX:InitialRAMPercentage. This is not specified by default and will be removed in a future release. You need to specify --XX:InitialRAMPercentage directly in JAVA_OPTS instead.</p> <div style="display: flex; align-items: flex-start;"> <div style="margin-right: 10px;">  </div> <div> <p>NOTE</p> <p>You no longer need to set JAVA_INITIAL_MEM_RATIO=0 to disable automatic computation. Because no default value is provided for this environment variable.</p> </div> </div>
JAVA_MAX_MEM_RATIO	<p>Environment variable to configure the -XX:MaxRAMPercentage JVM option. Set the maximum heap size as a percentage of the total memory available for the Java VM. The default value is 80%. Setting JAVA_MAX_MEM_RATIO=0 disables this default value.</p>

Environment variables	Description
JAVA_OPTS	<p>Environment variable to provide additional options to the JVM, for example, JAVA_OPTS=-Xms512m -Xmx1024m</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>NOTE</p> <p>If you set a value for -Xms, the -XX:InitialRAMPercentage option is ignored. If you set a value for -Xmx, the -XX:MaxRAMPercentage option is ignored.</p> </div> </div>
JAVA_MAX_INITIAL_MEM	<p>This environment variable is now deprecated. Use JAVA_OPTS to provide the <code>-Xms`</code> option, for example, JAVA_OPTS=-Xms256m</p>

7.2. JVM GARBAGE COLLECTION SETTINGS

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

Garbage Collection Settings

```
-XX:+UseParallelGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -XX:+ExitOnOutOfMemoryError
```

Garbage Collection Logging Settings

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

7.3. JVM ENVIRONMENT VARIABLES

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

Table 7.1. JVM Environment Variables

Variable Name	Example	Default Value	JVM Settings	Description
---------------	---------	---------------	--------------	-------------

Variable Name	Example	Default Value	JVM Settings	Description
JAVA_OPTS	- verbose: class	No default	Multiple	<p>JVM options to pass to the java command.</p> <p>Use JAVA_OPTS_APPEND to configure additional JVM settings. If you use JAVA_OPTS, some unconfigurable defaults are not added to the server JVM settings. You must explicitly add these settings.</p> <p>Using JAVA_OPTS disables certain settings added by default by the container scripts. Disabled settings include:</p> <ul style="list-style-type: none"> • -XX:MetaspaceSize=96M • -Djava.net.preferIPv4Stack=true • - Djboss.modules.system.pkgs=jdk.n ashorn.api,com.sun.crypto.provide r • -Djava.awt.headless=true <p>Add these defaults if you use JAVA_OPTS to configure additional settings.</p>
JAVA_OPTS_APPEND	- Dsome. property =value	No default	Multiple	<p>User-specified Java options to append to generated options in JAVA_OPTS.</p>
JAVA_MAX_MEM_RATIO	50	80	-Xmx	<p>Use this variable when the -Xmx option is not specified in JAVA_OPTS. The value of this variable is used to calculate the 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 -Xmx is set to the specified ratio of the container's available memory. The default value, <i>50</i> 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 <i>0</i>. No -Xmx option will be added to JAVA_OPTS.</p>

Variable Name	Example	Default Value	JVM Settings	Description
JAVA_INITIAL_MEM_RATIO	25	-Xms	-Xms	Use this variable when the -Xms option is not specified in JAVA_OPTS . 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 -Xms is set to the specified ratio of the -Xmx 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 -Xms option will be added to JAVA_OPTS .
JAVA_MAX_INITIAL_MEM	4096	4096	-Xms	JAVA_MAX_INITIAL_MEM environment variable is now deprecated, use JAVA_OPTS to provide -Xms option. For example, JAVA_OPTS=-Xms256m
JAVA_DIAGNOSTICS	true	false (disabled)	-Xlog:gc:utctime -XX:NativeMemoryTracking=summary	Set the value of this variable to true to include diagnostic information in standard output when events occur. If this variable is defined as true in an environment where JAVA_DIAGNOSTICS has already been defined as true , diagnostics are still included.
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.
GC_MIN_HEAP_FREE_RATIO	20	10	-XX:MinHeapFreeRatio	Minimum percentage of heap free after garbage collection to avoid expansion.

Variable Name	Example	Default Value	JVM Settings	Description
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.
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	No default	-XX:MaxMetaspaceSize	The maximum metaspace size.
GC_CONTAINER_OPTIONS	-XX:+UseRGC	-XX:-UseParallelGC	-XX:-UseParallelGC	Specifies the Java garbage collection to use. The value of the variable is specified by using the Java Runtime Environment (JRE) command-line options. The specified JRE command 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**.

