



Red Hat JBoss Enterprise Application Platform 7.3

Getting Started with JBoss EAP for OpenShift Online

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

Red Hat JBoss Enterprise Application Platform 7.3 Getting Started with JBoss EAP for OpenShift Online

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

Legal Notice

Copyright © 2020 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 Online

Table of Contents

CHAPTER 1. INTRODUCTION	5
1.1. WHAT IS RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (JBOSS EAP)?	5
1.2. HOW DOES JBOSS EAP WORK ON OPENSIFT?	5
1.3. COMPARISON: JBOSS EAP AND JBOSS EAP FOR OPENSIFT	5
1.4. VERSION COMPATIBILITY AND SUPPORT	6
JDK 8 Images	6
JDK 11 Images	7
1.4.1. OpenShift 4.x Support	7
1.5. DEPLOYMENT OPTIONS	7
CHAPTER 2. BUILD AND RUN A JAVA APPLICATION ON THE JBOSS EAP FOR OPENSIFT IMAGE	8
2.1. PREREQUISITES	8
2.2. PREPARE OPENSIFT FOR APPLICATION DEPLOYMENT	8
2.3. IMPORT THE LATEST JBOSS EAP FOR OPENSIFT IMAGESTREAMS AND TEMPLATES	9
2.4. DEPLOY A JBOSS EAP SOURCE-TO-IMAGE (S2I) APPLICATION TO OPENSIFT	11
2.5. POST DEPLOYMENT TASKS	12
2.6. CHAINED BUILD SUPPORT IN JBOSS EAP FOR OPENSIFT	13
CHAPTER 3. CONFIGURING THE JBOSS EAP FOR OPENSIFT IMAGE FOR YOUR JAVA APPLICATION	14
3.1. HOW THE JBOSS EAP FOR OPENSIFT S2I PROCESS WORKS	14
3.2. CONFIGURING JBOSS EAP FOR OPENSIFT USING ENVIRONMENT VARIABLES	15
3.2.1. JVM Memory Configuration	15
3.2.1.1. JVM Default Memory Settings	15
3.2.1.2. JVM Garbage Collection Settings	16
3.2.1.3. Resource Limits in Default Settings	16
3.2.1.4. JVM Environment Variables	17
3.3. BUILD EXTENSIONS AND PROJECT ARTIFACTS	30
3.3.1. S2I Artifacts	31
3.3.1.1. Modules, Drivers, and Generic Deployments	31
3.3.2. Runtime Artifacts	34
3.3.2.1. Datasources	35
3.3.2.2. Resource Adapters	35
3.4. JBOSS EAP IMAGES FOR OPENSIFT	37
3.5. SSO CONFIGURATION OF RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM FOR OPENSIFT IMAGES	37
3.6. DEFAULT DATASOURCE	37
3.7. DEPLOYMENT CONSIDERATIONS FOR THE JBOSS EAP FOR OPENSIFT IMAGE	38
3.7.1. Scaling Up and Persistent Storage Partitioning	38
3.7.2. Scaling Down and Transaction Recovery	38
CHAPTER 4. CAPABILITY TRIMMING IN JBOSS EAP FOR OPENSIFT	39
4.1. PROVISION A CUSTOM JBOSS EAP SERVER	39
4.2. AVAILABLE JBOSS EAP LAYERS	39
4.2.1. Base Layers	39
datasources-web-server	39
jaxrs-server	40
cloud-server	40
4.2.2. Decorator Layers	41
sso	41
observability	41
web-clustering	41
4.3. PROVISIONING USER-DEVELOPED LAYERS IN JBOSS EAP	41

4.3.1. Building Custom Layers for JBoss EAP	42
4.3.2. Custom Provisioning Files for JBoss EAP	43
4.3.3. Building an Application Provisioned with User-developed Layers	44
CHAPTER 5. MIGRATION OF APPLICATIONS FROM JBOSS EAP IMAGESTREAMS ON OPENSIFT 3 TO EAP73 IMAGESTREAMS	46
5.1. UPDATES TO LIVENESS AND READINESS PROBE CONFIGURATION FOR EAP73 IMAGESTREAMS	46
5.2. DEFAULT DATASOURCE REMOVED	47
CHAPTER 6. TROUBLESHOOTING	48
6.1. TROUBLESHOOTING POD RESTARTS	48
6.2. TROUBLESHOOTING USING THE JBOSS EAP MANAGEMENT CLI	48
CHAPTER 7. EAP OPERATOR FOR AUTOMATING APPLICATION DEPLOYMENT ON OPENSIFT	50
7.1. INSTALLING EAP OPERATOR USING THE WEB CONSOLE	50
7.2. INSTALLING EAP OPERATOR USING THE CLI	51
7.3. DEPLOYING A JAVA APPLICATION ON OPENSIFT USING THE EAP OPERATOR	53
7.3.1. Creating a Secret	56
7.3.2. Creating a ConfigMap	56
7.3.3. Creating a ConfigMap from a standalone.xml File	56
7.3.4. Configuring Persistent Storage for Applications	57
7.4. UNINSTALLING EAP OPERATOR USING WEB CONSOLE	57
7.5. UNINSTALLING EAP OPERATOR USING THE CLI	58
7.6. EAP OPERATOR FOR SAFE TRANSACTION RECOVERY	59
7.6.1. StatefulSets for Stable Network Host Names	60
7.6.2. Monitoring the Scaledown Process	60
7.6.2.1. Pod Status During Scaledown	60
7.6.3. Scaling Down During Transactions with Heuristic Outcomes	61
7.7. EJB REMOTING ON OPENSIFT	61
7.7.1. Configuring EJB on OpenShift	62
CHAPTER 8. REFERENCE INFORMATION	64
8.1. PERSISTENT TEMPLATES	64
8.2. INFORMATION ENVIRONMENT VARIABLES	64
8.3. CONFIGURATION ENVIRONMENT VARIABLES	65
8.4. APPLICATION TEMPLATES	70
8.5. EXPOSED PORTS	71
8.6. DATASOURCES	71
8.6.1. JNDI Mappings for Datasources	71
8.6.1.1. Database Drivers	72
8.6.1.2. Datasource Configuration Environment Variables	72
8.6.1.3. Examples	74
8.6.1.3.1. Single Mapping	75
8.6.1.3.2. Multiple Mappings	75
8.7. CLUSTERING	75
8.7.1. Configuring a JGroups Discovery Mechanism	75
8.7.1.1. Configuring KUBE_PING	76
8.7.1.2. Configuring DNS_PING	76
8.7.2. Configuring JGroups to Encrypt Cluster Traffic	78
8.7.2.1. Configuring SYM_ENCRYPT	78
8.7.2.2. Configuring ASYM_ENCRYPT	79
8.8. HEALTH CHECKS	79
8.9. MESSAGING	80
8.9.1. Configuring External Red Hat AMQ Brokers	80

Example OpenShift Application Definition	80
8.10. SECURITY DOMAINS	80
8.11. HTTPS ENVIRONMENT VARIABLES	81
8.12. ADMINISTRATION ENVIRONMENT VARIABLES	81
8.13. S2I	82
8.13.1. Custom Configuration	82
8.13.1.1. Custom Modules	82
8.13.2. Deployment Artifacts	82
8.13.3. Artifact Repository Mirrors	82
8.13.3.1. Secure Artifact Repository Mirror URLs	83
8.13.4. Scripts	83
8.13.5. Custom Scripts	83
8.13.5.1. Mounting a configmap to execute custom scripts	84
8.13.5.2. Using install.sh to execute custom scripts	84
8.13.6. Environment Variables	85
8.14. SSO	87
8.15. TRANSACTION RECOVERY	88
8.15.1. Unsupported Transaction Recovery Scenarios	88
8.15.2. Manual Transaction Recovery Process	88
8.15.2.1. Caveats	88
8.15.2.2. Prerequisite	89
8.15.2.3. Procedure	90
8.15.2.3.1. Resolving In-doubt Branches	90
8.15.2.3.2. Extract the Global Transaction ID and Node Identifier from Each XID	91
8.15.2.3.3. Obtain the List of Node Identifiers of All Running JBoss EAP Instances in Any Cluster that Can Contact the Resource Managers	93
8.15.2.3.4. Find the Transaction Logs	93
8.15.2.3.5. Cleaning Up the Transaction Logs for Reconciled In-doubt Branches	94
8.16. INCLUDED JBOSS MODULES	95
8.17. EAP OPERATOR: API INFORMATION	95
8.17.1. WildFlyServer	96
8.17.2. WildFlyServerList	96
8.17.3. WildFlyServerSpec	96
8.17.4. StorageSpec	98
8.17.5. StandaloneConfigMapSpec	98
8.17.6. WildFlyServerStatus	99
8.17.7. PodStatus	99

CHAPTER 1. INTRODUCTION

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

Red Hat JBoss Enterprise Application Platform 7 (JBoss EAP) is a middleware platform built on open standards and compliant with the Java Enterprise Edition 7 specification. It provides preconfigured options for features such as high-availability clustering, messaging, and distributed caching. It includes a modular structure that allows you to enable services only when required, which results in improved startup speed.

The web-based management console and management command line interface (CLI) make editing XML configuration files unnecessary and add the ability to script and automate tasks. In addition, JBoss EAP includes APIs and development frameworks that allow you to quickly develop, deploy, and run secure and scalable Jakarta EE applications. JBoss EAP 7 is a Jakarta EE 8 compatible implementation for both Web Profile and Full Platform specifications and also a certified implementation of the Java EE 8 Full Platform and Web Profile specifications.

1.2. HOW DOES JBOSS EAP WORK ON OPENSHIFT?

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

1.3. COMPARISON: JBOSS EAP AND JBOSS EAP FOR OPENSHIFT

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

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

JBoss EAP Feature	Status in JBoss EAP for OpenShift	Description
JBoss EAP management console	Not included	The JBoss EAP management console is not included in this release of JBoss EAP for OpenShift.
JBoss EAP management CLI	Not recommended	The JBoss EAP management CLI is not recommended for use with JBoss EAP running in a containerized environment. Any configuration changes made using the management CLI in a running container will be lost when the container restarts. 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.

JBoss EAP Feature	Status in JBoss EAP for OpenShift	Description
Default root page	Disabled	The default root page is disabled, but you can deploy your own application to the root context as ROOT.war .
Remote messaging	Supported	Red Hat AMQ for inter-pod and remote messaging is supported. ActiveMQ Artemis is only supported for messaging within a single pod with JBoss EAP instances, and is only enabled when Red Hat AMQ is absent.
Transaction recovery	Partially supported	There are some unsupported transaction recovery scenarios and caveats when undertaking transaction recovery with the JBoss EAP for OpenShift image. 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 .

1.4. VERSION COMPATIBILITY AND SUPPORT

JBoss EAP for OpenShift provides images for both JDK 8 and JDK 11.

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

The following modifications have been applied to images in JBoss EAP for OpenShift 7.3.

