Red Hat JBoss Enterprise Application Platform 7.4

Getting Started with JBoss EAP for OpenShift Container Platform

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

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

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

Legal Notice

Abstract

Guide to using Red Hat JBoss Enterprise Application Platform for OpenShift

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. To provide feedback, you can highlight the text in a document and add comments. Follow the steps in the procedure to learn about submitting feedback on Red Hat documentation.

Prerequisites

- Log in to the Red Hat Customer Portal.

- In the Red Hat Customer Portal, view the document in **Multi-page HTML** format.

Procedure

1. Click **Feedback** to see existing reader comments.
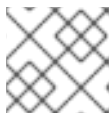
   NOTE

   The feedback feature is enabled only in the **Multi-page HTML** format.

2. Highlight the section of the document where you want to provide feedback.

3. In the prompt menu that displays near the text you selected, click **Add Feedback**.
   A text box opens in the feedback section on the right side of the page.

4. Enter your feedback in the text box and click **Submit**.
   You have created a documentation issue.

5. To view the issue, click the issue tracker link in the feedback view.

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. INTRODUCTION

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

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

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

1.2. HOW DOES JBOSS EAP WORK ON OPENSHIFT?

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

1.3. COMPARISON: JBOSS EAP AND JBOSS EAP FOR OPENSHIFT

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

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

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

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

1.4. VERSION COMPATIBILITY AND SUPPORT

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

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

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

- Default drivers and modules are removed.

- Templates for MySQL and PostgreSQL are removed. You can provision these capabilities with a custom layer.

- The Hawkular agent is not active in these images. If configured, it is ignored.

- The datasource, **ExampleDS**, is no longer added by default at container startup. If you need the default datasource, use the environment variable

ENABLE_GENERATE_DEFAULT_DATASOURCE with a value of **true** (**ENABLE_GENERATE_DEFAULT_DATASOURCE=true**) to include it.

> **NOTE**
>
> The following discovery mechanism protocols were deprecated and are replaced by other protocols:
>
> - The **openshift.DNS_PING** protocol was deprecated and is replaced with the **dns.DNS_PING** protocol. If you referenced the **openshift.DNS_PING** protocol in a **customized standalone-openshift.xml** file, replace the protocol with the **dns.DNS_PING** protocol.
>
> - The **openshift.KUBE_PING** discovery mechanism protocol was deprecated and is replaced with the **kubernetes.KUBE_PING** protocol.

Supported architectures for OpenJDK images

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

1. OpenJDK images and architectures

JDK (OS)	Architecture supported	Red Hat Ecosystem Catalog
OpenJDK8 (RHEL 7)	x86_64	Builder image and Runtime image
OpenJDK11 (RHEL 8)	x86_64, IBM Z, and IBM Power Systems	Builder image and Runtime image

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

Additional resources

- OpenShift and Atomic Platform Tested Integrations

- Capability Trimming in JBoss EAP for OpenShift

1.4.1. OpenShift 4.x Support

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

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

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

1.4.2. IBM Z Support

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

- **journal-type**: Setting the **journal-type** to **ASYNCIO** has no effect. The value of this attribute defaults to **NIO** at runtime.

- **journal-max-io**: This attribute has no effect.

- **journal-store-enable-async-io**: This attribute has no effect.

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

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

Additional resources

Updates to **standalone-openshift.xml** when upgrading JBoss EAP 7.1 to JBoss EAP 7.4 on OpenShift

1.5. DEPLOYMENT OPTIONS

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

- A JBoss EAP for OpenShift template.

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

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

Additional resources

- For more information about JBoss EAP for OpenShift templates, see jboss-eap-openshift-templates on *Git Hub*.

- For more information about the EAP operator, see EAP Operator for Automating Application Deployment on OpenShift.

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

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

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

2.1. PREREQUISITES

You have an OpenShift instance installed and operational. For more information on installing and configuring your OpenShift instance, see OpenShift Container Platform Getting Started guide.

2.2. PREPARE OPENSHIFT FOR APPLICATION DEPLOYMENT

1. Log in to your OpenShift instance using the **oc login** command.

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

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

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

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

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

   > **NOTE**
   >
   > Creating a keystore and a secret is required if you are using any HTTPS-enabled features in your OpenShift project. For example, if you are using the **eap74-https-s2i** template, you must create a keystore and secret.
   >
   > This workflow demonstration for the **kitchensink** quickstart does not use an HTTPS template, so a keystore and secret are not required.

   a. Create a keystore.

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

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

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

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

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

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

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

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

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

2.3. CONFIGURE AUTHENTICATION TO THE RED HAT CONTAINER REGISTRY

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

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

1. Follow the instructions on Red Hat Customer Portal to create an authentication token using a registry service account.

2. Download the YAML file containing the OpenShift secret for the token. You can download the YAML file from the **OpenShift Secret** tab on your token's **Token Information** page.

3. Create the authentication token secret for your OpenShift project using the YAML file that you downloaded:

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

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

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

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

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

2.4. IMPORT THE LATEST JBOSS EAP FOR OPENSHIFT IMAGESTREAMS AND TEMPLATES

You must import the latest JBoss EAP for OpenShift imagestreams and templates for your JDK into the namespace of your OpenShift project.
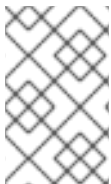
NOTE

Log in to the Red Hat Container Registry using your Customer Portal credentials to import the JBoss EAP imagestreams and templates. For more information, see Red Hat Container Registry Authentication.

Import command for JDK 8

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

This command imports the following imagestreams and templates.

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

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

NOTE

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

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

Import command for JDK 11

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

This command imports the following imagestreams and templates.

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

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

Import command for templates

```
for resource in \
  eap74-amq-persistent-s2i.json \
  eap74-amq-s2i.json \
  eap74-basic-s2i.json \
  eap74-https-s2i.json \
  eap74-sso-s2i.json

do
  oc replace -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-openshift-
templates/eap74/templates/${resource}
done
```

This command imports all templates specified in the command.

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

If you have administrative access to the general **openshift** namespace and want the imagestreams and templates to be accessible by all projects, add **-n openshift** to the **oc replace** line of the command. For example:

```
...
oc replace -n openshift -f \
...
```

If you use the cluster-samples-operator, refer to the OpenShift documentation on configuring the cluster samples operator. See Configuring the Samples Operator for details about configuring the cluster samples operator.

2.5. DEPLOY A JBOSS EAP SOURCE-TO-IMAGE (S2I) APPLICATION TO OPENSHIFT

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

Prerequisites

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

Procedure

1. Create a new OpenShift application that uses the JBoss EAP for OpenShift image and the source code of your Java application. You can use one of the provided JBoss EAP for OpenShift templates for S2I builds. You can also choose to provision a trimmed server.

For example, to deploy the **kitchensink** quickstart using the JDK 8 builder image, enter the following command to use the **eap74-basic-s2i** template in the **eap-demo** project, created in Prepare OpenShift for Application Deployment, with the **kitchensink** source code on GitHub. This quickstart does not support the trimming capability.

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

1 The template to use.

2 The latest imagestreams and templates were imported into the project's namespace, so you must specify the namespace where to find the imagestream. This is usually the project's name.

3 The name of the EAP builder image stream for JDK8.

4 The name of the EAP runtime image stream for JDK8.

5 URL to the repository containing the application source code.

6 The Git repository reference to use for the source code. This can be a Git branch or tag reference.

7 The directory within the source repository to build.

As another example, to deploy the **helloworld-html5** quickstart using the JDK 11 runtime image and trimming JBoss EAP to include only the **jaxrs-server** layer, enter the following command. The command uses the **eap74-basic-s2i** template in the **eap-demo** project, created in Prepare OpenShift for Application Deployment, with the **helloworld-html5** source code on GitHub.