7.4. DEFAULT DATASOURCE

The datasource **ExampleDS** is not available in JBoss EAP 8.0.

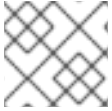
Some quickstarts require this datasource:

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

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

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

ENABLE_GENERATE_DEFAULT_DATASOURCE=true



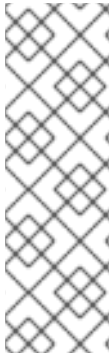
NOTE

This environment variable works only when **cloud-default-config** galleon layer is used.

CHAPTER 8. CAPABILITY TRIMMING IN JBOSS EAP FOR OPENSIFT

Trimming the server can reduce the security exposure of the provisioned server, or reduce the memory footprint so it is more appropriate for a microservice container.

When building an image that includes JBoss EAP, you can control the JBoss EAP features and subsystems to be included in the image. You can do this by using the JBoss EAP Maven plug-in when you create a new application during the Source-to-Image (S2I) build process. For more information, see [Provisioning a JBoss EAP server using the Maven plug-in](#).



NOTE

During the S2I build process, you can use the following environment variables instead of the JBoss EAP Maven plug-in:

- GALLEON_PROVISION_FEATURE_PACKS
- GALLEON_PROVISION_LAYERS
- GALLEON_PROVISION_CHANNELS

8.1. AVAILABLE JBOSS EAP LAYERS

Red Hat provides base and decorator layers that allow you to customize provisioning your JBoss EAP server in OpenShift. The base layers provide core functionality, and the decorator layers 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

8.1.1. 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 an Infinispan-based second-level entity with local caching to 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:

- Native Health
- Native Metrics

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

- Jakarta Security 1.0

cloud-default-config

This layer provisions a server with server configuration based on **standalone-ha.xml** and includes the subsystem configuration **messaging-activemq**. On the contrary, the **modcluster** and **core-management** subsystems configuration are not included. This is configured to be used in the cloud. Additionally, all JBoss EAP server JBoss modules will be installed.

ee-core-profile-server

The **ee-core-profile-server** layer provisions a server with the Jakarta EE 10 Core Profile. The Core Profile provides a small, lightweight profile for users that provides both core JBoss EAP server functionality and Jakarta EE APIs. The **ee-core-profile-server** layer is best suited for smaller runtimes such as cloud-native applications and microservices.

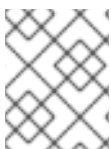
8.1.2. Decorator layers

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

observability

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

- Native Health
- Native Metrics



NOTE

This layer is built into 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.

8.2. 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 [The Galleon provisioning file](#).
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](#).

8.2.1. 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 8.0.

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 8.0 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 a Galleon feature-pack containing layers to provision a MariaDB driver and data source for the JBoss EAP 8.0 server in OpenShift.

8.2.1.1. Preparing the Maven project

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

Procedure

1. Create a new Maven project by running the following command:

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

2. Navigate to **mariadb-galleon-pack** directory and 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 JBoss EAP Galleon feature-pack and the MariaDB driver:

```
<dependencies>
  <dependency>
    <groupId>org.jboss.eap</groupId>
    <artifactId>wildfly-ee-galleon-pack</artifactId>
    <version>8.0.0.GA-redhat-00010</version>
    <type>zip</type>
  </dependency>
  <dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>2.7.2</version>
  </dependency>
</dependencies>
```



NOTE

- **<version>A.B.C-redhat-XXXXX</version>** Where **A.B.C** is the release number and **XXXXX** is build number of your JBoss EAP instance. See the Red Hat Maven repository for version details about JBoss EAP releases. The release and build numbers are available for all JBoss EAP releases. <https://maven.repository.redhat.com/earlyaccess/all/org/jboss/eap/wildfly-ee-galleon-pack/>.

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>6.4.8.Final-redhat-00001</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>
```

```

    </executions>
  </plugin>
</plugins>
</build>

```

8.2.1.2. 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="java.se"/>
    <module name="jakarta.transaction.api"/>
    <module name="jdk.net"/>
  </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"/>
    </feature>
  </feature>
</layer-spec>

```

```

    <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 data source 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 data source 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**.

- 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.

8.2.1.3. 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.jboss.eap.demo:mariadb-galleon-pack:1.0-SNAPSHOT** to an accessible Maven repository.



NOTE

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 **oc** command-line installed
- You are logged in to an OpenShift 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

- Start the **MariaDB** database by running the following command. This example uses the **MariaDB image mariadb-105-rhel7**. You must use the latest supported version of **MariaDB** image. See [Red Hat Ecosystem Catalog](#) to get more information about **MariaDB images**.

