# Red Hat JBoss Enterprise Application Platform 7.2

## Getting Started with JBoss EAP for OpenShift Container Platform

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

Last Updated: 2019-09-12

# Red Hat JBoss Enterprise Application Platform 7.2 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

# CHAPTER 1. INTRODUCTION

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

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

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

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

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

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

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

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

| JBoss EAP Feature | Status in JBoss EAP for OpenShift | Description |
|---|---|---|
| JBoss EAP management console | Not included | The JBoss EAP management console is not included in this release of JBoss EAP for OpenShift. |
| JBoss EAP management CLI | Not recommended | The JBoss EAP management CLI is not recommended for use with JBoss EAP running in a containerized environment. Any configuration changes made using the management CLI in a running container will be lost when the container restarts. The management CLI is accessible from within a pod for troubleshooting purposes. |
| Managed domain | Not supported | Although a JBoss EAP managed domain is not supported, creation and distribution of applications are managed in the containers on OpenShift. |

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

## 1.4. VERSION COMPATIBILITY AND SUPPORT

This guide covers the following JBoss EAP for OpenShift images:

- **jboss-eap-7/eap72-openshift**(JDK 8)
- **jboss-eap-7/eap72-openjdk11-openshift-rhel8**(JDK 11)

You can see information on the latest tag for these images in the Red Hat Container Catalog:

- JDK 8
- JDK 11

The Hawkular agent is not active in JDK 11, and will be ignored if configured.

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

### 1.4.1. OpenShift 4.1 Support

Changes in OpenShift 4.1 affect access to Jolokia, and the Open Java Console is no longer available in the OpenShift 4.1 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 is not supported. As a result, accessing Jolokia in this manner is no longer supported.

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

## 1.5. TECHNOLOGY PREVIEW FEATURES

**Automated Transaction Recovery**

IMPORTANT

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

When a cluster is scaled down, it is possible for transaction branches to be in doubt. The JBoss EAP for OpenShift image has an automated transaction recovery feature that can complete these branches. At the moment, this implementation of automated transaction recovery is provided as technology preview only.

The **eap72-tx-recovery-s2i** application template that is provided to demonstrate automatic transaction recovery on scale down of application pods is also provided as a technology preview only.

For JDK 11 image streams, the application template is **eap72-openjdk11-tx-recovery-s2i**.

# 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 Java EE web-enabled database application using JSF, CDI, EJB, JPA, and Bean Validation. See the **kitchensink** quickstart that ships with JBoss EAP 7 for more information.

## 2.1. PREREQUISITES

This workflow assumes that you already have an OpenShift instance installed and operational, similar to that created in the *OpenShift Primer*.

## 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 **eap72-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 using the following command.

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

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

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

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

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

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

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

## 2.3. 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 below 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
```

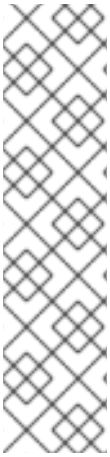
See the OpenShift documentation for more information on other methods for configuring access to secured registries.

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 IMAGE STREAMS AND TEMPLATES

Use the following command to import the latest JBoss EAP for OpenShift image streams and templates into your OpenShift project's namespace.

```
for resource in \
  eap72-image-stream.json \
  eap72-amq-persistent-s2i.json \
  eap72-amq-s2i.json \
  eap72-basic-s2i.json \
  eap72-https-s2i.json \
  eap72-mongodb-persistent-s2i.json \
  eap72-mongodb-s2i.json \
  eap72-mysql-persistent-s2i.json \
  eap72-mysql-s2i.json \
  eap72-postgresql-persistent-s2i.json \
  eap72-postgresql-s2i.json \
  eap72-third-party-db-s2i.json \
  eap72-tx-recovery-s2i.json \
  eap72-sso-s2i.json
do
  oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-7-openshift-image/eap72/templates/${resource}
done
```



NOTE

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

If you have administrative access to the general **openshift** namespace and want the image streams 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 --force -f \
...
```