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

1 The template to use.

2 The latest imagestreams and templates were imported into the project's namespace, so you must specify the namespace where to find the imagestream. This is usually the project's name.

3 The name of the EAP builder image stream for JDK11.

4　The name of the EAP runtime image stream for JDK11.

5　URL to the repository containing the application source code.

6　The Git repository reference to use for the source code. This can be a Git branch or tag reference.

7　Provision a trimmed server with only the **jaxrs-server** layer.

8　The directory within the source repository to build.

> **NOTE**
>
> You might also want to configure environment variables when creating your new OpenShift application.
>
> For example, if you are using an HTTPS template such as **eap74-https-s2i**, you must specify the required HTTPS environment variables **HTTPS_NAME**, **HTTPS_PASSWORD**, and **HTTPS_KEYSTORE** to match your keystore details.

> **NOTE**
>
> If the template uses AMQ, you must include the **AMQ_IMAGE_NAME** parameter with the appropriate value.
>
> If the template uses SSO, you must include the **SSO_IMAGE_NAME** parameter with the appropriate value.

2. Retrieve the name of the build configuration.

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

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

2.6. 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 the previous command's output.
   If your application does not use the JBoss EAP root context, append the context of the application to the URL. For example, for the **kitchensink** quickstart, the URL might be **http://HOST_PORT_VALUE/kitchensink/**.