```
oc new-app -e MYSQL_USER=admin -e MYSQL_PASSWORD=admin -e
MYSQL_DATABASE=mariadb registry.redhat.io/rhsc/mariadb-105-rhel7
```

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

- Create a secret from the feature-pack 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.

8.2.1.4. Importing the JBoss EAP 8 image stream

You can import the JBoss EAP 8.0 image stream by following the procedure below.

Procedure

1. Import the JBoss EAP 8.0 image stream:

```
oc import-image jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest --
from=registry.redhat.io/jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8:latest
--confirm
```

8.2.1.4.1. Creating an S2I build using the JBoss EAP maven plugin

The **eap-maven-plugin** has been configured with both a reference to the JBoss EAP **galleon feature-pack**, JBoss EAP **cloud galleon feature-pack** and the **mariadb galleon feature-pack**. See an extract of the **pom.xml**:

```
<feature-packs>
  <feature-pack>
    <location>org.jboss.eap:wildfly-ee-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap.cloud:eap-cloud-galleon-pack</location>
  </feature-pack>
  <feature-pack>
    <location>org.jboss.eap.demo:mariadb-galleon-pack:1.0-SNAPSHOT</location> 1
  </feature-pack>
</feature-packs>
<layers>
  <layer>jaxrs-server</layer>
  <layer>mariadb-datasource</layer> 2
</layers>
```

1 The **mariadb feature-pack** version is required. It is not resolved in the JBoss EAP 8 configured channel.

2 The **mariadb-datasource** layer.

Procedure

1. Create the S2I build by running the following command:

```
oc new-build eap8-openjdk17-builder-openshift-rhel8:latest~https://github.com/jboss-
container-images/jboss-eap-8-openshift-image#EAP_8.0.0 \
--context-dir=examples/eap/custom-layers/application \
--build-secret=mariadb-galleon-pack:/tmp/demo-maven-
repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT \ 1
--name=mariadb-app-build
```

1 The **mariadb-galleon-pack** secret is mounted in the **/tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT** directory.

Additional resources

For more information see [the JBoss EAP 8.0 demo example](#).

8.2.1.4.2. Creating an S2I build using the legacy S2I provisioning capabilities

You can use the **openshift-legacy** profile to configure your S2I build so that you can provision your server.

Procedure

1. Create a new OpenShift build by running the following command:

```
oc new-build eap8-openjdk17-builder-openshift-rhel8:latest~https://github.com/jboss-
container-images/jboss-eap-8-openshift-image#EAP_8.0.0 \
--context-dir=examples/eap/custom-layers/application \
--env=GALLEON_PROVISION_CHANNELS="org.jboss.eap.channels:eap-8.0" \ 1
--env=GALLEON_PROVISION_FEATURE_PACKS="org.jboss.eap:wildfly-ee-galleon-
pack,org.jboss.eap.cloud:eap-cloud-galleon-pack,org.jboss.eap.demo:mariadb-galleon-
pack:1.0-SNAPSHOT" \ 2
--env=GALLEON_PROVISION_LAYERS="jaxrs-server,mariadb-datasource" \ 3
--env=GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO="/tmp/demo-maven-
repository" \ 4
--env=MAVEN_ARGS="-Popenshift-legacy" \ 5
--build-secret=mariadb-galleon-pack:/tmp/demo-maven-
repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT \ 6
--name=mariadb-app-build
```

- 1 This environment variable uses the JBoss EAP 8.0 channel during provisioning.
- 2 This environment variable references the JBoss EAP 8.0 **feature-pack**, **cloud feature-pack** and the **mariadb feature-pack**.
- 3 This environment variable references the set of Galleon layers you want to use to provision the server. **jaxrs-server** is a base server layer, **mariadb-datasource** is our custom layer that brings the **mariadb** driver and a new data source to the server installation.
- 4 This points to the location of your local maven repository where the **mariadb feature-pack** is contained.
- 5 This environment variable redefines the **MAVEN_ARGS** to enable the **openshift-legacy** profile.
- 6 The **mariadb-galleon-pack** secret is mounted in the **/tmp/demo-maven-repository/org/jboss/eap/demo/mariadb-galleon-pack/1.0-SNAPSHOT** directory.



NOTE

This directory path complies with Maven repository artifact coordinates to path mapping.

8.2.1.4.3. Starting the build

You can create the **mariadb-app-build** image by creating a new build.

Procedure

1. Start a new build from the same OpenShift build that you created earlier and run the following command:


```
oc start-build mariadb-app-build
```

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

8.2.1.4.4. Creating a new deployment

You can create a new deployment by providing the environment variables that are required to bind the data source to the running **MariaDB** database

Procedure

1. Create a new deployment by running the following command:

```
oc new-app --name=mariadb-app mariadb-app-build \
--env=MARIADB_PORT=3306 \
--env=MARIADB_USER=admin \
--env=MARIADB_PASSWORD=admin \
--env=MARIADB_HOST=mariadb-105-rhel7 \
--env=MARIADB_DATABASE=mariadb \
--env=MARIADB_DATASOURCE=Demo 1
```

- 1** The demo expects the data source to be named **Demo**



NOTE

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

2. Expose the **mariadb-app** application, run the following command:

```
oc expose svc/mariadb-app
```

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

```
curl -X POST http://$(oc get route mariadb-app --template='{{ .spec.host }}')/tasks/title/foo
```

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

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

The added task is displayed in a browser.

8.2.2. 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

- [Custom Galleon feature pack environment variables](#).

8.2.3. 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 8.1. Descriptions of custom Galleon feature pack environment variables

Environment variable	Description
GALLEON_DIR=<path>	Where <i><path></i> is a directory relative to the root directory of your application project. Your <i><path></i> directory contains your optional Galleon custom content, such as the settings.xml file and local Maven repository cache. This cache contains the custom Galleon feature packs. Directory defaults to galleon .
GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO=<path>	<i><path></i> is the absolute path to a Maven local repository directory that contains custom feature packs. Directory defaults to galleon/repository .

Environment variable	Description
GALLEON_PROVISION_FEATURE_PACKS= <list_of_galleon_feature_packs>	<p>Where <list_of_galleon_feature_packs> 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 8.0 server present in the builder image.</p> <p>You can use the GALLEON_PROVISION_LAYERS environment variable to set the Galleon layers, which were defined by your custom feature packs, for your server.</p>

CHAPTER 9. DEPLOYING YOUR JBOSS EAP APPLICATION ON THE OPENSIFT CONTAINER PLATFORM

9.1. JBOSS 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](#).

9.1.1. 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.

9.1.2. 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

1. 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
...
```

2. 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
```

- 1** Name of the operator to subscribe to.
- 2** 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.

3. 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.

9.1.3. Deploying a Java application on OpenShift 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 web console](#) 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.
- 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 8.0.

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-pasword
```



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.xml
```



NOTE

Creating a **ConfigMap** from the **standalone.xml** file is not supported in JBoss EAP 8.0.

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

```
spec:
  disableHTTPRoute: true
```

9.1.3.1. 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'
```

9.1.3.2. 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.

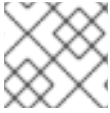
Procedure

- To create a configmap:

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

9.1.3.3. 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

Providing a **standalone.xml** file from the **ConfigMap** is not supported in JBoss EAP 8.0.

Procedure

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

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

9.1.3.4. 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

1. 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.

9.1.4. 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.

9.1.5. Uninstalling EAP operator using web console

You can 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.

9.1.6. Uninstalling JBoss EAP operator using the CLI

You can 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
```

9.1.7. JBoss EAP operator for safe transaction recovery

JBoss EAP operator ensures data consistency before terminating your application cluster. To do this, the operator verifies that all transactions are completed before scaling down the replicas and marking a pod as **clean** for termination.

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](#).

9.1.7.1. 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.

9.1.7.2. 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.

9.1.7.2.1. Pod status during Scaledown

The following table describes the different pod statuses during scaledown:

Table 9.1. 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 clean to be removed from the cluster.

9.1.7.3. 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.

9.1.7.4. 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.xml** configuration file **oc rsh <podname> cat /opt/server/standalone/configuration/standalone.xml**.

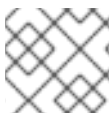
Expected output:

```
<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>
```

9.1.8. 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

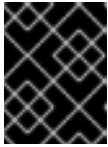
Using HPA ensures that transaction recovery is still handled when a pod is scaled down.

Procedure

1. 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

- [Automatically scaling pods with the horizontal pod autoscaler](#)

9.1.9. Jarkarta enterprise beans remoting on OpenShift**9.1.9.1. Jakarta Enterprise Beans remoting on openShift**

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.
- 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.

9.1.9.1.1. 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 10. TROUBLESHOOTING

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](#).

10.1. TROUBLESHOOTING POD RESTARTS

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
```

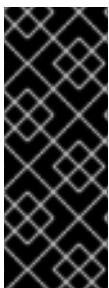
- b. To see the OpenShift events, use the following command.

```
$ 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](#).

10.2. 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 your JBoss EAP server and 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/server/bin/jboss-cli.sh
```

10.3. TROUBLESHOOTING ERRORS WHEN UPDATING HELM CHART FROM VERSION 1.0.0 TO 1.1.0 ON JBOSS EAP 8

There may be errors when upgrading Helm Chart to the latest version on JBoss EAP 8. If you modify the immutable field before upgrading Helm Chart, the following error message may be displayed during the upgrade:

```
UPGRADE FAILED: cannot patch "<helm-release-name>" with kind Deployment: Deployment.apps "<helm-release-name>" is invalid: spec.selector: Invalid value: v1.LabelSelector{MatchLabels:map[string]string{"app.kubernetes.io/instance": "<helm-release-name>", "app.kubernetes.io/name": "<helm-release-name>"}, MatchExpressions: []v1.LabelSelectorRequirement(nil)}: field is immutable
```

To resolve this error, delete the deployment resource by running the command **oc delete deployment <helm-release-name>** before running the command **helm upgrade <helm-release-name>**.

CHAPTER 11. REFERENCE INFORMATION FOR OPENSIFT CONTAINER PLATFORM

The content in this section is derived from the engineering documentation for this application image. The content is provided as a reference for development purposes and for testing beyond the scope of the product documentation.

11.1. INFORMATION ENVIRONMENT VARIABLES

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

Table 11.1. Information Environment Variables

Variable Name	Description and Value
JBOSS_IMAGE_NAME	The image names. Values: <ul style="list-style-type: none"> • jboss-eap-8/eap8-openjdk17-builder-openshift-rhel8 (JDK 17 / RHEL 8)
JBOSS_IMAGE_VERSION	The image version. Value: This is the image version number. See the Red Hat Container Catalog for the latest values: <ul style="list-style-type: none"> • JDK 17 / RHEL 8
JBOSS_MODULES_SYSTEM_PKGS	A comma-separated list of JBoss EAP system modules packages that are available to applications. Value: jdk.nashorn.api
STI_BUILDER	Provides OpenShift S2I support for jee project types. Value: jee

11.2. 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 11.2. Configuration environment variables

Variable Name	Description
CLI_GRACEFUL_SHUTDOWN	<p>If set to any non-zero length value, the image will prevent shutdown with the TERM signal and will require execution of the shutdown command using the JBoss EAP management CLI.</p> <p>Example value: true</p>
CONTAINER_HEAP_PERCENT	<p>Set the maximum Java heap size, as a percentage of available container memory.</p> <p>Example value: 0.5</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: custom,shared</p>
DEFAULT_JMS_CONNECTION_FACTORY	<p>This value is used to specify the default JNDI binding for the Jakarta Messaging connection factory, for example jms-connection-factory='java:jboss/DefaultJMSConnectionFactory'.</p> <p>Example value: java:jboss/DefaultJMSConnectionFactory</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"> • The JBoss EAP 6.4 OpenShift image uses a custom JBoss Web Access Log Valve. • The JBoss EAP for OpenShift image uses the Undertow AccessLogHandler in the JBoss EAP 7.4 <i>Development Guide</i>. <p>Defaults to false.</p>
INITIAL_HEAP_PERCENT	<p>Set the initial Java heap size, as a percentage of the maximum heap size.</p> <p>Example value: 0.5</p>
JAVA_OPTS_APPEND	<p>Server startup options.</p> <p>Example value: -Dfoo=bar</p>
JBOSS_MODULES_SYSTEM_PKGS_APPEND	<p>A comma-separated list of package names that will be appended to the JBOSS_MODULES_SYSTEM_PKGS environment variable.</p> <p>Example value: org.jboss.byteman</p>