- Default drivers and modules have been removed.
- Templates for MySQL, PostgreSQL, and MongoDB have been removed. You can provision these capabilities with a custom layer.
- The Hawkular agent is not active in these images. If configured, it is ignored.

JDK 8 Images

- Red Hat Universal Base Image: 7
- Prefix for template names: eap73-*
- Container Catalog: <https://access.redhat.com/containers/?tab=overview#/registry.access.redhat.com/jboss-eap-7/eap73-openjdk8-openshift-rhel7>
- Builder Image: **jboss-eap-7/jboss-eap73-openshift**

- Runtime Image: **jboss-eap-7/jboss-eap73-runtime-openshift**

JDK 11 Images

- Red Hat Universal Base Image: 8
- Prefix for template names: eap73-openjdk11-*
- Container Catalog: <https://access.redhat.com/containers/?tab=overview#/registry.access.redhat.com/jboss-eap-7/eap73-openjdk11-openshift-rhel8>
- Builder Image: **jboss-eap-7/jboss-eap73-openjdk11-openshift**
- Runtime Image: **jboss-eap-7/jboss-eap73-openjdk11-runtime-openshift**

JBoss EAP for OpenShift is updated frequently. Therefore, it is important to understand which versions of the images are compatible with which versions of OpenShift. See [OpenShift and Atomic Platform Tested Integrations](#) on the Red Hat Customer Portal for more information on version compatibility and support.

Additional Resources

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

1.4.1. OpenShift 4.x Support

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

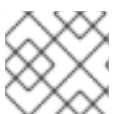
In previous releases of OpenShift, certain kube-apiserver proxied requests were authenticated and passed through to the cluster. This behavior is now considered insecure, and so, accessing Jolokia in this manner is no longer supported.

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

1.5. DEPLOYMENT OPTIONS

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

- A JBoss EAP for OpenShift [template](#). For more information, see [Build and Run a Java Application on the JBoss EAP CD for OpenShift Image](#).
- 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. For more information, see [EAP Operator for Automating Application Deployment on OpenShift](#).



NOTE

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

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

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

As an example, the **kitchensink** quickstart is used in this procedure. It demonstrates a Jakarta EE web-enabled database application using JSF, CDI, EJB, JPA, and Bean Validation. See the **kitchensink** quickstart that ships with JBoss EAP 7 for more information.

2.1. PREREQUISITES

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

2.2. PREPARE OPENSIFT FOR APPLICATION DEPLOYMENT

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

```
$ oc new-project PROJECT_NAME
```

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

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

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



NOTE

Creating a keystore and a secret is required if you are using any HTTPS-enabled features in your OpenShift project. For example, if you are using the **eap73-https-s2i** template (for JDK 8) or the **eap73-openjdk11-https-s2i** template (for JDK 11), you must create a keystore and secret.

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

- a. Create a keystore.

**WARNING**

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

You can use the Java **keytool** command to generate a keystore using the following command.

```
$ keytool -genkey -keyalg RSA -alias ALIAS_NAME -keystore
KEYSTORE_FILENAME.jks -validity 360 -keysize 2048
```

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

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

- b. Create a secret from the keystore.

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

```
$ oc secrets new SECRET_NAME KEYSTORE_FILENAME.jks
```

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

```
$ oc secrets new eap7-app-secret keystore.jks
```

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

Select one of the following commands to import the latest JBoss EAP for OpenShift JDK 8 and JDK 11 imagestreams and templates into your OpenShift project's namespace.

Use the following command for JDK 8 image stream:

```
for resource in \
  eap73-amq-persistent-s2i.json \
  eap73-amq-s2i.json \
  eap73-basic-s2i.json \
  eap73-https-s2i.json \
  eap73-image-stream.json \
  eap73-sso-s2i.json \
  eap73-starter-s2i.json \
do
  oc replace --force -f \
```

```
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-7-openshift-
image/eap73/templates/${resource}
done
```

This command imports the following images and templates.

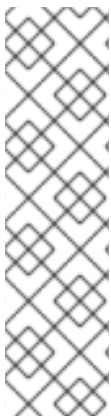
- The JDK 8 builder image: `jboss-eap73-openshift`
- The JDK 8 runtime image: `jboss-eap73-runtime-openshift`
- All templates specified in the command.

Use the following command for JDK 11 image stream:

```
for resource in \
  eap73-openjdk11-amq-persistent-s2i.json \
  eap73-openjdk11-amq-s2i.json \
  eap73-openjdk11-basic-s2i.json \
  eap73-openjdk11-https-s2i.json \
  eap73-openjdk11-image-stream.json \
  eap73-openjdk11-sso-s2i.json \
  eap73-openjdk11-starter-s2i.json \
do
  oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-7-openshift-
image/eap73/templates/${resource}
done
```

This command imports the following images and templates.

- The JDK 11 builder image: `jboss-eap73-openjdk11-openshift`
- The JDK 11 runtime image: `jboss-eap73-openjdk11-runtime-openshift`
- All templates specified in the command.



NOTE

The JBoss EAP imagestreams and templates imported using the above command are only available within that OpenShift project.

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

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

If you use the `cluster-samples-operator`, refer to the OpenShift documentation on configuring the cluster samples operator. See https://docs.openshift.com/container-platform/latest/openshift_images/configuring-samples-operator.html for details about configuring the cluster samples operator.

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

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

JDK 8 template names use the prefix **eap73-***; for example, **eap73-https-s2i**. JDK 11 template names use the prefix **eap73-openjdk11-***; for example, **eap73-openjdk11-https-s2i**.

Prerequisites

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

Procedure

1. Create a new OpenShift application that uses the JBoss EAP for OpenShift image and the source code of your Java application. You can use one of the provided JBoss EAP for OpenShift templates for S2I builds. You can also choose to provision a trimmed server. For example, to deploy the **kitchensink** quickstart using the JDK 8 builder image, enter the following command to use the **eap73-basic-s2i** template in the **eap-demo** project, created in [Prepare OpenShift for Application Deployment](#), with the **kitchensink** source code on GitHub. This quickstart does not support the trimming capability.

```
oc new-app --template=eap73-basic-s2i \ 1
-p IMAGE_STREAM_NAMESPACE=eap-demo \ 2
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \ 3
-p SOURCE_REPOSITORY_REF=openshift \ 4
-p CONTEXT_DIR=kitchensink 5
```

- 1 The template to use. The **eap73** prefix specifies the JDK 8 template.
- 2 The latest imagestreams and templates [were imported into the project's namespace](#), so you must specify the namespace where to find the imagestream. This is usually the project's name
- 3 URL to the repository containing the application source code.
- 4 The Git repository reference to use for the source code. This can be a Git branch or tag reference.
- 5 The directory within the source repository to build.

As another example, to deploy the **helloworld-html5** quickstart using the JDK 11 runtime image and trimming JBoss EAP to include only the **jaxrs-server** layer, enter the following command. The command uses the **eap73-openjdk11-basic-s2i** template in the **eap-demo** project, created in [Prepare OpenShift for Application Deployment](#), with the **helloworld-html5** source code on GitHub.

```
oc new-app --template=eap73-openjdk11-basic-s2i \ 1
-p IMAGE_STREAM_NAMESPACE=eap-demo \ 2
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \ 3
```

```
-p SOURCE_REPOSITORY_REF=openshift \ 4
-p GALLEON_PROVISION_LAYERS=jaxrs-server \ 5
-p CONTEXT_DIR=helloworld-html5 6
```

- 1 The template to use. The **eap73-openjdk11** prefix specifies the JDK 11 template.
- 2 The latest imagestreams and templates [were imported into the project's namespace](#), so you must specify the namespace where to find the imagestream. This is usually the project's name.
- 3 URL to the repository containing the application source code.
- 4 The Git repository reference to use for the source code. This can be a Git branch or tag reference.
- 5 Provision a trimmed server with only the **jaxrs-server** layer.
- 6 The directory within the source repository to build.



NOTE

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

For example, if you are using an HTTPS template such as **eap73-https-s2i**, you must specify the required [HTTPS environment variables](#) **HTTPS_NAME**, **HTTPS_PASSWORD**, and **HTTPS_KEYSTORE** to match your keystore details.

2. Retrieve the name of the build configuration.

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

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

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

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

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

Additional Resources

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

2.5. POST DEPLOYMENT TASKS

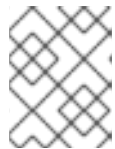
Depending on your application, some tasks might need to be performed after your OpenShift application has been built and deployed. This might include exposing a service so that the application is viewable from outside of OpenShift, or scaling your application to a specific number of replicas.

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


```
$ oc get service
```

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

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



NOTE

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

3. Get the URL of the route.

```
$ oc get route
```

4. Access the application in your web browser using the URL. The URL is the value of the **HOST/PORT** field from previous command's output.
If your application does not use the JBoss EAP root context, append the context of the application to the URL. For example, for the **kitchensink** quickstart, the URL might be **http://HOST_PORT_VALUE/kitchensink/**.
5. Optionally, you can also scale up the application instance by running the following command. This increases the number of replicas to **3**.

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

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

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

2.6. CHAINED BUILD SUPPORT IN JBOSS EAP FOR OPENSIFT

JBoss EAP for OpenShift supports chained builds in OpenShift.

For details, see the OpenShift documentation.

Additional Resources

[OpenShift Chained build documentation](#)

CHAPTER 3. CONFIGURING THE JBOSS EAP FOR OPENSIFT IMAGE FOR YOUR JAVA APPLICATION

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



IMPORTANT

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

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

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

3.1. HOW THE JBOSS EAP FOR OPENSIFT S2I PROCESS WORKS



NOTE

The variable **EAP_HOME** is used to denote the path to the JBoss EAP installation inside the JBoss EAP for OpenShift image.

The S2I process for JBoss EAP for OpenShift works as follows:

1. If a **pom.xml** file is present in the source code repository, a Maven build process is triggered that uses the contents of the **\$MAVEN_ARGS** environment variable. Although you can specify custom Maven arguments or options with the **\$MAVEN_ARGS** environment variable, Red Hat recommends that you use the **\$MAVEN_ARGS_APPEND** environment variable to do this. The **\$MAVEN_ARGS_APPEND** variable takes the default arguments from **\$MAVEN_ARGS** and appends the options from **\$MAVEN_ARGS_APPEND** to it.

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



NOTE

To use Maven behind a proxy on JBoss EAP for OpenShift image, set the **\$HTTP_PROXY_HOST** and **\$HTTP_PROXY_PORT** environment variables. Optionally, you can also set the **\$HTTP_PROXY_USERNAME**, **HTTP_PROXY_PASSWORD**, and **HTTP_PROXY_NONPROXYHOSTS** variables.