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
```

2.7. CHAINED BUILD SUPPORT IN JBOSS EAP FOR OPENSHIFT

JBoss EAP for OpenShift supports chained builds in OpenShift.

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

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

- The final image, **[application name]**

For details about chained builds, see the OpenShift documentation.

Additional Resources

OpenShift Chained build documentation

CHAPTER 3. CONFIGURING THE JBOSS EAP FOR OPENSHIFT 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 OPENSHIFT S2I PROCESS WORKS

Flowchart illustrating the S2I process for JBoss EAP:
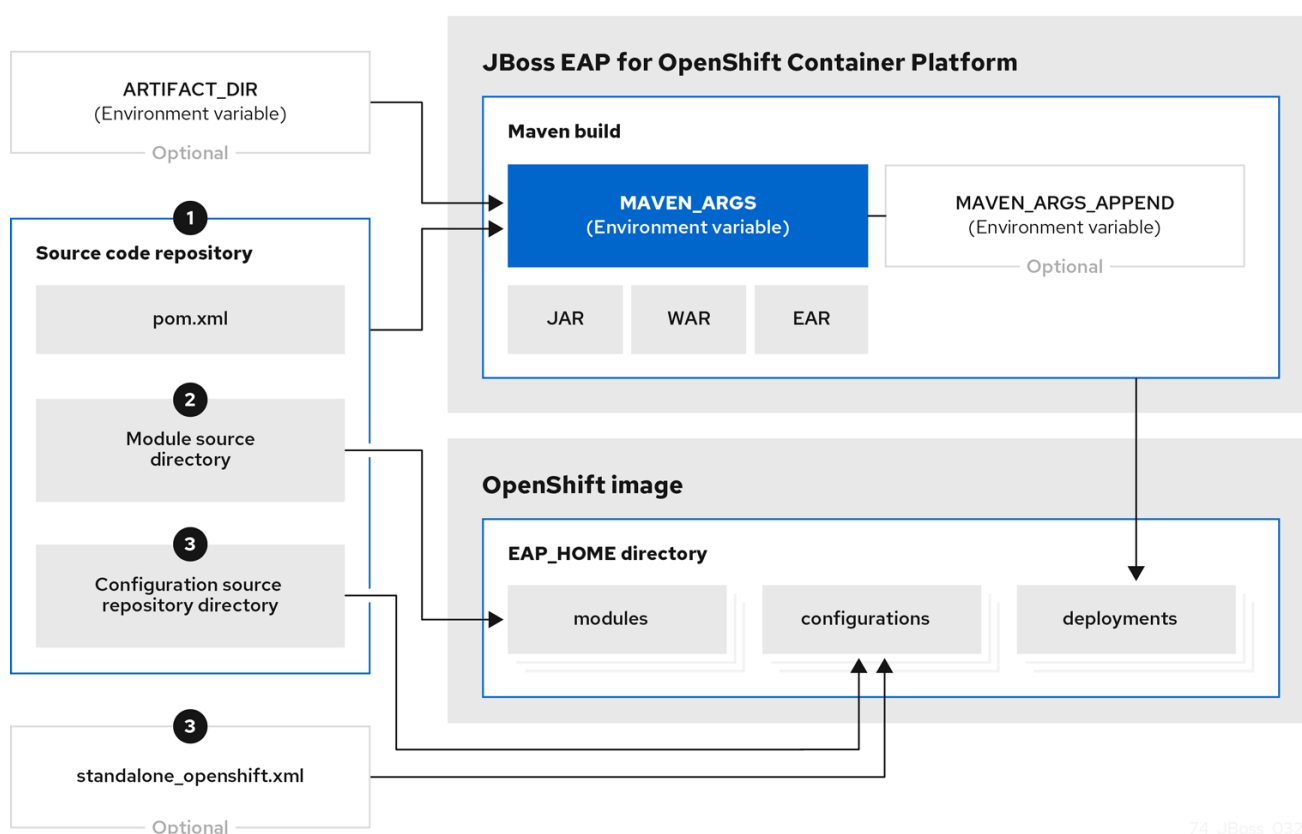
1. If a **pom.xml** file is present in the source code repository, the S2I builder image initiates a Maven build process. The Maven build uses the contents of **$MAVEN_ARGS**.
   If a **pom.xml** file is not present in the source code repository, the S2I builder image initiates a binary type build.

To add custom Maven arguments or options, use **$MAVEN_ARGS_APPEND**. The **$MAVEN_ARGS_APPEND** variable appends options to **$MAVEN_ARGS**.

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

The results of a successful Maven build are copied to the *EAP_HOME*/**standalone**/**deployments**/ directory inside the JBoss EAP for OpenShift image. This includes all JAR, WAR, and EAR files from the source repository specified by the **$ARTIFACT_DIR** environmental variable. The default value of **ARTIFACT_DIR** is the Maven target directory.
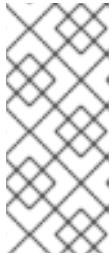
> **NOTE**
>
> To use Maven behind a proxy on JBoss EAP for OpenShift image, set the **$HTTP_PROXY_HOST** and **$HTTP_PROXY_PORT** environment variables. Optionally, you can also set the **$HTTP_PROXY_USERNAME**, **$HTTP_PROXY_PASSWORD**, and **$HTTP_PROXY_NONPROXYHOSTS** variables.

2. All files in the **modules** source repository directory are copied to the *EAP_HOME*/**modules**/ directory in the JBoss EAP for OpenShift image.

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

Additional Resources

- See Binary (local) source on the OpenShift 4.2 documentation for additional information on binary type builds.

- See Artifact Repository Mirrors for additional guidance on how to instruct the S2I process to use the custom Maven artifacts repository mirror.

3.2. CONFIGURING JBOSS EAP FOR OPENSHIFT USING ENVIRONMENT VARIABLES

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

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

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

Available environment variables for the JBoss EAP for OpenShift image are listed in Reference Information.

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,uptimemillis: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. Use **JAVA_OPTS_APPEND** to configure additional JVM settings. If you use **JAVA_OPTS**, some unconfigurable defaults are not added to the server JVM settings. You must explicitly add these settings. Using **JAVA_OPTS** disables certain settings added by default by the container scripts. Disabled settings include

Variable Name	Example	Default Value	JVM Settings	Description
				- -XX:MetaspaceSize=96M - -Djava.net.preferIPv4Stack=true - -Djboss.modules.system.pkgs=jdk.nashorn.api,com.sun.crypto.provider - -Djava.awt.headless=true In addition, if automatic memory calculation is not enabled, the inital Java memory (-Xms) and maximum Java memory (-Xmx) are not defined. Add these defaults if you use **JAVA_OPTS** to configure additional settings.
JAVA_OPTS_APPEND	-Dsome.property=value	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	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**.

Variable Name	Example	Default Value	JVM Settings	Description
JAVA_INITIAL_MEM_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)	The settings depend on the JDK used by the container. • OpenJDK 8: -XX:NativeMemoryTracking=summary -XX:+PrintGC -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UnlockDiagnosticVMOptions • OpenJDK 11: -Xlog:gc:utctime -XX:NativeMemoryTracking=summary	Set the value of this variable to *true* to include diagnostic information in standard output when events occur. If this variable is defined as *true* in an environment where **JAVA_DIAGNOSTICS** has already been defined as *true*, diagnostics are still included.
DEBUG	true	false	-agentlib:jdwp=transport=dt_socket,address=$DEBUG_PORT,server=y,suspend=n	Enables remote debugging.
DEBUG_PORT	8787	8787	-agentlib:jdwp=transport=dt_socket,address=$DEBUG_PORT,server=y,suspend=n	Specifies the port used for debugging.

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

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

The following environment variables are deprecated:

- **JAVA_OPTIONS**: Use **JAVA_OPTS**.

- **INITIAL_HEAP_PERCENT**: Use **JAVA_INITIAL_MEM_RATIO**.

- **CONTAINER_HEAP_PERCENT**: Use **JAVA_MAX_MEM_RATIO**.

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.
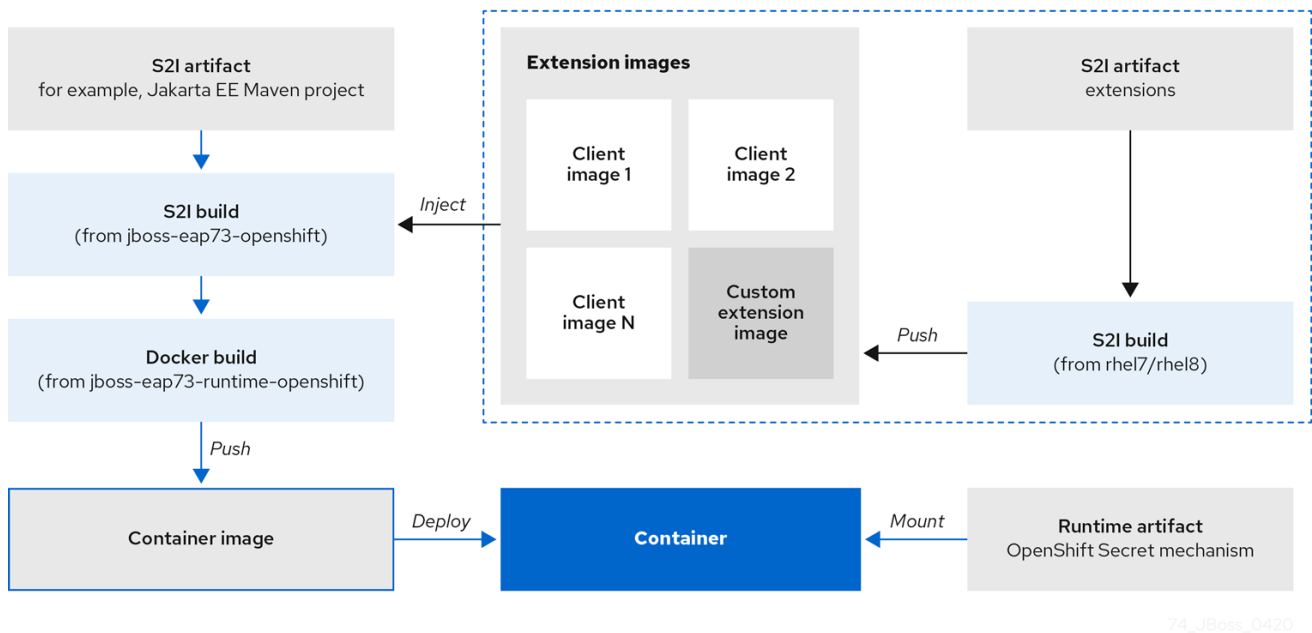
IMPORTANT

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 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.

Adding Datasource Drivers

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

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

Example drivers.env File for MySQL Datasource

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

Example drivers.env File for PostgreSQL Datasource

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

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

Generic Deployments

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

- If the **CUSTOM_INSTALL_DIRECTORIES** environment variable has been declared but no **install.sh** scripts are found in the custom installation directories, the following artifact directories will be copied to their respective destinations in the built image:

  - **modules/*** copied to **$JBOSS_HOME/modules/**

  - **configuration/*** copied to **$JBOSS_HOME/standalone/configuration**

  - **deployments/*** copied to **$JBOSS_HOME/standalone/deployments**

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

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 or in the OpenShift repository. Configuration of these datasources is provided by environment files added to OpenShift Secrets.

2. External datasources. These datasources do not run on OpenShift. Configuration of external datasources is provided by environment files added to OpenShift Secrets.

> **NOTE**
>
> For more details about creating and configuring OpenShift Secrets, see Secrets.

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

Example: Datasource Environment File

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

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

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

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

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

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

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

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
*PREFIX*_ID	The identifier of the resource adapter as specified in the server configuration file.
*PREFIX*_ARCHIVE	The resource adapter archive.
*PREFIX*_MODULE_SLOT	The slot subdirectory, which contains the **module.xml** configuration file and any required JAR files.
*PREFIX*_MODULE_ID	The JBoss Module ID where the object factory Java class can be loaded from.
*PREFIX*_CONNECTION_CLASS	The fully qualified class name of a managed connection factory or admin object.

Attribute	Description
*PREFIX*_CONNECTION_JNDI	The JNDI name for the connection factory.
*PREFIX*_PROPERTY_ParentDirectory	Directory where the data files are stored.
*PREFIX*_PROPERTY_AllowParentPaths	Set **AllowParentPaths** to **false** to disallow **..** in paths. This prevents requesting files that are not contained in the parent directory.
*PREFIX*_POOL_MAX_SIZE	The maximum number of connections for a pool. No more connections will be created in each sub-pool.
*PREFIX*_POOL_MIN_SIZE	The minimum number of connections for a pool.
*PREFIX*_POOL_PREFILL	Specifies if the pool should be prefilled. Changing this value requires a server restart.
*PREFIX*_POOL_FLUSH_STRATEGY	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. RESULTS OF USING JBOSS EAP TEMPLATES FOR OPENSHIFT

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

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

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

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

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

The environmment 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

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

Some quickstarts require this datasource:

- **cmt**

- **thread-racing**

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

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