Variable Name	Description
JGROUPS_CLUSTER_PASSWORD	<p>Password used to authenticate the node so it is allowed to join the JGroups cluster. Required, when using ASYM_ENCRYPT JGroups cluster traffic encryption protocol. If not set, authentication is disabled, cluster communication is not encrypted and a warning is issued. Optional, when using SYM_ENCRYPT JGroups cluster traffic encryption protocol.</p> <p>Example value: mypassword</p>
JGROUPS_ENCRYPT_KEYSTORE	<p>Name of the keystore file within the created secret specified when using SYM_ENCRYPT JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.</p> <p>Example value: jgroups.jceks</p>
JGROUPS_ENCRYPT_KEYSTORE_DIR	<p>Directory path in which the secret containing the keystore is mounted.</p> <p>Example value: /etc/jgroups-encrypt-secret-volume</p>
JGROUPS_ENCRYPT_NAME	<p>Name associated with the server's certificate, when using SYM_ENCRYPT JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.</p> <p>Example value: jgroups</p>
JGROUPS_ENCRYPT_PASSWORD	<p>Password used to access the keystore and the certificate, when using SYM_ENCRYPT JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.</p> <p>Example value: mypassword</p>
JGROUPS_ENCRYPT_PROTOCOL	<p>JGroups protocol to use for encryption of cluster traffic. Can be either SYM_ENCRYPT or ASYM_ENCRYPT.</p> <p>Defaults to SYM_ENCRYPT.</p> <p>Example value: ASYM_ENCRYPT</p>
JGROUPS_PING_PROTOCOL	<p>JGroups protocol to use for node discovery. Can be either dns.DNS_PING or kubernetes.KUBE_PING.</p>
MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION	<p>For backwards compatibility, set to true to use MyQueue and MyTopic as physical destination name defaults instead of queue/MyQueue and topic/MyTopic.</p>

Variable Name	Description
OPENSIFT_DNS_PING_SERVICE_NAME	Name of the service exposing the ping port on the servers for the DNS discovery mechanism. Example value: eap-app-ping
OPENSIFT_DNS_PING_SERVICE_PORT	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 8888 is used. Defaults to 8888 .
OPENSIFT_KUBE_PING_LABELS	Clustering labels selector for the Kubernetes discovery mechanism. Example value: app=eap-app
OPENSIFT_KUBE_PING_NAMESPACE	Clustering project namespace for the Kubernetes discovery mechanism. Example value: myproject
SCRIPT_DEBUG	If set to true , ensures that the Bash scripts are executed with the -x option, printing the commands and their arguments as they are executed.

11.3. EXPOSED PORTS

Table 11.3. Exposed Ports

Port Number	Description
8443	HTTPS

11.4. 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.

11.4.1. 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](#).

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.

11.4.1.1. 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 connection-url property.</p> <p>Example value: 192.168.1.3</p>
<code>POOLNAME_DATABASETYPE_SERVICE_PORT</code>	<p>Defines the database server's port for the datasource.</p> <p>Example value: 5432</p>
<code>PREFIX_BACKGROUND_VALIDATION</code>	<p>When set to true database connections are validated periodically in a background thread prior to use. Defaults to false, meaning the validate-on-match method is enabled by default instead.</p>
<code>PREFIX_BACKGROUND_VALIDATION_MILLIS</code>	<p>Specifies frequency of the validation, in milliseconds, when the background-validation database connection validation mechanism is enabled (PREFIX_BACKGROUND_VALIDATION variable is set to true). Defaults to 10000.</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: org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker</p>
<code>PREFIX_DATABASE</code>	<p>Defines the database name for the datasource.</p> <p>Example value: myDatabase</p>
<code>PREFIX_DRIVER</code>	<p>Defines Java database driver for the datasource.</p> <p>Example value: postgresql</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: org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter</p>

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

11.4.1.2. Examples

These examples show how value of the **DB_SERVICE_PREFIX_MAPPING** environment variable influences datasource creation.

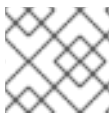
11.4.1.2.1. Single Mapping

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**.

11.4.1.2.2. 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**.

11.5. CLUSTERING

11.5.1. 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.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 Helm chart 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.

11.5.1.1. 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. 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 [Preparing OpenShift to deploy an application](#) for more information on adding policies to service accounts.

11.5.1.2. 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.

11.5.2. 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.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.