2. The results of a successful Maven build are copied to the **EAP_HOME/standalone/deployments/** directory inside the JBoss EAP for OpenShift image. This includes all JAR, WAR, and EAR files from the source repository specified by the

\$ARTIFACT_DIR environment variable. The default value of **\$ARTIFACT_DIR** is the Maven target directory.

3. All files in the **configuration** source repository directory are copied to the **EAP_HOME/standalone/configuration/** directory inside the JBoss EAP for OpenShift image. If you want to use a custom JBoss EAP configuration file, it should be named **standalone-openshift.xml**.
4. All files in the **modules** source repository directory are copied to the **EAP_HOME/modules/** directory inside the JBoss EAP for OpenShift image.

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

3.2. CONFIGURING JBOSS EAP FOR OPENSIFT USING ENVIRONMENT VARIABLES

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

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

```
oc new-app --template=eap73-basic-s2i \
-p IMAGE_STREAM_NAMESPACE=eap-demo \
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \
-p SOURCE_REPOSITORY_REF=openshift \
-p CONTEXT_DIR=kitchensink \
-e ADMIN_USERNAME=myspecialuser \
-e ADMIN_PASSWORD=myspecialp@ssw0rd
```



NOTE

This example uses the JDK 8 template. For JDK 11, use the **eap73-openjdk11-basic-s2i** template.

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

3.2.1. JVM Memory Configuration

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

3.2.1.1. JVM Default Memory Settings

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

The container memory limit is retrieved from the file `/sys/fs/cgroup/memory/memory.limit_in_bytes`. The total available memory is retrieved using the `/proc/meminfo` command.

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

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

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

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

- **JAVA_MAX_MEM_RATIO**
- **JAVA_INITIAL_MEM_RATIO**
- **JAVA_MAX_INITIAL_MEM**

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

- **JAVA_INITIAL_MEM_RATIO**
- **JAVA_MAX_MEM_RATIO**

3.2.1.2. JVM Garbage Collection Settings

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

Garbage Collection Settings

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

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

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

Garbage Collection Logging Settings for Java 11 (modular JVM)

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

3.2.1.3. Resource Limits in Default Settings

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

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

The value of `{core-limit}` is defined using the **JAVA_CORE_LIMIT** environment variable, or by the CPU core limit imposed by the container.

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

3.2.1.4. JVM Environment Variables

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

Table 3.1. JVM Environment Variables


Variable Name	Example	Default Value	JVM Settings	Description
JAVA_OPTS	-verbose:class	No default	Multiple	JVM options to pass to the java command.


**N
O
T
E**


Use **JAVA_OPTS** **APPEND** to configure additional JVM settings. If you use **JAVA**

Variable Name	Example	Default Value	JVM Settings	Description	V A	
					<p>OPTS, some unconfigurable defaults are not added to the server JVM settings. You must explicitly add these settings.</p>	
					<p>IMP O</p>	


Variable Name	Example	Default Value	JVM Settings	Description
				<p>R T A N T</p> <p>Us in g J A V A</p> <p>– O P T S</p> <p>di sa bl es ce rt ai n se tti ng s ad de d by de fa ult by th e co nt ai ne r sc rip ts. Di sa bl ed se tti ng s in cl ud e</p>

Variable Name	Example	Default Value	JVM Settings	Description
				

Variable Name	Example	Default Value	JVM Settings	Description
				

Variable Name	Example	Default Value	JVM Settings	Description
				 <p data-bbox="1394 1151 1433 2130">In addition, if automatic memory calculation is not enabled, the initial</p>

Variable Name	Example	Default Value	JVM Settings	Description
				<p>Java memory (-Xms) and maximum Java memory (-Xmx) are not defined.</p> <p>Add these defaults if you use JAVA_OPTS to configure</p>

Variable Name	Example	Default Value	JVM Settings	Description
				 additional settings.
JAVA_OPTS_APP END	- Dsome.property=v alue	No default	Multiple	User-specified Java options to append to generated options in JAVA_OPTS .

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

Variable Name	Example	Default Value	JVM Settings	Description
JAVA_INITIAL_MEMORY_RATIO	25	25	-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 .

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

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

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

Variable Name	Example	Default Value	JVM Settings	Description
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	90	90	-XX:AdaptiveSizePolicyWeight	The weighting given to the current garbage collection time versus the previous garbage collection times.
GC_METASPACE_SIZE	20	96	-XX:MetaspaceSize	The initial metaspace size.
GC_MAX_METASPACE_SIZE	100	256	-XX:MaxMetaspaceSize	The maximum metaspace size.
GC_CONTAINER_OPTIONS	-XX:+UserG1GC	-XX:-UseParallelOldGC	-XX:-UseParallelOldGC	Specifies the Java garbage collection to use. The value of the variable should be the JRE command-line options to specify the required garbage collection. The JRE command specified overrides the default.

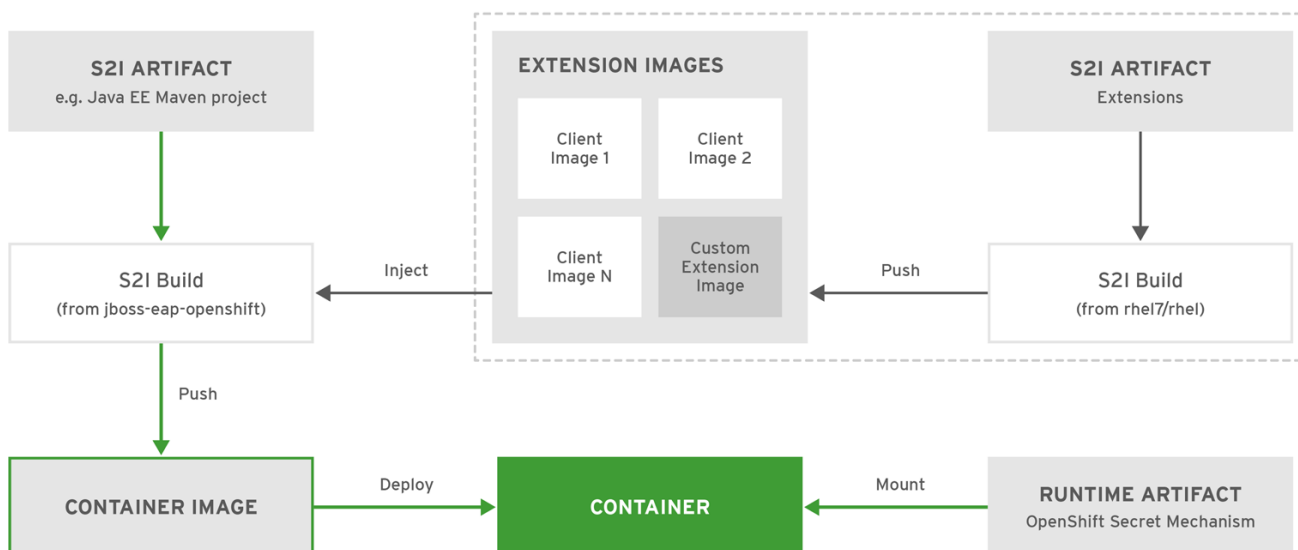
The following environment variables are deprecated:

- **JAVA_OPTIONS**: Use **JAVA_OPTS**.
- **INITIAL_HEAP_PERCENT**: Use **JAVA_INITIAL_MEM_RATIO**.
- **CONTAINER_HEAP_PERCENT**: Use **JAVA_MAX_MEM_RATIO**.

3.3. BUILD EXTENSIONS AND PROJECT ARTIFACTS

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

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



JBOSS_409952_0617

IMPORTANT

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

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

- MySQL
- PostgreSQL

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

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

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

Additional Resources

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

3.3.1. S2I Artifacts

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

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

3.3.1.1. Modules, Drivers, and Generic Deployments

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

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

install.sh Script Example

```
#!/bin/bash

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

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

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

- **install_modules**
- **configure_drivers**
- **install_deployments**

Modules

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

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

Example module.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="org.apache.derby">
  <resources>
    <resource-root path="derby-10.12.1.1.jar"/>
    <resource-root path="derbyclient-10.12.1.1.jar"/>
  </resources>
```

```

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

```

Example module.xml File for PostgreSQL Datasource

```

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

```

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

```

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

```



NOTE

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

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

Drivers

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

Example drivers.env File

```

#DRIVER
DRIVERS=DERBY
DERBY_DRIVER_NAME=derby
DERBY_DRIVER_MODULE=org.apache.derby

```

```
DERBY_DRIVER_CLASS=org.apache.derby.jdbc.EmbeddedDriver
DERBY_XA_DATASOURCE_CLASS=org.apache.derby.jdbc.EmbeddedXADataSource
```

Adding Datasource Drivers

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

The mechanism follows the **Derby** driver example and uses S2I artifacts. Create a **drivers.env** file for each datasource to be installed.

Example **drivers.env** File for MySQL Datasource

```
#DRIVER
DRIVERS=MYSQL
MYSQL_DRIVER_NAME=mysql
MYSQL_DRIVER_MODULE=org.mysql
MYSQL_DRIVER_CLASS=com.mysql.cj.jdbc.Driver
MYSQL_XA_DATASOURCE_CLASS=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
```

Example **drivers.env** File for PostgreSQL Datasource

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

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

Generic Deployments

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

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

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

3.3.2. Runtime Artifacts

3.3.2.1. Datasources

There are two types of datasources:

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

Example: Datasource Environment File

```
# derby datasource
ACCOUNTS_DERBY_DATABASE=accounts
ACCOUNTS_DERBY_JNDI=java:/accounts-ds
ACCOUNTS_DERBY_DRIVER=derby
ACCOUNTS_DERBY_USERNAME=derby
ACCOUNTS_DERBY_PASSWORD=derby
ACCOUNTS_DERBY_TX_ISOLATION=TRANSACTION_READ_UNCOMMITTED
ACCOUNTS_DERBY_JTA=true

# Connection info for xa datasource
ACCOUNTS_DERBY_XA_CONNECTION_PROPERTY_DataSourceName=/home/jboss/source/data/databases/derby/accounts

# _HOST and _PORT are required, but not used
ACCOUNTS_DERBY_SERVICE_HOST=dummy
ACCOUNTS_DERBY_SERVICE_PORT=1527
```

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

Datasources contain two types of properties: connection pool-specific properties and database driver-specific properties. Database driver-specific properties use the generic **XA_CONNECTION_PROPERTY**, because the driver itself is configured as a driver S2I artifact. The suffix of the driver property is specific to the particular driver for the datasource.

In the above example, **ACCOUNTS** is the datasource prefix, **XA_CONNECTION_PROPERTY** is the generic driver property, and **DataSourceName** is the property specific to the driver.