```
ENABLE_GENERATE_DEFAULT_DATASOURCE=true
```

CHAPTER 4. CAPABILITY TRIMMING IN JBOSS EAP FOR OPENSHIFT

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:

- Jakarta Server Faces 2.3

- Jakarta Enterprise Beans 3.2

- Jakarta XML 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.

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

- **core-management**

- **datasources**

- **deployment-scanner**

- **ee**

- **elytron**

- **io**

- **jca**

- **jmx**

- **logging**

- **naming**

- **request-controller**

- **security-manager**

- **transactions**

- **undertow**

The following Jakarta EE specifications are supported in this layer:

- Jakarta JSON Processing 1.1

- Jakarta JSON Binding 1.0

- Jakarta Servlet 4.0

- Jakarta Expression Language 3.0

- Jakarta Server Pages 2.3

- Jakarta Standard Tag Library 1.2

- Jakarta Concurrency 1.1

- Jakarta Annotations 1.3

- Jakarta XML Binding 2.3

- Jakarta Debugging Support for Other Languages 1.0

- Jakarta Transactions 1.3

- Jakarta Connectors 1.7

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

- **jaxrs**

- **weld**

- **jpa**

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

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

- Jakarta Contexts and Dependency Injection 2.0

- Jakarta Bean Validation 2.0

- Jakarta Interceptors 1.2

- Jakarta RESTful Web Services 2.1

- Jakarta Persistence 2.2

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

- **resource-adapters**

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

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

- Health subsystem

- Metrics subsystem

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

- Jakarta Security 1.0

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:

- Health subsystem

- Metrics subsystem

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

web-clustering

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

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. *Optional:* You can use custom Galleon feature pack environment variables to customize Galleon feature packs and layers during the S2I image build process.
   For more information about custom Galleon feature pack environment variables, see custom Galleon feature pack environment variables.

5. 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

Create your custom layer feature pack as a Maven project.

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

2. Within the Maven project, create your layer content in 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** file with the following content.

     ```xml
     <?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** file with the following content.

  ```xml
  <?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/${env.POSTGRESQL_DATASOURCE,env.OPENSHIFT_P
  OSTGRESQL_DATASOURCE:PostgreSQLDS}"/>
          <param name="connection-url"
  value="jdbc:postgresql://${env.POSTGRESQL_SERVICE_HOST,\
  env.OPENSHIFT_POSTGRESQL_DB_HOST}:${env.POSTGRESQL_SERVICE_PORT,\

  env.OPENSHIFT_POSTGRESQL_DB_PORT}/${env.POSTGRESQL_DATABASE,
  env.OPENSHIFT_POSTGRESQL_DB_NAME}"/>
          <param name="driver-name" value="postgresql"/>
          <param name="user-name" value="${env.POSTGRESQL_USER,
  env.OPENSHIFT_POSTGRESQL_DB_USERNAME}"/>
          <param name="password" value="${env.POSTGRESQL_PASSWORD,
  env.OPENSHIFT_POSTGRESQL_DB_PASSWORD}"/>
          <param name="check-valid-connection-sql" value="SELECT 1"/>
          <param name="background-validation" value="true"/>
          <param name="background-validation-millis" value="60000"/>
          <param name="flush-strategy" value="IdleConnections"/>
          <param name="statistics-enabled" value="${wildfly.datasources.statistics-
  enabled:${wildfly.statistics-enabled:false}}" />
      </feature>
  ```

3. In the **pom.xml** file used to build your custom feature pack, refer to the JBoss EAP dependencies.

   ```xml
   <dependency>
       <groupId>org.jboss.eap</groupId>
       <artifactId>wildfly-ee-galleon-pack</artifactId>    ❶
       <version>7.4.0.GA-redhat-XXXXX</version>    ❷
       <type>zip</type>
   </dependency>
   ```

   ❶ When using the JBoss EAP expansion pack (JBoss EAP XP), the value of this element should be **wildfly-galleon-pack**.

   ❷ Where **XXXXX** is the minor version of your JBoss EAP instance.

   These dependencies are available in the Red Hat Maven repository:
   https://maven.repository.redhat.com/ga/

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

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.
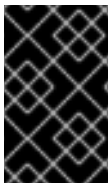
> **IMPORTANT**
>
> Do not create a custom provisioning file when using the **GALLEON_PROVISION_LAYERS** environment variable, because this environment variable configures the S2I build process to ignore the file.

The following code illustrates a custom provisioning file.

```xml
<?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

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.

Additional resources

- For more information about using the **GALLEON_PROVISION_LAYERS** environment variable, see Provision a Custom JBoss EAP server.

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
  <?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

4.3.4. Providing custom Galleon feature pack support for your JBoss EAP S2I image.

You can use custom Galleon feature pack environment variables to customize Galleon feature packs and layers during the S2I image build process. These environment variables expand on the fixed set of Galleon layers provided by the **GALLEON_PROVISION_LAYERS** environment variable.

The example in this procedure shows the use of the **GALLEON_PROVISION_FEATURE_PACKS** environment variable for provisioning a JBoss EAP server with a custom Galleon feature pack on OpenShift.

Procedure

- Enter the following command that uses the **eap74-basic-2si** template to deploy an example quickstart, such as **helloworld-html5**, with a provisioned **postgresql** driver for a server. You must prepend a build environment variable with the **--build-env** argument.

```
oc new-app --template=eap74-basic-s2i \
-p IMAGE_STREAM_NAMESPACE=eap-demo \
-p EAP_IMAGE_NAME=jboss-eap74-openjdk11-openshift:7.4.0 \
-p EAP_RUNTIME_IMAGE_NAME=jboss-eap74-openjdk11-runtime-openshift:7.4.0 \
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \
-p SOURCE_REPOSITORY_REF=7.4.x \
-p GALLEON_PROVISION_LAYERS=jaxrs-server,postgresql-driver \    ❶
-p CONTEXT_DIR=helloworld-html5 \
--build-env POSTGRESQL_DRIVER_VERSION=42.2.19 \    ❷
--build-env GALLEON_PROVISION_FEATURE_PACKS="org.jboss.eap:eap-datasources-galleon-pack:7.4.0.GA-redhat-00003"    ❸
```

❶ Use the **GALLEON_PROVISON_LAYERS** environment variable to specify the **postgresql-driver** Galleon layer to provision a **postgresql** driver for your server.

❷ Use the **POSTGRESQL_DRIVER_VERSION** build environment variable to specify the **postgresql** driver version to retrieve the driver artifact from the remote Maven repository when you provision the **postgresql** driver for your server.

③ Use the **GALLEON_PROVISION_FEATURE_PACKS** build environment variable to specify a custom Galleon feature pack for your S2I image.

> **NOTE**
>
> The **org.jboss.eap:eap-datasources-galleon-pack** feature pack contains Galleon layers that you can use to provision drivers and datasources, such as Microsoft SQL Server, Oracle database, or PostgreSQL, for JDBC drivers and datasources. The feature pack provides support for using the **POSTGRESQL_DRIVER_VERSION** environment variable in your project.

Expected outcome

The S2I process provisions your JBoss EAP server on OpenShift and the server contains the **postgresql** driver.

Additional resources

- For more information about custom Galleon feature pack environment variables, see custom Galleon feature pack environment variables.

- For more information about using the **GALLEON_PROVISION_LAYERS** environment variable, see Provision a Custom JBoss EAP server.

- For information about using an **eap74-basic-2si** template to deploy an example quickstart, see Deploy a JBoss EAP Source-to-Image (S2I) Application to OpenShift .

4.3.5. Configure Galleon by using advanced environment variables

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

- **GALLEON_DIR=<path>**, which overrides the default **<project_root_dir>/galleon** directory path to **<project_root_dir>/<GALLEON_DIR>**.

- **GALLEON_CUSTOM_FEATURE_PACKS_MAVEN_REPO=<path>**, which overrides the **<project root dir>/galleon/repository** directory path with an absolute path to a Maven local repository cache directory. This repository contains custom Galleon feature packs.