11.5.2.1. 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_KEYSTORE_DIR** environment variable must be set to the directory path in which the secret containing the keystore is mounted. For example:

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

3. The **JGROUPS_ENCRYPT_KEYSTORE** environment variable must be set to the name of the keystore file within the created secret specified. If not set, cluster communication is not encrypted and a warning is issued. For example:

```
JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
```

4. 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
```

5. 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
```

11.5.2.2. Configuring ASYM_ENCRYPT

**NOTE**

JBoss EAP 8.0 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_NAME="encrypt_name" \
-e JGROUPS_ENCRYPT_PASSWORD="encrypt_password" \
-e JGROUPS_ENCRYPT_KEYSTORE="encrypt_keystore" \
-e JGROUPS_CLUSTER_PASSWORD="cluster_password"
```

11.5.3. Considerations for scaling up pods

Based on the discovery mechanism in JGroups, a starting node searches for an existing cluster coordinator node. If a coordinator node is not 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 make an assumption of being the first member that leads 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 JBoss EAP Operator uses the **StatefulSet** API, which starts the creation of pods in order, that is, one by one, preventing the creation of split clusters.

11.6. NATIVE HEALTH CHECKS

The JBoss EAP for OpenShift image implements Liveness and readiness probes that are included in OpenShift by default. For more information, see [liveness and readiness probes](#) in the *OpenShift Container Platform Developer 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 11.4. Liveness and Readiness Checks

Performed Test	Liveness	Readiness
Server Status	Any status	Running
Boot Errors	None	None
Deployment Status ^[a]	N/A or no failed entries	N/A or no failed entries

Performed Test	Liveness	Readiness
Native Health Checks	UP	UP
[a] N/A is only a valid state when no deployments are present.		

11.7. MESSAGING

11.7.1. 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.

11.8. 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 11.5. Security Domains

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

11.9. HTTPS ENVIRONMENT VARIABLES

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

11.10. ADMINISTRATION ENVIRONMENT VARIABLES

Table 11.6. Administration Environment Variables

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

11.11. 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.

11.11.1. 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.xml** into the **configuration/** directory.

See [example](#) for such a deployment.

11.11.1.1. 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.

11.11.2. 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**

11.11.3. 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.

11.11.3.1. 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**.

11.11.4. Scripts

run

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

11.11.5. 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

11.11.5.1. 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/server/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.xml
:whoami
quit
```