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

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

3.3.2.2. Resource Adapters

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

Table 3.2. Resource Adapter Properties

Attribute	Description
<i>PREFIX_ID</i>	The identifier of the resource adapter as specified in the server configuration file.
<i>PREFIX_ARCHIVE</i>	The resource adapter archive.
<i>PREFIX_MODULE_SLOT</i>	The slot subdirectory, which contains the module.xml configuration file and any required JAR files.
<i>PREFIX_MODULE_ID</i>	The JBoss Module ID where the object factory Java class can be loaded from.
<i>PREFIX_CONNECTION_CLASS</i>	The fully qualified class name of a managed connection factory or admin object.
<i>PREFIX_CONNECTION_JNDI</i>	The JNDI name for the connection factory.
<i>PREFIX_PROPERTY_ParentDirectory</i>	Directory where the data files are stored.
<i>PREFIX_PROPERTY_AllowParentPaths</i>	Set AllowParentPaths to false to disallow .. in paths. This prevents requesting files that are not contained in the parent directory.
<i>PREFIX_POOL_MAX_SIZE</i>	The maximum number of connections for a pool. No more connections will be created in each sub-pool.
<i>PREFIX_POOL_MIN_SIZE</i>	The minimum number of connections for a pool.
<i>PREFIX_POOL_PREFILL</i>	Specifies if the pool should be prefilled. Changing this value requires a server restart.
<i>PREFIX_POOL_FLUSH_STRATEGY</i>	How the pool should be flushed in case of an error. Valid values are: FailingConnectionOnly (default), IdleConnections , and EntirePool .

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

Example: Resource Adapter Environment File

```
#RESOURCE_ADAPTER
RESOURCE_ADAPTERS=MYRA
MYRA_ID=myra
MYRA_ARCHIVE=myra.rar
```



```

MYRA_CONNECTION_CLASS=org.javaee7.jca.connector.simple.connector.outbound.MyManagedCo
nnectionFactory
MYRA_CONNECTION_JNDI=java:/eis/MySimpleMFC

```

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

```

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

```

3.4. JBOSS EAP IMAGES FOR OPENSIFT

When you import JBoss EAP images for OpenShift, two images are generated:

- An **[application name]-build-artifacts** image that is created from the JBoss EAP S2I builder image.
- An image created from the JBoss EAP runtime image in which the server and your application are installed.

In most cases, you only need the runtime image. You can delete the **[application name]-build-artifacts** image.

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

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

The environment variable **SSO_FORCE_LEGACY_SECURITY** is set to **true** in these images.

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

3.6. DEFAULT DATASOURCE

In JBoss EAP 7.3, the default datasource is removed

Some quickstarts require this datasource:

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

Applications developed by customers might also require the default datasource.

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

```

ENABLE_GENERATE_DEFAULT_DATASOURCE=true

```

3.7. DEPLOYMENT CONSIDERATIONS FOR THE JBOSS EAP FOR OPENSIFT IMAGE

3.7.1. Scaling Up and Persistent Storage Partitioning

There are two methods for deploying JBoss EAP with persistent storage: single-node partitioning, and multi-node partitioning.

Single-node partitioning stores the JBoss EAP data store directory, including transaction data, in the storage volume.

Multi-node partitioning creates additional, independent **split-*n*** directories to store the transaction data for each JBoss EAP pod, where ***n*** is an incremental integer. This communication is not altered if a JBoss EAP pod is updated, goes down unexpectedly, or is redeployed. When the JBoss EAP pod is operational again, it reconnects to the associated split directory and continues as before. If a new JBoss EAP pod is added, a corresponding **split-*n*** directory is created for that pod.

To enable the multi-node configuration you must set the **SPLIT_DATA** parameter to **true**. This results in the server creating independent **split-*n*** directories for each instance within the persistent volume which are used as their data store.



WARNING

Using the environment variables like **SPLIT_DATA** while using the EAP operator can cause consistency issues. You must use the EAP operator to manage transaction discovery in OpenShift 4 and later versions.



IMPORTANT

Due to the different storage methods of single-node and multi-node partitioning, changing a deployment from single-node to multi-node results in the application losing all data previously stored in the data directory, including messages, transaction logs, and so on. This is also true if changing a deployment from multi-node to single-node, as the storage paths will not match.

3.7.2. Scaling Down and Transaction Recovery

When the JBoss EAP for OpenShift image is deployed using a **multi-node** configuration, it is possible for unexpectedly terminated transactions to be left in the data directory of a terminating pod if the cluster is scaled down.

See [manual transaction recovery](#) to complete these branches.

CHAPTER 4. CAPABILITY TRIMMING IN JBOSS EAP FOR OPENSIFT

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

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

4.1. PROVISION A CUSTOM JBOSS EAP SERVER

To provision a custom server with trimmed capabilities, pass the **GALLEON_PROVISION_LAYERS** environment variable during the S2I build phase.

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

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

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

4.2. AVAILABLE JBOSS EAP LAYERS

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

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

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

- JSR-372: JavaServer Faces 2.3
- JSR-345: Enterprise JavaBeans 3.2 (lite)
- JSR-224: Java API for XML-Based Web Services 2.3

4.2.1. Base Layers

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

datasources-web-server

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

This layer does not include MicroProfile capabilities.

The following Jakarta EE specifications are supported in this layer:

- JSR-374: Java API for JSON Processing 1.1
- JSR-367: Java API for JSON Binding 1.0
- JSR-369: Java Servlet 4.0
- JSR-341: Expression Language 3.0
- JSR-245: JavaServer Pages 2.3
- JSR-52: Standard Tag Library for JavaServer Pages (JSTL) 1.2
- JSR-236: Concurrency Utilities for Java EE 1.0
- JSR-250: Common Annotations for the Java Platform 1.3
- JSR-222: Java Architecture for XML Binding (JAXB) 2.3
- JSR-45: Debugging Support for Other Languages 1.0
- JSR-907: Java Transaction API 1.2
- JSR-322: Java EE Connector Architecture 1.7

jaxrs-server

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

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

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

The following MicroProfile capability is included in this layer:

- MicroProfile REST Client

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

- JSR-365: Contexts and Dependency Injection for Java 2.0
- JSR-330: Dependency Injection for Java 1.0
- JSR-380: Bean Validation 2.0
- JSR-318: Interceptors 1.2
- JSR-370: Java API for RESTful Web Services 2.1
- JSR-338: Java 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:

- MicroProfile Health
- MicroProfile Metrics
- MicroProfile Config
- OpenTracing

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

- JSR-375: Java EE Security API 1.0

4.2.2. Decorator Layers

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

sso

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

observability

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

- MicroProfile Health
- MicroProfile Metrics
- MicroProfile Config
- OpenTracing



NOTE

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

web-clustering

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

4.3. PROVISIONING USER-DEVELOPED LAYERS IN JBOSS EAP

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

Procedure

1. Build a custom layer using the Galleon Maven plugin.
For more information, see [Building Custom Layers for JBoss EAP](#)
2. Deploy the custom layer to an accessible Maven repository.

3. 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, see [Custom Provisioning Files for JBoss EAP](#)
4. Run the S2I process to provision a JBoss EAP server in OpenShift.
For more information, see [Building an Application Provisioned with User-developed Layers](#)

4.3.1. Building Custom Layers for JBoss EAP

Use the **build-user-feature-pack** goal in the Galleon Maven plugin to build custom layers.

Custom layers depend on at least a base layer. Select the base layer that provides the capabilities you need for your custom layer.

Within the Maven project, the default location for layer content is the directory **src/main/resources**.

For example, to create layers to provision support for PostgreSQL and a PostgreSQL datasource, in the directory **src/main/resources** create the **layers/standalone** subdirectories. The **standalone** subdirectory includes the following content.

- **postgresql-driver**

This directory contains a **layer-spec.xml** with the following contents.

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

- **postgresql-datasource**

This directory contains a **layer-spec.xml** with the following contents.

```
<?xml version="1.0" ?>
<layer-spec xmlns="urn:jboss:galleon:layer-spec:1.0" name="postgresql-datasource">
  <dependencies>
    <layer name="postgresql-driver"/>
  </dependencies>
  <feature spec="subsystem.datasources.data-source">
    <param name="use-ccm" value="true"/>
    <param name="data-source" value="PostgreSQLDS"/>
    <param name="enabled" value="true"/>
    <param name="use-java-context" value="true"/>
    <param name="jndi-name"
value="java:jboss/datasources/${env.POSTGRESQL_DATASOURCE,env.OPENSIFT_POS
TGRESQL_DATASOURCE:PostgreSQLDS}"/>
  </feature>
</layer-spec>
```

```

    <param name="connection-url"
value="jdbc:postgresql://${env.POSTGRESQL_SERVICE_HOST},\
env.OPENSIFT_POSTGRESQL_DB_HOST}:${env.POSTGRESQL_SERVICE_PORT},\
env.OPENSIFT_POSTGRESQL_DB_PORT}/${env.POSTGRESQL_DATABASE,\
env.OPENSIFT_POSTGRESQL_DB_NAME}"/>
    <param name="driver-name" value="postgresql"/>
    <param name="user-name" value="${env.POSTGRESQL_USER,\
env.OPENSIFT_POSTGRESQL_DB_USERNAME}"/>
    <param name="password" value="${env.POSTGRESQL_PASSWORD,\
env.OPENSIFT_POSTGRESQL_DB_PASSWORD}"/>
    <param name="check-valid-connection-sql" value="SELECT 1"/>
    <param name="background-validation" value="true"/>
    <param name="background-validation-millis" value="60000"/>
    <param name="flush-strategy" value="IdleConnections"/>
    <param name="statistics-enabled" value="${wildfly.datasources.statistics-
enabled:${wildfly.statistics-enabled:false}}"/>
  </feature>

```

Additional Resources

[Base Layers](#)

[WildFly Galleon Maven Plugin Documentation](#)

[Example illustrating packaging of drivers and datasources as Galleon layers](#)

4.3.2. Custom Provisioning Files for JBoss EAP

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

The following code illustrates a custom provisioning file.

```

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

```

- 1 This element instructs the provisioning process to provision the current eap-s2i feature-pack. Note that a builder image includes only one feature pack.
- 2 This element instructs the provisioning process to exclude default configurations.
- 3 This element instructs the provisioning process to exclude default packages.
- 4 This element instructs the provisioning process to provision the **com.example.demo:my-galleon-feature-pack:1.0** 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 and the **my-custom-driver** and **my-custom-datasource** custom layers from the **com.example.demo:my-galleon-feature-pack:1.0** feature pack.
- 6 This element instructs the provisioning process to optimize provisioning of JBoss EAP modules.

4.3.3. Building an Application Provisioned with User-developed Layers

When you build an application from a directory that includes a custom provisioning file, the S2I build process detects the provisioning file and provisions the JBoss EAP server as instructed.

Prerequisites

- The user-developed layers must exist in an accessible Maven repository.
- The application directory must contain a valid provisioning file that refers to the user-developed layers and the feature pack that contains them.