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

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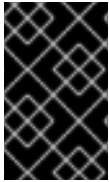
IMPORTANT

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

Additional resources

- For more information about custom Galleon feature pack environment variables, see custom Galleon feature pack environment variables.

4.3.6. Custom Galleon feature pack environment variables

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

Table 4.1. Descriptions of custom Galleon feature pack environment variables

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

CHAPTER 5. TROUBLESHOOTING

5.1. 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.

5.2. 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 6. 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, manage, and seamlessly upgrade instances of complex stateful applications.

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

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

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

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

For more information about operators and the OLM, see the OpenShift documentation.

6.1. 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.
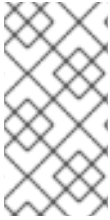
> **NOTE**
>
> The following procedure might change in accordance with the modifications in the OpenShift Container Platform web console. For the latest and most accurate procedure, see the Installing from the OperatorHub using the web console section in the latest version of the *Working with Operators 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.

6.2. INSTALLING EAP OPERATOR USING THE CLI

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

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

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

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

Procedure

1. View the list of operators available to the cluster from the OperatorHub:

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

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

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

   ❶ Name of the operator to subscribe to.

   ❷ The EAP operator is provided by the **redhat-operators** CatalogSource.

   For information about channels and approval strategy, see the web console version of this procedure.

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

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

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

6.3. THE EAP-S2I-BUILD TEMPLATE FOR CREATING APPLICATION IMAGES

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

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

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

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

Additional resources

- Building an application image using **eap-s2i-build** template.

6.4. BUILDING AN APPLICATION IMAGE USING EAP-S2I-BUILD TEMPLATE

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

Procedure

1. Import EAP images in OpenShift. For more information, see Importing the OpenShift image streams and templates for JBoss EAP XP.

2. Configure the imagestream to receive updates about the changes in the application imagestream and to trigger new builds. For more information, see Configuring periodic importing of imagestreamtags.

3. Create the **eap-s2i-build** template for building the application image using EAP S2I images:

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

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

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

```
$ oc process eap-s2i-build \
  -p APPLICATION_IMAGE=my-app \ ❶
  \
  -p EAP_IMAGE=jboss-eap-xp1-openjdk11-openshift:1.0 \ ❷
  -p EAP_RUNTIME_IMAGE=jboss-eap-xp1-openjdk11-runtime-openshift:1.0 \ ❸
  -p EAP_IMAGESTREAM_NAMESPACE=$(oc project -q) \ ❹
  \
  -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-
quickstarts.git \ ❺
  -p SOURCE_REPOSITORY_REF=xp-1.0.x \ ❻
  -p CONTEXT_DIR=microprofile-config | oc create -f - ❼
```

❶ The name for the application imagestream. The application image is tagged with the **latest**
tag.

❷ The imagestreamtag for EAP builder image.

❸ The imagestreamtag for EAP runtime image.

❹ The namespace in which the imagestreams for Red Hat Middleware images are installed. If
omitted, the **openshift** namespace is used. Modify this only if you have installed the
imagestreams in a namespace other than **openshift**.

❺ The Git source URL of your application.

❻ The Git branch or tag reference

❼ The path within the Git repository that contains the application to build.

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

   a. Configure the **WildFlyServer** resource:

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

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

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

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

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

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

Additional resources

- For more information about importing an application imagestream, see Importing the latest OpenShift image streams and templates for JBoss EAP XP.

- For more information about periodic importing of imagestreams, see Configuring periodic importing of imagestreamtags.

6.5. DEPLOYING A JAVA APPLICATION ON OPENSHIFT USING THE EAP OPERATOR

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

Prerequisites

- You have installed EAP operator. For more information about installing the EAP operator, see Installing EAP Operator Using the Webconsole and Installing EAP Operator Using the CLI.

- You have built a Docker image of the user application using JBoss EAP for OpenShift Source-to-Image (S2I) build image.

- The **APPLICATION_IMAGE** parameter in your **eap-s2i-build** template has an imagestream, if you want to enable automatic upgrade of your application after it is deployed on OpenShift. For more information about building your application image using the **eap-s2i-build** template, see Building an application image using eap-s2i-build template .

- You have created a **Secret** object, if your application's CustomResourceDefinition (CRD) file references one. For more information about creating a new **Secret** object, see Creating a Secret.

- You have created a **ConfigMap**, if your application's CRD file references one. For information about creating a **ConfigMap**, see Creating a ConfigMap .

- You have created a **ConfigMap** from the **standalone.xml** file, if you choose to do so. For information about creating a **ConfigMap** from the **standalone.xml** file, see Creating a ConfigMap from a standalone.xml File.

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

Procedure

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

2. Select the **Project** or namespace you want to use for your Java application.

3. Navigate to **Installed Operator** and select **JBoss EAP operator**.

4. On the **Overview** tab, click the **Create Instance** link.

5. Specify the application image details.
   The application image specifies the Docker image that contains the Java application. The image must be built using the JBoss EAP for OpenShift Source-to-Image (S2I) build image. If the **applicationImage** field corresponds to an imagestreamtag, any change to the image triggers an automatic upgrade of the application.

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

   - The name of the image: mycomp/myapp

   - A tag: mycomp/myapp:1.0

   - A digest:
     mycomp/myapp:@sha256:0af38bc38be93116b6a1d86a9c78bd14cd527121970899d719baf78e

   - An imagestreamtag: my-app:latest

6. Specify the size of the application. For example:

   ```
   spec:
     replicas:2
   ```

7. Configure the application environment using the **env spec**. The environment variables can come directly from values, such as POSTGRESQL_SERVICE_HOST or from **Secret** objects, such as POSTGRESQL_USER. For example:

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

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

- Specify the storage requirements for the server data directory. For more information, see Configuring Persistent Storage for Applications.

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

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

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

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

6.5.1. Creating a Secret

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

Procedure

- To create a **Secret**:

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

6.5.2. Creating a ConfigMap

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

Procedure

- To create a configmap:

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

6.5.3. Creating a ConfigMap from a standalone.xml File

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

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

Procedure

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

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

6.5.4. Configuring Persistent Storage for Applications

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

Procedure

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.

6.6. VIEWING METRICS OF AN APPLICATION USING THE EAP OPERATOR

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

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

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console, navigate to **Monitoring→ Metrics**.

2. On the **Metrics** screen, type the name of your application in the text box to select your application. The metrics for your application appear on the screen.

> **NOTE**
>
> All metrics related to JBoss EAP application server are prefixed with **jboss**. For example, **jboss_undertow_request_count_total**.

6.7. UNINSTALLING EAP OPERATOR USING WEB CONSOLE

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

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

You can uninstall the EAP operator using the web console.

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

Procedure

1. From the **Operators→ Installed Operators** page, select **JBoss EAP**.

2. On the right-hand side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu.

3. When prompted by the **Remove Operator Subscription** window, optionally select the **Also completely remove the Operator from the selected namespace** check box if you want all components related to the installation to be removed. This removes the CSV, which in turn removes the pods, deployments, custom resource definitions (CRDs), and custom resources (CRs) associated with the operator.

4. Click **Remove**. The EAP operator stops running and no longer receives updates.

6.8. UNINSTALLING EAP OPERATOR USING THE CLI

To delete, or uninstall, the EAP operator from your cluster, you can delete the subscription to remove it from the subscribed namespace. You can also remove the EAP operator's ClusterServiceVersion (CSV) and deployment.
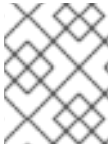
> **NOTE**
>
> To ensure data consistency and safety, scale down the number of pods in your cluster to 0 before uninstalling the EAP operator.