11.11.5.2. 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
```

11.11.6. 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 11.7. s2i Environment Variables

Variable name	Description
ARTIFACT_DIR	<p>The .war, .ear, and .jar files from this directory will be copied into the deployments/ directory.</p> <p>Example value: target</p>
ENABLE_GENERATE_DEFAULT_DATASOURCE	<p>Optional. When included with the value true, the server is provisioned with the default datasource. Otherwise, the default datasource is not included.</p>
GALLEON_PROVISION_LAYERS	<p>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.</p> <p>Example value: jaxrs</p>
GALLEON_PROVISION_CHANNELS	<p>This is a comma separated list of JBoss EAP channels manifest. The JBoss EAP channel manifest is identified by groupid:artifactId:[version].</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>NOTE</p> <p>The version is optional, which means that the latest channel manifest will be retrieved. For JBoss EAP 8.0, use this channel org.jboss.eap.channels:eap-8.0.</p> </div> </div>
GALLEON_PROVISION_FEATURE_PACKS	<p>Builds environment variable to specify a custom Galleon feature pack for your S2I image. For example: org.jboss.eap:wildfly-ee-galleon-pack:[version],org.jboss.eap.cloud:eap-cloud-galleon-pack:[version].</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>NOTE</p> <p>When you set up the GALLEON_PROVISION_CHANNELS=org.jboss.eap.channels:eap-8.0, the feature-packs versions are not required.</p> </div> </div>
HTTP_PROXY_HOST	<p>Host name or IP address of a HTTP proxy for Maven to use.</p> <p>Example value: 192.168.1.1</p>

Variable name	Description
HTTP_PROXY_PORT	TCP Port of a HTTP proxy for Maven to use. Example value: 8080
HTTP_PROXY_USERNAME	If supplied with HTTP_PROXY_PASSWORD , use credentials for HTTP proxy. Example value: myusername
HTTP_PROXY_PASSWORD	If supplied with HTTP_PROXY_USERNAME , use credentials for HTTP proxy. Example value: mypassword
HTTP_PROXY_NONPROXYHOSTS	If supplied, a configured HTTP proxy will ignore these hosts. Example value: some.example.org *.example.net
MAVEN_ARGS	Overrides the arguments supplied to Maven during build. Example value: -e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package
MAVEN_ARGS_APPEND	Appends user arguments supplied to Maven during build. Example value: -Dfoo=bar
MAVEN_MIRROR_URL	URL of a Maven Mirror/repository manager to configure. Example value: https://10.0.0.1:8443/repository/internal/ Note that the specified URL should be secure. For details see Secure artifact repository mirror URLs
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: true
APP_DATADIR	If defined, directory in the source from where data files are copied. Example value: mydata
DATA_DIR	Directory in the image where data from \$APP_DATADIR will be copied. Example value: EAP_HOME/data

**NOTE**

For more information, see [Building and running JBoss EAP applications on OpenShift Container Platform](#), which uses Maven and the S2I scripts included in the JBoss EAP for OpenShift image.

11.12. 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.

**NOTE**

Transactions propagated over [JBoss Remoting](#) is supported using EAP operator.

11.13. INCLUDED JBOSS MODULES

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

Table 11.8. 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

11.14. EAP OPERATOR: API INFORMATION

The EAP operator introduces the following APIs:

11.14.1. WildFlyServer

WildFlyServer defines a custom JBoss EAP resource.

Table 11.9. WildFlyServer

Field	Description	Scheme	Required
metadata	Standard object's metadata	ObjectMeta v1 meta	false
spec	Specification of the desired behaviour of the JBoss EAP deployment.	WildFlyServerSpec	true
status	Most recent observed status of the JBoss EAP deployment. Read-only.	WildFlyServerStatus	false

11.14.2. WildFlyServerList

WildFlyServerList defines a list of JBoss EAP deployments.

Table 11.10. Table

Field	Description	Scheme	Required
metadata	Standard list's metadata	metav1.ListMeta	false
items	List of WildFlyServer	WildFlyServer	true

11.14.3. 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 11.11. WildFlyServerSpec

Field	Description	Scheme	Required
applicationImage	Name of the application image to be deployed	string	false
replicas	the desired number of replicas for the application	int32]	true
standaloneConfigMap	Spec to specify how a standalone configuration can be read from a ConfigMap .	StandaloneConfigMapSpec	false

Field	Description	Scheme	Required
resources	Resources spec to specify the request or limits of the Stateful Set. If omitted, the namespace defaults are used.	Resources	false
SecurityContext	SecurityContext 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 securityContext	*corev1.SecurityContext	false
storage	Storage spec to specify how storage should be used. If omitted, an EmptyDir is used (that does not persist data across pod restart)	StorageSpec	false
serviceAccountName	Name of the ServiceAccount to use to run the JBoss EAP pods	string	false
envFrom	List of environment variables present in the containers from configMap or secret	corev1.EnvFromSource	false
env	List of environment variable present in the containers	corev1.EnvVar	false
secrets	List of secret names to mount as volumes in the containers. Each secret is mounted as a read-only volume at /etc/secrets/<secret name>	string	false

Field	Description	Scheme	Required
configMaps	List of ConfigMap names to mount as volumes in the containers. Each ConfigMap is mounted as a read-only volume under /etc/configmaps/<config map name>	string	false
disableHTTPRoute	Disable the creation a route to the HTTP port of the application service (false if omitted)	boolean	false
sessionAffinity	If connections from the same client IP are passed to the same JBoss EAP instance/pod each time (false if omitted)	boolean	false

11.14.4. 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](#).

11.14.5. 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 11.12. Table

Field	Description	Scheme	Required
emptyDir	EmptyDirVolumeSource to be used by the JBoss EAP StatefulSet	corev1.EmptyDirVolumeSource	false

Field	Description	Scheme	Required
volumeClaimTemplate	A PersistentVolumeClaim spec to configure Resources requirements to store JBoss EAP standalone data directory. The name of the template is derived from the WildFlyServer name. The corresponding volume is mounted in ReadWriteOnce access mode.	corev1.PersistentVolumeClaim	false

11.14.6. 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 11.13. StandaloneConfigMapSpec

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

11.14.7. WildFlyServerStatus

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

Table 11.14. WildFlyServerStatus

Field	Description	Scheme	Required
replicas	The actual number of replicas for the application	int32	true

Field	Description	Scheme	Required
selector	selector for pods, used by HorizontalPodAutoscaler	string	true
hosts	Hosts that route to the application HTTP service	string	true
pods	Status of the pods	PodStatus	true
scalingdownPods	Number of pods that are under scale down cleaning process	int32	true

11.14.8. PodStatus

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

Table 11.15. PodStatus

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

Revised on 2024-02-21 14:04:12 UTC