Procedure

- Enter a standard S2I build command to build the application.
For example, assume you create the following custom provisioning file in your application directory.

```
<?xml version="1.0" ?>
<installation xmlns="urn:jboss:galleon:provisioning:3.0">
  <feature-pack location="eap-s2i@maven(org.jboss.universe:s2i-universe)">
    <default-configs inherit="false"/>
    <packages inherit="false"/>
  </feature-pack>
  <feature-pack location="com.example.demo:my-galleon-feature-pack:1.0">
    <default-configs inherit="false"/>
    <packages inherit="false"/>
  </feature-pack>
  <config model="standalone" name="standalone.xml">
    <layers>
      <include name="cloud-server"/>
      <include name="my-custom-driver"/>
      <include name="my-custom-datasource"/>
    </layers>
  </config>
</options>
```



```
<option name="optional-packages" value="passive+"/>
</options>
</installation>
```

The following command builds an application using the **com.example.demo:my-galleon-feature-pack:1.0** feature pack, which includes the **my-custom-driver** and **my-custom-datasource** layers. The resulting application is named **eap-my-custom-db**. The connection to the database is configured using environment variables.

```
oc build my-app \
-e DEMO_DB=demo \
-e DEMO_PASSWORD=demo \
-e DEMO_HOST=127.0.0.1 \
-e DEMO_PORT=5432 \
-e DEMO_USER=demo \
eap-my-custom-db
```

You can log in to the database on port 5432 with the user **demo** and the password **demo**.

Additional Resources

[Custom Provisioning Files for JBoss EAP](#)

CHAPTER 5. MIGRATION OF APPLICATIONS FROM JBOSS EAP IMAGESTREAMS ON OPENSIFT 3 TO EAP73 IMAGESTREAMS

Applications developed for the **eap72** imagestream require changes to function correctly in the **eap73** imagestream.

5.1. UPDATES TO LIVENESS AND READINESS PROBE CONFIGURATION FOR EAP73 IMAGESTREAMS

The **YAML** configuration of probes must be adjusted when migrating from the **eap72** image running on OpenShift 3.11 to any **eap73** image.

On the **eap72** image, the default **YAML** configuration for a liveness probe is similar to the following code example:

Example YAML Configuration for eap72 Image on OpenShift 3.11 Liveness Probe

```
livenessProbe:
  exec:
    command:
      - /bin/bash
      - '-c'
      - /opt/eap/bin/livenessProbe.sh
  initialDelaySeconds: 60
  periodSeconds: 10
  successThreshold: 1
  failureThreshold: 3
```

In this example, the liveness probe is located at **/opt/eap/bin/livenessProbe.sh** within the JBoss EAP image. The probe is triggered the first time after a 60 second initial delay and then every 10 seconds after a pod is started on the JBoss EAP server.

After three unsuccessful probes, the container is deemed unhealthy and OpenShift restarts the container in its pod.

On the **eap72** image, a single call lasts 5 seconds before it returns as a success or failure. The call is followed by a 10 second waiting period. This means that 3 calls last approximately 35 seconds before the container inside the pod is restarted if the JBoss EAP image is unhealthy.

On any **eap73** image, a single call lasts less than 1 second. Three calls last approximately 23 seconds. The configuration of the probe for **eap73** images should be adjusted in the **YAML** configuration as follows:

Example YAML Configuration for any eap73 Imagestream Liveness Probe

```
livenessProbe:
  exec:
    command:
      - /bin/bash
      - '-c'
      - /opt/eap/bin/livenessProbe.sh
  initialDelaySeconds: 60
```

```
periodSeconds: 16
successThreshold: 1
failureThreshold: 3
```

In this example, **periodSeconds** has been increased by 6 seconds. Now the first call lasts 1 second, followed by a 16 second waiting period. Three calls would last approximately 34 seconds, which is nearly equivalent to the **eap72** image behavior of the probe.

In the readiness probe, update **periodSeconds** in the **YAML** configuration with a similar value.

Example YAML Configuration for any eap73 Imagestream Readiness Probe

```
readinessProbe:
  exec:
    command:
      - /bin/bash
      - '-c'
      - /opt/eap/bin/readinessProbe.sh
  initialDelaySeconds: 10
  periodSeconds: 16
  successThreshold: 1
  failureThreshold: 3
```

Additional Resources

[Monitoring Container Health on OpenShift 4](#)

[Liveness and Readiness Probes on OpenShift 3.11](#)

5.2. DEFAULT DATASOURCE REMOVED

In JBoss EAP 7.3, the default datasource is removed from JBoss EAP imagestreams.

If you developed custom applications that use the default datasource, you can include it when provisioning a server. Use the **ENABLE_GENERATE_DEFAULT_DATASOURCE** environment variable with a value of **true**.

```
ENABLE_GENERATE_DEFAULT_DATASOURCE=true
```

CHAPTER 6. TROUBLESHOOTING

6.1. TROUBLESHOOTING POD RESTARTS

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

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

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

```
$ oc get pods
```

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

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

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

```
$ oc get events
```

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

6.2. TROUBLESHOOTING USING THE JBOSS EAP MANAGEMENT CLI

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



IMPORTANT

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

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

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

```
$ oc rsh POD_NAME
```

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

```
█ $ /opt/eap/bin/jboss-cli.sh
```

CHAPTER 7. EAP OPERATOR FOR AUTOMATING APPLICATION DEPLOYMENT ON OPENSHIFT

EAP operator is a JBoss EAP-specific controller that extends the OpenShift API. You can use the EAP operator to create, configure, and manage 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 EJB remoting and transaction recovery processing. The **StatefulSet** ensures persistent storage and network hostname stability even after pods are restarted.

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

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

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

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

7.1. INSTALLING EAP OPERATOR USING THE WEB CONSOLE

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

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

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



NOTE

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

Prerequisites

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

Procedure

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



NOTE

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

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

7.2. INSTALLING EAP OPERATOR USING THE CLI

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

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

- **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 that update request to have the operator updated to the new version.

When installing the EAP operator from the OperatorHub using the CLI, use the **oc** command to create or update a **CatalogSourceConfig** object, then add a **Subscription** object. The [web console](#) version of this procedure handles the creation of the **CatalogSourceConfig** and **Subscription** objects behind the scenes for you, appearing as if it was one step.

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
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. You must have an OperatorGroup if the installMode you chose is the SingleNamespace mode. For more information about OperatorGroups, see the [OperatorGroups](#) section in the *Working with Operators in OpenShift Container Platform* guide. The following is an example **Subscription** object YAML file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: eap
  namespace: openshift-operators
spec:
  channel: alpha
```



```
installPlanApproval: Automatic
name: eap
source: redhat-operators
sourceNamespace: openshift-marketplace
```

- name: Name of the operator to subscribe to.
- source: Name of the CatalogSource that was created.

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
```

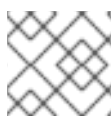
The EAP operator is successfully installed. At this point, the OLM is aware of the EAP operator. A ClusterServiceVersion (CSV) for the operator appears in the target namespace, and APIs provided by the EAP operator is available for creation.

7.3. DEPLOYING A JAVA APPLICATION ON OPENSIFT USING THE EAP OPERATOR

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

Prerequisites

- You have installed EAP operator. For more information about installing the EAP operator, see [Installing EAP Operator Using the Webconsole](#) and [Installing EAP Operator Using the CLI](#).
- You have built a Docker image of the user application using JBoss EAP for OpenShift Source-to-Image (S2I) build image.
- You have created a **Secret** object, if your application's CustomResourceDefinition (CRD) file references one. For more information about creating a new **Secret** object, see [Creating a Secret](#).
- You have created a **ConfigMap**, if your application's CRD file references one. For information about creating a **ConfigMap**, see [Creating a ConfigMap](#).
- You have created a **ConfigMap** from the **standalone.xml** file, if you choose to do so. For information about creating a **ConfigMap** from the **standalone.xml** file, see [Creating a ConfigMap from a standalone.xml File](#).



NOTE

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

Procedure

1. Access OperatorHub using your web browser.
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). 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

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. To find the stored **Secret**:

```
$ 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-openshift.xml
```



NOTE

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

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

```
spec:
  disableHTTPRoute: true
```

7.3.1. 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'
```

7.3.2. 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
```

7.3.3. 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

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

Procedure

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

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

7.3.4. 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.

7.4. UNINSTALLING EAP OPERATOR USING WEB CONSOLE

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



NOTE

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

You can uninstall the EAP operator using the web console.

**WARNING**

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

Procedure

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

7.5. UNINSTALLING EAP OPERATOR USING THE CLI

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

**NOTE**

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

You can uninstall the EAP operator using the command line.

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

**WARNING**

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

Procedure

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

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

2. Delete the EAP operator's subscription:

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

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

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

7.6. EAP OPERATOR FOR SAFE TRANSACTION RECOVERY

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

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 EJB 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 EJB calls that target it.

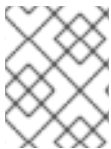
If you want to monitor the state of the scaledown process, observe the status of the **wildflyserver** instance. For more information, see [Monitoring the Scaledown Process](#). For information about pod statuses during scaledown, see [Pod Status During Scaledown](#).

7.6.1. StatefulSets for Stable Network Host Names

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

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

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



NOTE

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

7.6.2. Monitoring the Scaledown Process

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

Procedure

- To observe the state of the scaledown process:

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

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

7.6.2.1. Pod Status During Scaledown

The following table describes the different pod statuses during scaledown:

Table 7.1. Pod Status Description

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

7.6.3. 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 EJB client recovery folder.
 - a. Remove all files from the pod EJB client recovery directory:

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

4. The status of your pod changes to **SCALING_DOWN_CLEAN** and the pod is terminated.

7.7. EJB REMOTING ON OPENSIFT

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



NOTE

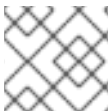
When deploying on OpenShift, consider the use of the [EAP operator](#). The EAP operator uses [StatefulSet](#) for the appropriate handling of EJB remoting and transaction recovery processing. The [StatefulSet](#) ensures persistent storage and network hostname stability even after pods are restarted.

Network hostname stability is required when the JBoss EAP instance is contacted using an EJB 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.

An EJB remote caller has two options to configure the remote calls:

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

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



NOTE

The name of the target EJB 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 EJB call reaches any EAP instance connected to the cluster. A bootstrap connection is used to initialize the EJB client, which gathers the structure of the EAP cluster as the next step.

7.7.1. Configuring EJB on OpenShift

You must configure the JBoss EAP servers that act as callers for EJB remoting. The target server must configure a user with permission to receive the EJB 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 EJB 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 user and password you configured.