You can uninstall the EAP operator using the command line.

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

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

Procedure

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

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

2. Delete the EAP operator's subscription:

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

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

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

6.9. EAP OPERATOR FOR SAFE TRANSACTION RECOVERY

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

> **NOTE**
>
> Some scenarios are not supported. For more information about the unsupported scenarios, see Unsupported Transaction Recovery Scenarios .

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

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

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

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

6.9.1. 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.

6.9.2. 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.

6.9.2.1. Pod Status During Scaledown

The following table describes the different pod statuses during scaledown:

Table 6.1. 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.

6.9.3. Scaling Down During Transactions with Heuristic Outcomes

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

Prerequisites

- The status of your pod is stuck at **SCALING_DOWN_RECOVERY_DIRTY**.

Procedure

1. Access your JBoss EAP instance using CLI.

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

3. Remove all records from the enterprise bean client recovery folder.

   a. Remove all files from the pod enterprise bean client recovery directory:

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

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

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

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

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

The JDBC object store can be set up with the environment variable **TX_DATABASE_PREFIX_MAPPING**. This variable has the same structure as **DB_SERVICE_PREFIX_MAPPING**.

Prerequisite

- You have created a datasource based on the value of the environment variables.

- You have ensured consistent data reads and writes permissions exist between the database and the **transaction manager** communicating over the JDBC object store. For more information see [configuring JDBC data sources](#)

Procedure

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

  Example

  ```
  # Narayana JDBC objectstore configuration via s2i env variables
  - name: TX_DATABASE_PREFIX_MAPPING
    value: 'PostgresJdbcObjectStore-postgresql=PG_OBJECTSTORE'
  - name: POSTGRESJDBCOBJECTSTORE_POSTGRESQL_SERVICE_HOST
  ```

```
      value: 'postgresql'
    - name: POSTGRESJDBCOBJECTSTORE_POSTGRESQL_SERVICE_PORT
      value: '5432'
    - name: PG_OBJECTSTORE_JNDI
      value: 'java:jboss/datasources/PostgresJdbc'
    - name: PG_OBJECTSTORE_DRIVER
      value: 'postgresql'
    - name: PG_OBJECTSTORE_DATABASE
      value: 'sampledb'
    - name: PG_OBJECTSTORE_USERNAME
      value: 'admin'
    - name: PG_OBJECTSTORE_PASSWORD
      value: 'admin'
```

Verification

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

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

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

Additional resources

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

6.10. AUTOMATICALLY SCALING PODS WITH THE HORIZONTAL POD AUTOSCALER HPA

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

NOTE

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

Procedure

1. Configure the resources:

   ```
   apiVersion: wildfly.org/v1alpha1
   kind: WildFlyServer
   metadata:
     name: eap-helloworld
   spec:
     applicationImage: 'eap-helloworld:latest'
     replicas: 1
     resources:
       limits:
         cpu: 500m
         memory: 2Gi
       requests:
         cpu: 100m
         memory: 1Gi
   ```

   IMPORTANT

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

2. Create the Horizontal pod autoscaler:

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

Verification

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

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

Additional resources

- https://access.redhat.com/documentation/en-us/openshift_container_platform/4.10/html-single/nodes/index#nodes-pods-autoscaling

6.11. JAKARTA ENTERPRISE BEANS REMOTING ON OPENSHIFT

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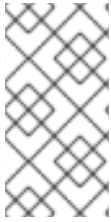
NOTE

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

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

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

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

- Define a remote outbound connection. For more information, see Configuring a Remote Outbound Connection.

- Use a programmatic JNDI lookup for the bean at the remote server. For more information, see Using Remote Jakarta Enterprise Beans Clients .

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

NOTE

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

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

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

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

6.11.1. Configuring Jakarta Enterprise Beans on OpenShift

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

Prerequisites

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

- The clustering is set correctly. For more information about JBoss EAP clustering, see the Clustering section.

Procedure

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

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

2. Configure the caller JBoss EAP application server.

   a. Create the **eap-config.xml** file in **$JBOSS_HOME/standalone/configuration** using the custom configuration functionality. For more information, see Custom Configuration.

   b. Configure the caller JBoss EAP application server with the **wildfly.config.url** property:

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

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

   Example Configuration

   ```xml
   <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 7. REFERENCE INFORMATION
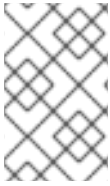
> **NOTE**
>
> The content in this section is derived from the engineering documentation for this image. It is provided for reference as it can be useful for development purposes and for testing beyond the scope of the product documentation.

7.1. 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.

7.2. INFORMATION ENVIRONMENT VARIABLES

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

Table 7.1. Information Environment Variables

Variable Name	Description and Value
JBOSS_IMAGE_NAME	The image names. Values: • **jboss-eap-7/eap74-openjdk8-openshift-rhel7** (JDK 8 / RHEL 7) • **jboss-eap-7/eap74-openjdk11-openshift-rhel8** (JDK 11 / RHEL 8)

Variable Name	Description and Value
JBOSS_IMAGE_VERSION	The image version. Value: This is the image version number. See the Red Hat Container Catalog for the latest values: • JDK 8 / RHEL 7 • JDK 11 / RHEL 8
JBOSS_MODULES_SYSTEM_PKGS	A comma-separated list of JBoss EAP system modules packages that are available to applications. Value: **jdk.nashorn.api**
STI_BUILDER	Provides OpenShift S2I support for **jee** project types. Value: **jee**

7.3. CONFIGURATION ENVIRONMENT VARIABLES

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

> **NOTE**
>
> See the JBoss EAP documentation for other environment variables that are not listed here.

Table 7.2. Configuration environment variables

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

Variable Name	Description
AB_JOLOKIA_PASSWORD_RANDOM	Determines if a random **AB_JOLOKIA_PASSWORD** should be generated. Set to **true** to generate a random password. The generated value is saved in the **/opt/jolokia/etc/jolokia.pw** file.
AB_JOLOKIA_PORT	The port to listen to. Defaults to **8778**. Example value: **5432**
AB_JOLOKIA_USER	The name of the user to use for basic authentication. Defaults to **jolokia**. Example value: **myusername**
AB_PROMETHEUS_ENABLE	If set to **true**, this variable activates the **jmx-exporter** java agent that exposes Prometheus format metrics. Default is set to **false**. **NOTE** The MicroProfile Metrics subsystem is the preferred method to expose data in the Prometheus format. For more information about the MicroProfile Metrics susbsystem, see Eclipse MicroProfile in the *Configuration Guide* for JBoss EAP.
AB_PROMETHEUS_JMX_EXPORTER_CONFIG	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.
AB_PROMETHEUS_JMX_EXPORTER_PORT	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** API for the application to include the service exposing this endpoint.
CLI_GRACEFUL_SHUTDOWN	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. Example value: **true**

Variable Name	Description
CONTAINER_HEAP_PERCENT	Set the maximum Java heap size, as a percentage of available container memory. Example value: **0.5**
CUSTOM_INSTALL_DIRECTORIES	A list of comma-separated directories used for installation and configuration of artifacts for the image during the S2I process. Example value: **custom,shared**
DEFAULT_JMS_CONNECTION_FACTORY	This value is used to specify the default JNDI binding for the Jakarta Messaging connection factory, for example **jms-connection-factory='java:jboss/DefaultJMSConnectionFactory'**. Example value: **java:jboss/DefaultJMSConnectionFactory**
DISABLE_EMBEDDED_JMS_BROKER	The use of an embedded messaging broker in OpenShift containers is deprecated. Support for an embedded broker will be removed in a future release. If the following conditions are true, a warning is logged. • A container is configured to use an embedded messaging broker. • A remote broker is not configured for the container. • This variable is not set or is set with a value of **false**. If this variable is included with the value set to **true**, the embedded messaging broker is disabled, and no warning is logged. Include this variable set to **true** for any container that is not configured with remote messaging destinations.
ENABLE_ACCESS_LOG	Enable logging of access messages to the standard output channel. Logging of access messages is implemented using following methods: • 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**. Defaults to **false**.

Variable Name	Description
INITIAL_HEAP_PERCENT	Set the initial Java heap size, as a percentage of the maximum heap size. Example value: **0.5**
JAVA_OPTS_APPEND	Server startup options. Example value: **-Dfoo=bar**
JBOSS_MODULES_SYSTEM_PKGS_APPEND	A comma-separated list of package names that will be appended to the **JBOSS_MODULES_SYSTEM_PKGS** environment variable. Example value: **org.jboss.byteman**
JGROUPS_CLUSTER_PASSWORD	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. Example value: **mypassword**
JGROUPS_ENCRYPT_KEYSTORE	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. Example value: **jgroups.jceks**
JGROUPS_ENCRYPT_KEYSTORE_DIR	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. Example value: **/etc/jgroups-encrypt-secret-volume**
JGROUPS_ENCRYPT_NAME	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. Example value: **jgroups**

Variable Name	Description
JGROUPS_ENCRYPT_PASSWORD	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. Example value: **mypassword**
JGROUPS_ENCRYPT_PROTOCOL	JGroups protocol to use for encryption of cluster traffic. Can be either **SYM_ENCRYPT** or **ASYM_ENCRYPT**. Defaults to **SYM_ENCRYPT**. Example value: **ASYM_ENCRYPT**
JGROUPS_ENCRYPT_SECRET	Name of the secret that contains 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 **dns.DNS_PING** or **kubernetes.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**.
OPENSHIFT_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**
OPENSHIFT_DNS_PING_SERVICE_PORT	The port number of the ping port for the DNS discovery mechanism. If not specified, an attempt is made to discover the port number from the SRV records for the service, otherwise the default **8888** is used. Defaults to **8888**.
OPENSHIFT_KUBE_PING_LABELS	Clustering labels selector for the Kubernetes discovery mechanism. Example value: **app=eap-app**

Variable Name	Description
OPENSHIFT_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.

7.4. APPLICATION TEMPLATES

Table 7.3. Application Templates

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

7.5. EXPOSED PORTS

Table 7.4. Exposed Ports

Port Number	Description
8443	HTTPS
8778	Jolokia Monitoring

7.6. 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.

7.6.1. JNDI Mappings for Datasources

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

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

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

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

> **WARNING**
>
> Do not use any special characters for the **_POOLNAME_** parameter.
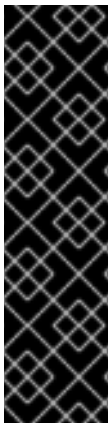
> **DATABASE DRIVERS**
>
> Support for using the Red Hat-provided internal datasource drivers with the JBoss EAP for OpenShift image is now deprecated. Red Hat recommends that you use JDBC drivers obtained from your database vendor for your JBoss EAP applications.
>
> The following internal datasources are no longer provided with the JBoss EAP for OpenShift image:
>
> - MySQL
>
> - PostgreSQL

For more information about installing drivers, see Modules, Drivers, and Generic Deployments.

For more information on configuring JDBC drivers with JBoss EAP, see JDBC drivers in the JBoss EAP *Configuration Guide*.

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

7.6.1.1. Datasource Configuration Environment Variables

To configure other datasource properties, use the following environment variables.
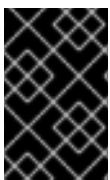
> **IMPORTANT**
>
> Be sure to replace the values for **_POOLNAME_**, **_DATABASETYPE_**, and **_PREFIX_** in the following variable names with the appropriate values. These replaceable values are described in this section and in the Datasources section.

Variable Name	Description
*POOLNAME_DATABASETYPE*_SERVICE _HOST	Defines the database server's host name or IP address to be used in the datasource's **connection-url** property. Example value: **192.168.1.3**
*POOLNAME_DATABASETYPE*_SERVICE _PORT	Defines the database server's port for the datasource. Example value: **5432**
*PREFIX*_BACKGROUND_VALIDATION	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.
*PREFIX*_BACKGROUND_VALIDATION_M ILLIS	Specifies frequency of the validation, in milliseconds, when the **background-validation** database connection validation mechanism is enabled (*PREFIX*_**BACKGROUND_VALIDATION** variable is set to **true**). Defaults to **10000**.
*PREFIX*_CONNECTION_CHECKER	Specifies a connection checker class that is used to validate connections for the particular database in use. Example value: **org.jboss.jca.adapters.jdbc.extensions.postgres.Postg reSQLValidConnectionChecker**
*PREFIX*_DATABASE	Defines the database name for the datasource. Example value: **myDatabase**
*PREFIX*_DRIVER	Defines Java database driver for the datasource. Example value: **postgresql**
*PREFIX*_EXCEPTION_SORTER	Specifies the exception sorter class that is used to properly detect and clean up after fatal database connection exceptions. Example value: **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLE xceptionSorter**
*PREFIX*_JNDI	Defines the JNDI name for the datasource. Defaults to **java:jboss/datasources/***POOLNAME_DATABASETYPE*, where *POOLNAME* and *DATABASETYPE* are taken from the triplet described above. This setting is useful if you want to override the default generated JNDI name. Example value: **java:jboss/datasources/test-postgresql**

Variable Name	Description
*PREFIX*_JTA	Defines Jakarta Transactions option for the non-XA datasource. The XA datasources are already Jakarta Transactions capable by default. Defaults to **true**.
*PREFIX*_MAX_POOL_SIZE	Defines the maximum pool size option for the datasource. Example value: **20**
*PREFIX*_MIN_POOL_SIZE	Defines the minimum pool size option for the datasource. Example value: **1**
*PREFIX*_NONXA	Defines the datasource as a non-XA datasource. Defaults to **false**.
*PREFIX*_PASSWORD	Defines the password for the datasource. Example value: **password**
*PREFIX*_TX_ISOLATION	Defines the java.sql.Connection transaction isolation level for the datasource. Example value: **TRANSACTION_READ_UNCOMMITTED**
*PREFIX*_URL	Defines connection URL for the datasource. Example value: **jdbc:postgresql://localhost:5432/postgresdb**
*PREFIX*_USERNAME	Defines the username for the datasource. Example value: **admin**

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.

7.6.1.2. Examples

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

7.6.1.2.1. Single Mapping

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

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

7.6.1.2.2. Multiple Mappings

You can specify multiple datasource mappings.

NOTE

Always separate multiple datasource mappings with a comma.

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

This creates the following two datasources:

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

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

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

7.7. CLUSTERING

7.7.1. Configuring a JGroups Discovery Mechanism

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

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

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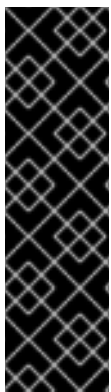
IMPORTANT

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

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

7.7.1.1. 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.

7.7.1.2. Configuring DNS_PING

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

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

   ```
   JGROUPS_PING_PROTOCOL=dns.DNS_PING
   ```

2. The **OPENSHIFT_DNS_PING_SERVICE_NAME** environment variable must be set to the name of the ping service for the cluster.

   ```
   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 the **service.alpha.kubernetes.io/tolerate-unready-endpoints** and the **publishNotReadyAddresses** properties, both set to **true**.