Example Configuration

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

CHAPTER 8. REFERENCE INFORMATION



NOTE

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

8.1. PERSISTENT TEMPLATES

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

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

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

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

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

8.2. INFORMATION ENVIRONMENT VARIABLES

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

Table 8.1. Information Environment Variables

Variable Name	Description and Value
JBOSS_IMAGE_NAME	<p>The image names.</p> <p>Values:</p> <ul style="list-style-type: none"> • jboss-eap-7/{EAPOpenShiftImageNameJDK8} (JDK 8 / RHEL 7) • jboss-eap-7/{EAPOpenShiftImageNameJDK11} (JDK 11 / RHEL 8)

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

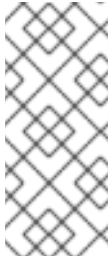
8.3. CONFIGURATION ENVIRONMENT VARIABLES

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

Table 8.2. Configuration Environment Variables

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

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

Variable Name	Description
AB_JOLOKIA_PASSWORD_RANDOM	<p>Determines if a random AB_JOLOKIA_PASSWORD should be generated.</p> <p>Set to true to generate a random password. The generated value is saved in the <code>/opt/jolokia/etc/jolokia.pw</code> file.</p>
AB_JOLOKIA_PORT	<p>The port to listen to.</p> <p>Defaults to 8778.</p> <p>Example value: 5432</p>
AB_JOLOKIA_USER	<p>The name of the user to use for basic authentication.</p> <p>Defaults to jolokia.</p> <p>Example value: myusername</p>
AB_PROMETHEUS_ENABLE	<p>If set to true, this variable activates the jmx-exporter java agent that exposes Prometheus format metrics. Default is set to false.</p> <div data-bbox="687 1039 798 1294" style="display: inline-block; vertical-align: middle;">  </div> <p>NOTE</p> <p>The MicroProfile Metrics subsystem is the preferred method to expose data in the Prometheus format. For more information about the MicroProfile Metrics subsystem, see Eclipse MicroProfile in the <i>Configuration Guide</i> for JBoss EAP.</p>
AB_PROMETHEUS_JMX_EXPORTER_CONFIG	<p>The path within the container to a user-specified configuration.yaml for the jmx-exporter agent to use instead of the default configuration.yaml file. To find out more about the S2I mechanism to incorporate additional configuration files, see S2I Artifacts.</p>
AB_PROMETHEUS_JMX_EXPORTER_PORT	<p>The port on which the jmx-exporter agent listens for scrapes from the Prometheus server. Default is 9799. The agent listens on localhost. Metrics can be made available outside of the container by configuring the DeploymentConfig file for the application to include the service exposing this endpoint.</p>
CLI_GRACEFUL_SHUTDOWN	<p>If set to any non-zero length value, the image will prevent shutdown with the TERM signal and will require execution of the shutdown command using the JBoss EAP management CLI.</p> <p>Example value: true</p>

Variable Name	Description
CONTAINER_HEAP_PERCENT	<p>Set the maximum Java heap size, as a percentage of available container memory.</p> <p>Example value: 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 JMS 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. <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_APP END	<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 secret specified via JGROUPS_ENCRYPT_SECRET variable, 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 of the keystore file within the secret specified via JGROUPS_ENCRYPT_SECRET variable, 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: /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>

Variable Name	Description
JGROUPS_ENCRYPT_SECRET	Name of the secret, containing the JGroups keystore file used for securing the JGroups communications, when using SYM_ENCRYPT JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued. Example value: eap7-app-secret
JGROUPS_PING_PROTOCOL	JGroups protocol to use for node discovery. Can be either openshift.DNS_PING or openshift.KUBE_PING .
MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION	For backwards compatibility, set to true to use MyQueue and MyTopic as physical destination name defaults instead of queue/MyQueue and topic/MyTopic .
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 will be made to discover the port number from the SRV records for the service, otherwise the default 8888 will be 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.



NOTE

Other environment variables not listed above that can influence the product can be found in the [JBoss EAP documentation](#).

8.4. APPLICATION TEMPLATES

Table 8.3. Application Templates

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

8.5. EXPOSED PORTS

Table 8.4. Exposed Ports

Port Number	Description
8443	HTTPS
8778	Jolokia Monitoring

8.6. DATASOURCES

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

The most important environment variable is **DB_SERVICE_PREFIX_MAPPING**, as it defines JNDI mappings for the datasources. The allowed value for this variable is a comma-separated list of **POOLNAME-DATABASETYPE=PREFIX** triplets, where:

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

8.6.1. JNDI Mappings for Datasources

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



NOTE

The first part (before the equal sign) of the **DB_SERVICE_PREFIX_MAPPING** should be lowercase.

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

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

**WARNING**

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

8.6.1.1. Database Drivers**IMPORTANT**

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

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

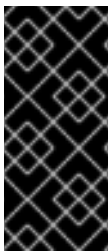
- MySQL
- PostgreSQL

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

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

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

Every JDK 8 image contains Java drivers for MySQL, PostgreSQL, and MongoDB databases deployed. Datasources are generated only for MySQL and PostgreSQL databases.

**IMPORTANT**

JDK 11 imagestreams do not contain drivers for MySQL, PostgreSQL, and MongoDB databases and the datasources are not generated.

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

**NOTE**

For MongoDB database there are no JNDI mappings created because MongoDB is not a SQL database.

8.6.1.2. 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
<i>POOLNAME_DATABASETYPE_SERVICE_HOST</i>	<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>
<i>POOLNAME_DATABASETYPE_SERVICE_PORT</i>	<p>Defines the database server's port for the datasource.</p> <p>Example value: 5432</p>
<i>PREFIX_BACKGROUND_VALIDATION</i>	<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>
<i>PREFIX_BACKGROUND_VALIDATION_MILLIS</i>	<p>Specifies frequency of the validation, in milliseconds, when the background-validation database connection validation mechanism is enabled (<i>PREFIX_BACKGROUND_VALIDATION</i> variable is set to true). Defaults to 10000.</p>
<i>PREFIX_CONNECTION_CHECKER</i>	<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>
<i>PREFIX_DATABASE</i>	<p>Defines the database name for the datasource.</p> <p>Example value: myDatabase</p>
<i>PREFIX_DRIVER</i>	<p>Defines Java database driver for the datasource.</p> <p>Example value: postgresql</p>
<i>PREFIX_EXCEPTION_SORTER</i>	<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
<code>PREFIX_JNDI</code>	<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>
<code>PREFIX_JTA</code>	<p>Defines Java Transaction API (JTA) option for the non-XA datasource. The XA datasources are already JTA capable by default.</p> <p>Defaults to true.</p>
<code>PREFIX_MAX_POOL_SIZE</code>	<p>Defines the maximum pool size option for the datasource.</p> <p>Example value: 20</p>
<code>PREFIX_MIN_POOL_SIZE</code>	<p>Defines the minimum pool size option for the datasource.</p> <p>Example value: 1</p>
<code>PREFIX_NONXA</code>	<p>Defines the datasource as a non-XA datasource. Defaults to false.</p>
<code>PREFIX_PASSWORD</code>	<p>Defines the password for the datasource.</p> <p>Example value: password</p>
<code>PREFIX_TX_ISOLATION</code>	<p>Defines the java.sql.Connection transaction isolation level for the datasource.</p> <p>Example value: TRANSACTION_READ_UNCOMMITTED</p>
<code>PREFIX_URL</code>	<p>Defines connection URL for the datasource.</p> <p>Example value: jdbc:postgresql://localhost:5432/postgresdb</p>
<code>PREFIX_USERNAME</code>	<p>Defines the username for the datasource.</p> <p>Example value: admin</p>

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

8.6.1.3. Examples

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

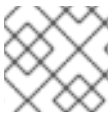
8.6.1.3.1. Single Mapping

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

This creates a datasource with **java:jboss/datasources/test_postgresql** name. Additionally, all the required settings like password and username are expected to be provided as environment variables with the **TEST_** prefix, for example **TEST_USERNAME** and **TEST_PASSWORD**.

8.6.1.3.2. Multiple Mappings

You can specify multiple datasource mappings.



NOTE

Always separate multiple datasource mappings with a comma.

Consider the following value for the **DB_SERVICE_PREFIX_MAPPING** environment variable: **cloud-postgresql=CLOUD,test-mysql=TEST_MYSQL**.

This creates the following two datasources:

1. **java:jboss/datasources/test_mysql**
2. **java:jboss/datasources/cloud_postgresql**

Then you can use **TEST_MYSQL** prefix for configuring things like the username and password for the MySQL datasource, for example **TEST_MYSQL_USERNAME**. And for the PostgreSQL datasource, use the **CLOUD_** prefix, for example **CLOUD_USERNAME**.

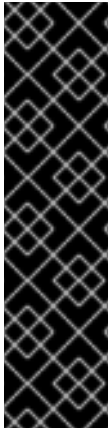
8.7. CLUSTERING

8.7.1. Configuring a JGroups Discovery Mechanism

To enable JBoss EAP clustering on OpenShift, configure the JGroups protocol stack in your JBoss EAP configuration to use either the **kubernetes.KUBE_PING** or **openshift.DNS_PING** discovery mechanism.

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

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



IMPORTANT

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

The **openshift.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 **openshift.DNS_PING** mechanism for discovery and the other using the **kubernetes.KUBE_PING** mechanism. Similarly, when performing a rolling upgrade, the discovery mechanism needs to be identical for both the source and the target clusters.

8.7.1.1. Configuring KUBE_PING

To use the **KUBE_PING** JGroups discovery mechanism:

1. The JGroups protocol stack must be configured to use **KUBE_PING** as the discovery mechanism.
You can do this by setting the **JGROUPS_PING_PROTOCOL** environment variable to **kubernetes.KUBE_PING**:

```
JGROUPS_PING_PROTOCOL=kubernetes.KUBE_PING
```

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

```
KUBERNETES_NAMESPACE=PROJECT_NAME
```

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

```
KUBERNETES_LABELS=application=APP_NAME
```

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

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

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

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



NOTE

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

8.7.1.2. Configuring DNS_PING

To use the **DNS_PING** JGroups discovery mechanism:

1. The JGroups protocol stack must be configured to use **DNS_PING** as the discovery mechanism. You can do this by setting the **JGROUPS_PING_PROTOCOL** environment variable to **openshift.DNS_PING**:

```
JGROUPS_PING_PROTOCOL=openshift.DNS_PING
```

2. The **OPENSHIFT_DNS_PING_SERVICE_NAME** environment variable must be set to the name of the ping service for the cluster. If not set, the server will act as if it is a single-node cluster (a "cluster of one").

```
OPENSHIFT_DNS_PING_SERVICE_NAME=PING_SERVICE_NAME
```

3. The **OPENSHIFT_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**.

```
OPENSHIFT_DNS_PING_SERVICE_PORT=PING_PORT
```

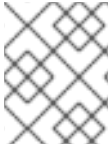
4. A ping service which exposes the ping port must be defined. This service should be headless (ClusterIP=None) and must have the following:
 - a. The port must be named.
 - b. The service must be annotated with **service.alpha.kubernetes.io/tolerate-unready-endpoints** set to **"true"**.



NOTE

Omitting this annotation will result in each node forming their own "cluster of one" during startup, then merging their cluster into the other nodes' clusters after startup, as the other nodes are not detected until after they have started.