   NOTE

   - Use both the **service.alpha.kubernetes.io/tolerate-unready-endpoints** and the **publishNotReadyAddresses** properties to ensure that the ping service works in both the older and newer OpenShift releases.

   - Omitting these annotations result in each node forming its own "cluster of one" during startup. Each node then merges its cluster into the other nodes' clusters after startup, because the other nodes are not detected until after they have started.

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

NOTE

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

7.7.2. 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.
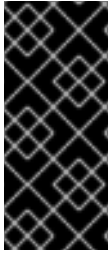
> **IMPORTANT**
>
> The **SYM_ENCRYPT** and **ASYM_ENCRYPT** protocols are not compatible with each other. It is not possible to form a supercluster out of two independent child clusters, with one using the **SYM_ENCRYPT** protocol for the encryption of cluster traffic and the other using the **ASYM_ENCRYPT** protocol. Similarly, when performing a rolling upgrade, the protocol needs to be identical for both the source and the target clusters.

7.7.2.1. Configuring SYM_ENCRYPT

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

1. The JGroups protocol stack must be configured to use **SYM_ENCRYPT** as the encryption protocol.
   You can do this by setting the **JGROUPS_ENCRYPT_PROTOCOL** environment variable to **SYM_ENCRYPT**:

   ```
   JGROUPS_ENCRYPT_PROTOCOL=SYM_ENCRYPT
   ```

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

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

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

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

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

   ```
   JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
   ```

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

   ```
   JGROUPS_ENCRYPT_NAME=jgroups
   ```

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

   ```
   JGROUPS_ENCRYPT_PASSWORD=mypassword
   ```

7.7.2.2. Configuring ASYM_ENCRYPT

> **NOTE**
>
> JBoss EAP 7.4 includes a new version of the **ASYM_ENCRYPT** protocol. The previous version of the protocol is deprecated. If you specify the **JGROUPS_CLUSTER_PASSWORD** environment variable, the deprecated version of the protocol is used and a warning is printed in the pod log.

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

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

7.7.3. Considerations for scaling up pods

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

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

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

7.8. 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 7.5. Liveness and Readiness Checks

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

Performed Test	Liveness	Readiness
Deployment Status [a]	N/A or no **failed** entries	N/A or no **failed** entries
Eclipse MicroProfile Health [b]	N/A or **UP**	N/A or **UP**

[a] N/A is only a valid state when no deployments are present.

[b] N/A is only a valid state when the **microprofile-health-smallrye** subsystem has been disabled.

7.9. MESSAGING

7.9.1. Configuring External Red Hat AMQ Brokers

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

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

Example: JDK 8

```
oc new-app eap74-amq-s2i \
-p EAP_IMAGE_NAME=jboss-eap74-openjdk8-openshift:7.4.0 \
-p EAP_RUNTIME_IMAGE_NAME=jboss-eap74-openjdk8-runtime-openshift:7.4.0 \
-p APPLICATION_NAME=eap74-mq \
-p MQ_USERNAME=MY_USERNAME \
-p MQ_PASSWORD=MY_PASSWORD
```
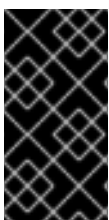
> **IMPORTANT**
>
> The template used in this example provides valid default values for the required parameters. If you do not use a template and provide your own parameters, be aware that the **MQ_SERVICE_PREFIX_MAPPING** name must match the **APPLICATION_NAME** name, appended with "-amq7=MQ".

7.10. 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 7.6. 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**

7.11. 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**

7.12. ADMINISTRATION ENVIRONMENT VARIABLES

Table 7.7. Administration Environment Variables

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

7.13. 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.

7.13.1. 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.

7.13.1.1. 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.

7.13.2. 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**

7.13.3. 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.

7.13.3.1. 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**.

7.13.4. 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.

7.13.5. 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

7.13.5.1. Mounting a configmap to execute custom scripts

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

To mount a configmap:

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

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

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

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

Example **postconfigure.sh**

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

Example **extensions.cli**

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

7.13.5.2. 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
```

7.13.6. 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 7.8. 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 7.13.3.1, "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.

7.14. SINGLE SIGN-ON IMAGE

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

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

Table 7.9. Single Sign-On environment variables

Variable name	Description
SSO_URL	URL of the Single Sign-On server.
SSO_REALM	Single Sign-On realm for the deployed applications.
SSO_PUBLIC_KEY	Public key of the Single Sign-On realm. This field is optional but if omitted can leave the applications vulnerable to man-in-middle attacks.
SSO_USERNAME	Single Sign-On user required to access the Single Sign-On REST API. Example value: **mySsoUser**
SSO_PASSWORD	Password for the Single Sign-On user defined by the **SSO_USERNAME** variable. Example value: **6fedmL3P**
SSO_SAML_KEYSTORE	Keystore location for SAML. Defaults to **/etc/sso-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	Single Sign-On client access type. (Optional) Example value: **true**
SSO_CLIENT	Path for Single Sign-On redirects back to the application. Defaults to match **module-name**.
SSO_ENABLE_CORS	If **true**, enable CORS for Single Sign-On applications. (Optional)
SSO_SECRET	The Single Sign-On 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 Single Sign-On server is unsecure, for example, the certificate validation is disabled with **curl**. Not set by default. Example value: **true**

7.15. UNSUPPORTED TRANSACTION RECOVERY SCENARIOS

- JTS transactions are not supported in OpenShift.

- XTS transactions are not supported in OpenShift.

- The XATerminator interface that some third parties use for transaction completion and crash recovery flows is not supported in OpenShift.

- Transactions propagated over JBoss Remoting is unsupported with OpenShift 3.

> **NOTE**
>
> Transactions propagated over JBoss Remoting is supported with OpenShift 4 and the EAP operator.

7.16. INCLUDED JBOSS MODULES

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

Table 7.10. Included JBoss Modules

JBoss Module
org.jboss.as.clustering.common
org.jboss.as.clustering.jgroups
org.jboss.as.ee

JBoss Module
org.jgroups
org.openshift.ping
net.oauth.core

7.17. EAP OPERATOR: API INFORMATION

The EAP operator introduces the following APIs:

7.17.1. WildFlyServer

WildFlyServer defines a custom JBoss EAP resource.

Table 7.11. WildFlyServer

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

7.17.2. WildFlyServerList

WildFlyServerList defines a list of JBoss EAP deployments.

Table 7.12. Table

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

7.17.3. 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 7.13. 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
standaloneConfigMap	Spec to specify how a standalone configuration can be read from a **ConfigMap**.	**StandaloneConfigMapSpec**	false
resources	**Resources** spec to specify the request or limits of the Stateful Set. If omitted, the namespace defaults are used.	**Resources**	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

Field	Description	Scheme	Required
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/<co nfig map name>**	string	false
disableHTTPRoute	Disable the creation a route to the HTTP port of the application service (false if omitted)	boolean	false
sessionAffinity	If connections from the same client IP are passed to the same JBoss EAP instance/pod each time (false if omitted)	boolean	false

7.17.4. Resources

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

7.17.5. StorageSpec

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

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

Table 7.14. Table

Field	Description	Scheme	Required
emptyDir	**EmptyDirVolumeSource** to be used by the JBoss EAP **StatefulSet**	corev1.EmptyDirVolume Source	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.PersistentVolume Claim	false

7.17.6. StandaloneConfigMapSpec

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

Table 7.15. 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

7.17.7. WildFlyServerStatus

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

Table 7.16. WildFlyServerStatus

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

7.17.8. PodStatus

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

Table 7.17. PodStatus

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

Revised on 2022-06-27 13:29:56 UTC