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

**NOTE**

DNS_PING does not require any modifications to the service account and works using the default permissions.

8.7.2. Configuring JGroups to Encrypt Cluster Traffic

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

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

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

**IMPORTANT**

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

8.7.2.1. Configuring SYM_ENCRYPT

To use the **SYM_ENCRYPT** protocol to encrypt JGroups cluster traffic:

1. The JGroups protocol stack must be configured to use **SYM_ENCRYPT** as the encryption protocol.

You can do this by setting the **JGROUPS_ENCRYPT_PROTOCOL** environment variable to **SYM_ENCRYPT**:

```
JGROUPS_ENCRYPT_PROTOCOL=SYM_ENCRYPT
```

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

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

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

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

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

```
JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
```

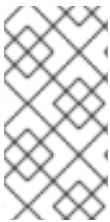
- The **JGROUPS_ENCRYPT_NAME** environment variable must be set to the name associated with the server's certificate. If not set, cluster communication is not encrypted and a warning is issued. For example:

```
JGROUPS_ENCRYPT_NAME=jgroups
```

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

```
JGROUPS_ENCRYPT_PASSWORD=mypassword
```

8.7.2.2. Configuring ASYM_ENCRYPT



NOTE

JBoss EAP 7.3 includes a new version of the **ASYM_ENCRYPT** protocol. The previous version of the protocol is deprecated. If you specify the **JGROUPS_CLUSTER_PASSWORD** environment variable, the deprecated version of the protocol is used and a warning is printed in the pod log.

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

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

8.8. HEALTH CHECKS

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

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

Table 8.5. Liveness and Readiness Checks

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

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

8.9. MESSAGING

8.9.1. Configuring External Red Hat AMQ Brokers

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

Example OpenShift Application Definition

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

Example: JDK 8

```
oc new-app eap73-amq-s2i \
-p APPLICATION_NAME=eap73-mq \
-p MQ_USERNAME=MY_USERNAME \
-p MQ_PASSWORD=MY_PASSWORD
```

Example: JDK 11

```
oc new-app eap73-openjdk11-amq-s2i \
-p APPLICATION_NAME=eap73-mq \
-p MQ_USERNAME=MY_USERNAME \
-p MQ_PASSWORD=MY_PASSWORD
```



IMPORTANT

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

8.10. SECURITY DOMAINS

To configure a new Security Domain, the user must define the **SECDOMAIN_NAME** environment variable.

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

Table 8.6. Security Domains

Variable name	Description
SECDOMAIN_NAME	Defines an additional security domain. Example value: 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

8.11. HTTPS ENVIRONMENT VARIABLES

Variable name	Description
HTTPS_NAME	If defined along with HTTPS_PASSWORD and HTTPS_KEYSTORE , enables HTTPS and sets the SSL name. This should be the value specified as the alias name of your keystore if you created it with the keytool -genkey command. Example value: example.com
HTTPS_PASSWORD	If defined along with HTTPS_NAME and HTTPS_KEYSTORE , enables HTTPS and sets the SSL key password. Example value: passw0rd
HTTPS_KEYSTORE	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 Example value: ssl.key

8.12. ADMINISTRATION ENVIRONMENT VARIABLES

Table 8.7. 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>

8.13. S2I

The image includes S2I scripts and Maven.

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

Only WAR deployments are supported at this time.

8.13.1. Custom Configuration

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

8.13.1.1. Custom Modules

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

8.13.2. Deployment Artifacts

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

8.13.3. Artifact Repository Mirrors

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

Benefits of using a mirror are:

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

- Improved build times.

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

1. Identify the name of the build configuration to apply **MAVEN_MIRROR_URL** variable against.

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

2. Update build configuration of **eap** with a **MAVEN_MIRROR_URL** environment variable.

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

3. Verify the setting.

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

4. Schedule new build of the application.



NOTE

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

8.13.3.1. Secure Artifact Repository Mirror URLs

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

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

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

8.13.4. Scripts

run

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

assemble

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

8.13.5. Custom Scripts

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

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

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

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

8.13.5.1. Mounting a configmap to execute custom scripts

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

To mount a configmap:

1. Create a configmap with content you want to include in the `postconfigure.sh`. For example, if a subdirectory called **extensions** included the scripts **postconfigure.sh** and **extensions.cli**, you would run the following command:

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

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

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

Example `postconfigure.sh`

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

Example `extensions.cli`

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

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

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

To execute custom scripts using `install.sh`:

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


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

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

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

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

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

8.13.6. Environment Variables

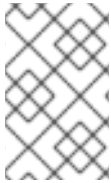
You can influence the way the build is executed by supplying environment variables to the **s2i build** command. The environment variables that can be supplied are:

Table 8.8. s2i Environment Variables

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

Variable name	Description
HTTP_PROXY_HOST	Host name or IP address of a HTTP proxy for Maven to use. Example value: 192.168.1.1
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 Section 8.13.3.1, "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

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

**NOTE**

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

8.14. SSO

This image contains support for Red Hat JBoss SSO-enabled applications.

**NOTE**

See the [Red Hat JBoss SSO for OpenShift documentation](#) for more information on how to deploy the Red Hat JBoss SSO for OpenShift image with the JBoss EAP for OpenShift image.

Table 8.9. SSO Environment Variables

Variable name	Description
SSO_URL	URL of the SSO server.
SSO_REALM	SSO realm for the deployed applications.
SSO_PUBLIC_KEY	Public key of the SSO Realm. This field is optional but if omitted can leave the applications vulnerable to man-in-middle attacks.
SSO_USERNAME	SSO User required to access the SSO REST API. Example value: mySsoUser
SSO_PASSWORD	Password for the SSO user defined by the SSO_USERNAME variable. Example value: 6fedmL3P
SSO_SAML_KEYSTORE	Keystore location for SAML. Defaults to /etc/ss0-saml-secret-volume/keystore.jks .
SSO_SAML_KEYSTORE_PASSWORD	Keystore password for SAML. Defaults to mykeystorepass .
SSO_SAML_CERTIFICATE_NAME	Alias for keys/certificate to use for SAML. Defaults to jboss .

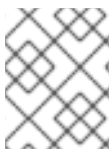
Variable name	Description
SSO_BEARER_ONLY	SSO Client Access Type. (Optional) Example value: true
SSO_CLIENT	Path for SSO redirects back to the application. Defaults to match module-name .
SSO_ENABLE_CORS	If true , enable CORS for SSO applications. (Optional)
SSO_SECRET	The SSO Client Secret for Confidential Access. Example value: KZ1Qylq4
SSO_DISABLE_SSL_CERTIFICATE_VALIDATION	If true the SSL/TLS communication between JBoss EAP and the RH-SSO server will be unsecure, for example, the certificate validation is disabled with curl . Not set by default. Example value: true

8.15. TRANSACTION RECOVERY

When a cluster is scaled down, it is possible for transaction branches to be in doubt. In these cases, [manual transaction recovery](#) might be necessary.

8.15.1. Unsupported Transaction Recovery Scenarios

- JTS transactions
Because the network endpoint of the parent is encoded in recovery coordinator IORs, recovery cannot work reliably if either the child or parent node recovers with either a new IP address, or if it is intended to be accessed using a virtualized IP address.
- XTS transactions
XTS does not work in a clustered scenario for recovery purposes. See [JBTM-2742](#) for details.
- Transactions propagated over [JBoss Remoting](#) is unsupported with OpenShift 3.



NOTE

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

- Transactions propagated over XATerminator
Because the EIS is intended to be connected to a single instance of a Java EE application server, there are no well-defined ways to couple these processes.

8.15.2. Manual Transaction Recovery Process

8.15.2.1. Caveats

This procedure only describes how to manually recover transactions that were wholly self-contained within a single JVM. The procedure does not describe how to recover JTA transactions that have been propagated to other JVMs.



IMPORTANT

There are various network partition scenarios in which OpenShift might start multiple instances of the same pod with the same IP address and same node name and where, due to the partition, the old pod is still running. During manual recovery, this might result in a situation where you might be connected to a pod that has a stale view of the object store. If you think you are in this scenario, it is recommended that all JBoss EAP pods be shut down to ensure that none of the resource managers or object stores are in use.

When you enlist a resource in an XA transaction, it is your responsibility to ensure that each resource type is supported for recovery. For example, it is known that PostgreSQL and MySQL are well-behaved with respect to recovery, but for others, such as A-MQ and JDV resource managers, you should check documentation of the specific OpenShift release.

The deployment must use a [JDBC object store](#).



IMPORTANT

The transaction manager relies on the uniqueness of node identifiers. The maximum byte length of an XID is set by the XA specification and cannot be changed. Due to the data that the JBoss EAP for OpenShift image must include in the XID, this leaves room for 23 bytes in the node identifier.

OpenShift coerces the node identifier to fit this 23 byte limit:

- For all node names, even those under 23 bytes, the - (dash) character is stripped out.
- If the name is still over 23 bytes, characters are truncated from the beginning of the name until length of the name is within the 23 byte limit.

However, this process might impact the uniqueness of the identifier. For example, the names **aaa123456789012345678m0jwh** and **bbb123456789012345678m0jwh** are both truncated to **123456789012345678m0jwh**, which breaks the uniqueness of the names that are expected. In another example, **this-pod-is-m0jwh** and **thispod-is-m0jwh** are both truncated to **thispodism0jwh**, again breaking the uniqueness of the names.

It is your responsibility to ensure that the node names you configure are unique, keeping in mind the above truncation process.

8.15.2.2. Prerequisite

It is assumed the OpenShift instance has been configured with a JDBC store, and that the store tables are partitioned using a table prefix corresponding to the pod name. This should be automatic whenever a JBoss EAP deployment is in use. You can verify that the JBoss EAP instance is using a JDBC object store by looking at the configuration of the transactions subsystem in a running pod:

1. Determine if the `/opt/eap/standalone/configuration/openshift-standalone.xml` configuration file contains an element for the transaction subsystem:

```
<subsystem xmlns="urn:jboss:domain:transactions:3.0">
```

- If the JDBC object store is in use, then there is an entry similar to the following:

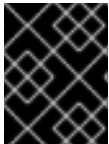
```
<jdbc-store datasource-jndi-name="java:jboss/datasources/jdbcstore_postgresql"/>
```



NOTE

The JNDI name identifies the datasource used to store the transaction logs.

8.15.2.3. Procedure



IMPORTANT

The following procedure details the process of manual transaction recovery solely for datasources.

- Use the database vendor tooling to list the XIDs (transaction branch identifiers) for in-doubt branches. It is necessary to list XIDs *for all datasources that were in use by any deployments running on the pod* that failed or was scaled down. Refer to the vendor documentation for the database product in use.
- For each such XID, determine which pod created the transaction and check to see if that pod is still running.
 - If it is running, then leave the branch alone.
 - If the pod is not running, assume it was removed from the cluster and you must apply the manual resolution procedure described here. Look in the transaction log storage that was used by the failed pod to see if there is a corresponding transaction log:
 - If there is a log, then manually commit the XID using the vendor tooling.
 - If there is not a log, assume it is an orphaned branch and roll back the XID using the vendor tooling.

The rest of this procedure explains in detail how to carry out each of these steps.

8.15.2.3.1. Resolving In-doubt Branches

First, find all the resources that the deployment is using.

It is recommended that you do this using the JBoss EAP management CLI. Although the resources should be defined in the JBoss EAP **standalone-openshift.xml** configuration file, there are other ways they can be made available to the transaction subsystem within the application server. For example, this can be done using a file in a deployment, or dynamically using the management CLI at runtime.

- Open a terminal on a pod running a JBoss EAP instance in the cluster of the failed pod. If there is no such pod, scale up to one.
- Create a management user using the **/opt/eap/bin/add-user.sh** script.
- Log into the management CLI using the **/opt/eap/bin/jboss-cli.sh** script.
- List the datasources configured on the server. These are the ones that may contain in-doubt transaction branches.

```

/subsystem=datasources:read-resource
{
  "outcome" => "success",
  "result" => {
    "data-source" => {
      "ExampleDS" => undefined,
      ...
    },
    ...
  }
}

```

- Once you have the list, find the connection URL for each of the datasources. For example:

```

/subsystem=datasources/data-source=ExampleDS:read-attribute(name=connection-url)
{
  "outcome" => "success",
  "result" => "jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE",
  "response-headers" => {"process-state" => "restart-required"}
}

```

- Connect to each datasource and list any in-doubt transaction branches.



NOTE

The table name that stores in-doubt branches will be different for each datasource vendor.

JBoss EAP has a default SQL query tool (H2) that you can use to check each database. For example:

```

java -cp /opt/eap/modules/system/layers/base/com/h2database/h2/main/h2-1.3.173.jar \
-url "jdbc:postgresql://localhost:5432/postgres" \
-user sa \
-password sa \
-sql "select gid from pg_prepared_xacts;"

```

Alternatively, you can use the resource's native tooling. For example, for a PostgreSQL datasource called **sampledb**, you can use the OpenShift client tools to remotely log in to the pod and query the in-doubt transaction table:

```

$ oc rsh postgresql-2-vwf9n # rsh to the named pod
sh-4.2$ psql sampledb
psql (9.5.7)
Type "help" for help.

sampledb=# select gid from pg_prepared_xacts;
131077_AAAAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGHAtanRhLWNyYXNoLXJlYy0zLXAj
Y2N3_AAAAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGgAAAAEAAAAA

```

8.15.2.3.2. Extract the Global Transaction ID and Node Identifier from Each XID

When all XIDs for in-doubt branches are identified, convert the XIDs into a format that you can compare to the logs stored in the transaction tables of the transaction manager.

For example, the following Bash script can be used to perform this conversion. Assuming that `$PG_XID` holds the XID from the [select statement](#) above, then the JBoss EAP transaction ID can be obtained as follows:

```
PG_XID="$1"
IFS='_' read -ra lines <<< "$PG_XID"
[[ "${lines[0]}" = 131077 ]] || exit 0; # this script only works for our own FORMAT ID
PG_TID=${lines[1]}

a=$(echo "$PG_TID" | base64 -d | xxd -ps | tr -d '\n' | while read -N16 i ; do echo 0x$i ; done)
b=$(echo "$PG_TID" | base64 -d | xxd -ps | tr -d '\n' | while read -N8 i ; do echo 0x$i ; done)
c=("${b[@]:4}") # put the last 3 32-bit hexadecimal numbers into array c
# the negative elements of c need special handling since printf below only works with positive
# hexadecimal numbers
for i in "${c[@]}"; do
  arg=${c[$i]}
  # inspect the MSB to see if arg is negative - if so convert it from a 2's complement number
  [[ $($arg >> 31) = 1 ]] && x=$(echo "obase=16; $(($arg - 0x100000000))" | bc) || x=$arg
  if [[ ${x:0:1} = \- ]] ; then # see if the first character is a minus sign
    neg[$i]="-";
    c[$i]=0x${x:1} # strip the minus sign and make it hex for use with printf below
  else
    neg[$i]=""
    c[$i]=$x
  fi
done
EAP_TID=$(printf %x:%x:%x:%x:%x:%x:%x:%x ${a[0]} ${a[1]} ${c[0]} ${c[1]} ${c[2]})
```

After completion, the `$EAP_TID` variable holds the global transaction ID of the transaction that created this XID. The node identifier of the pod that started the transaction is given by the output of the following bash command:

```
echo "$PG_TID" | base64 -d | tail -c +29
```



NOTE

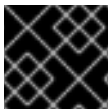
The node identifier starts from the 29th character of the PostgreSQL global transaction ID field.

- If this pod is [still running](#), then leave this in-doubt branch alone since the transaction is still in flight.
- If this pod is not running, then you need to search the relevant transaction log storage for the transaction log. The log storage is located in a JDBC table, which is named following the **os<node-identifier>jbosststxtable** pattern.
 - If there is no such table, leave the branch alone as it is owned by some other transaction manager. The URL for the datasource containing this table is defined in the transaction subsystem description shown below.
 - If there is such a table, look for an entry that matches the global transaction ID.
 - If there is an entry in the table that matches the global transaction ID, then the in-doubt branch needs to be committed using the datasource vendor tooling as described below.

- If there is no such entry, then the branch is an orphan and can safely be rolled back.

An example of how to commit an in-doubt PostgreSQL branch is shown below:

```
$ oc rsh postgresql-2-vwf9n
sh-4.2$ psql sampledb
psql (9.5.7)
Type "help" for help.
psql sampledb
commit prepared '131077_AAAAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGHAtanRh
----
LWNyYXNoLXJlYy0zLXAyY2N3_AAAAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGgAAAAEAAAAA';
```



IMPORTANT

Repeat this procedure for all datasources and in-doubt branches.

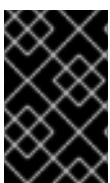
8.15.2.3.3. Obtain the List of Node Identifiers of All Running JBoss EAP Instances in Any Cluster that Can Contact the Resource Managers

Node identifiers are configured to be the same name as the pod name. You can obtain the pod names in use using the **oc** command. Use the following command to list the running pods:

```
$ oc get pods | grep Running
eap-manual-tx-recovery-app-4-26p4r 1/1 Running 0 23m
postgresql-2-vwf9n 1/1 Running 0 41m
```

For each running pod, look in the output of the pod's log and obtain the node name. For example, for first pod shown in the above output, use the following command:

```
$ oc logs eap-manual-tx-recovery-app-4-26p4r | grep "jboss.node.name" | head -1
jboss.node.name = tx-recovery-app-4-26p4r
```



IMPORTANT

The aforementioned [JBoss node name identifier](#) will always be truncated to the maximum length of 23 characters in total by removing characters from the beginning and retaining the trailing characters until the maximum length of 23 characters is reached.

8.15.2.3.4. Find the Transaction Logs

1. The transaction logs reside in a JDBC-backed object store. The JNDI name of this store is defined in the **transaction** subsystem definition of the JBoss EAP configuration file.
2. Look in the configuration file to find the datasource definition corresponding to the above JNDI name.
3. Use the JNDI name to derive the connection URL.
4. You can use the URL to connect to the database and issue a **select** query on the relevant in-doubt transaction table.

Alternatively, if you know which pod the database is running on, and you know the name of the database, it might be easier to open an OpenShift remote shell into the pod and use the database tooling directly.

For example, if the JDBC store is hosted by a PostgreSQL database called **sampledb** running on pod **postgresql-2-vwf9n**, then you can find the transaction logs using the following commands:



NOTE

The `ostxrecoveryapp426p4rjbossstxttable` table name listed in the following command has been chosen since it follows the [pattern for JDBC table names holding the log storage entries](#). In your environment the table name will have similar form:

- Starting with **os** prefix.
- The part in the middle is derived from the [JBoss node name above](#), possibly deleting the "-" (dash) character if present.
- Finally the **jbossstxttable** suffix is appended to create the final name of the table.

```
$ oc rsh postgresql-2-vwf9n
sh-4.2$ psql sampledb
psql (9.5.7)
Type "help" for help.

sampledb=# select uidstring from ostxrecoveryapp426p4rjbossstxttable where
TYPENAME='StateManager/BasicAction/TwoPhaseCoordinator/AtomicAction'
;
      uidstring
-----
0:ffff0a81009d:33789827:5a68b2bf:40
(1 row)
```

8.15.2.3.5. Cleaning Up the Transaction Logs for Reconciled In-doubt Branches



WARNING

Do not delete the log unless you are certain that there are no remaining in-doubt branches.

When all the branches for a given transaction are complete, and all potential resources managers have been checked, including A-MQ and JDV, it is safe to delete the transaction log.

Issue the following command, specify the transaction log to be removed using the appropriate **uidstring**:

```
DELETE FROM ostxrecoveryapp426p4rjbossststxtable where uidstring = UIDSTRING
```



IMPORTANT

If you do not delete the log, then completed transactions which failed after prepare, but which have now been resolved, will never be removed from the transaction log storage. The consequence of this is that unnecessary storage is used and future manual reconciliation will be more difficult.

8.16. INCLUDED JBOSS MODULES

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

Table 8.10. Included JBoss Modules

JBoss Module
org.jboss.as.clustering.common
org.jboss.as.clustering.jgroups
org.jboss.as.ee
org.jboss.logmanager.ext
org.jgroups
org.mongodb
org.openshift.ping
org.postgresql
com.mysql
net.oauth.core



NOTE

The following modules are not included in the JDK 11 image:

- **org.mongodb**
- **org.postgresql**
- **com.mysql**

8.17. EAP OPERATOR: API INFORMATION

The EAP operator introduces the following APIs:

8.17.1. WildFlyServer

WildFlyServer defines a custom JBoss EAP resource.

Table 8.11. Table

Field	Description	Scheme	Required
metadata	Standard object's metadata	metav1.ObjectMeta	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

8.17.2. WildFlyServerList

WildFlyServerList defines a list of JBoss EAP deployments.

Table 8.12. Table

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

8.17.3. WildFlyServerSpec

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

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

Table 8.13. 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

Field	Description	Scheme	Required
standaloneConfigMap	Spec to specify how a standalone configuration can be read from a ConfigMap .	StandaloneConfigMapSpec	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
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

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

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

Table 8.14. Table

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

8.17.5. StandaloneConfigMapSpec

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

Table 8.15. 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

8.17.6. WildFlyServerStatus

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

Table 8.16. WildFlyServerStatus

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

8.17.7. PodStatus

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

Table 8.17. PodStatus

Field	Description	Scheme	Required
name	Name of the pod	string	true
podIP	IP address allocated to the pod	string	true

Field	Description	Scheme	Required
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 2020-03-18 11:18:21 UTC