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

1. Create a new OpenShift application using the JBoss EAP for OpenShift image and your Java application's source code. Red Hat recommends using one of the provided JBoss EAP for OpenShift templates for S2I builds.
   For example, for the **kitchensink** quickstart, use the following command to use the **eap72-basic-s2i** template in the **eap-demo** project, created in Prepare OpenShift for Application Deployment, with the **kitchensink** source code on GitHub.

```
oc new-app --template=eap72-basic-s2i \ 1
 -p IMAGE_STREAM_NAMESPACE=eap-demo \ 2
 -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts
\ 3
 -p SOURCE_REPOSITORY_REF=openshift \ 4
 -p CONTEXT_DIR=kitchensink 5
```

1. The template to use.
2. The latest images streams and templates were imported into the project's namespace, so you must specify the namespace of where to find the image stream. This is usually the project's name.
3. URL to the repository containing the application source code.
4. The Git repository reference to use for the source code. This can be a Git branch or tag reference.
5. The directory within the source repository to build.

**NOTE**

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***.

**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 **eap72-https-s2i**, you must specify the required HTTPS environment variables **HTTPS_NAME**, **HTTPS_PASSWORD**, and **HTTPS_KEYSTORE** to match your keystore details.

2. Retrieve the name of the build configuration.

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

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

-

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

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

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

## 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 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
```

# 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

NOTE

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

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

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

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

   NOTE

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

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

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

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

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

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

## 3.2. CONFIGURING JBOSS EAP FOR 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:

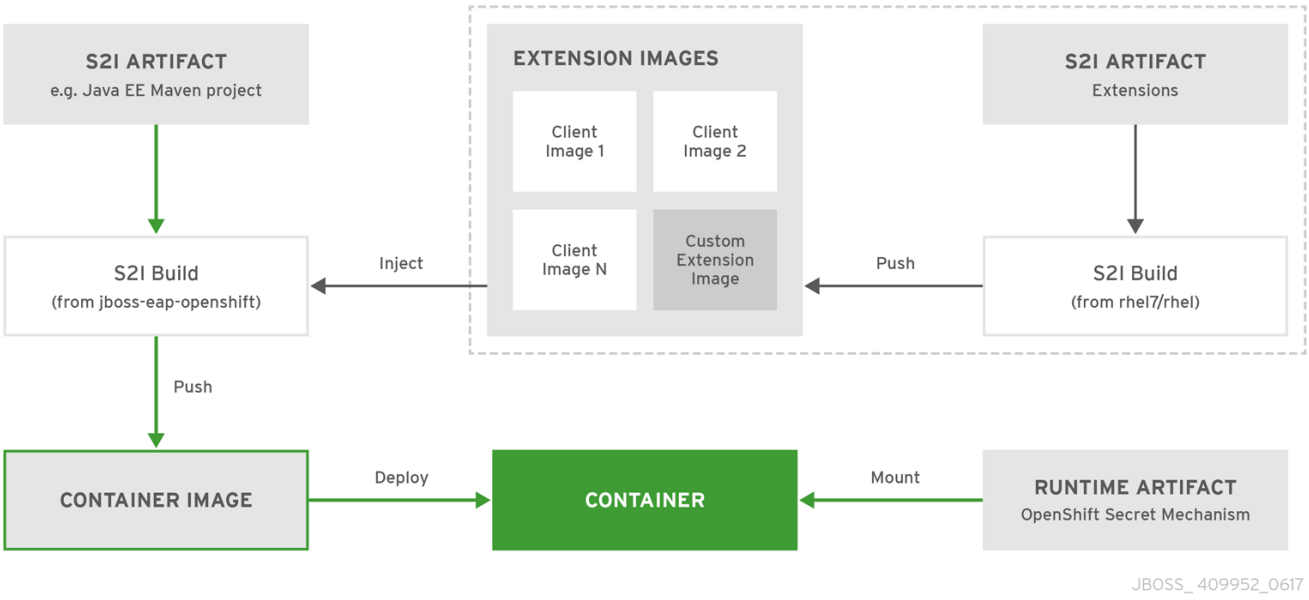
```
oc new-app --template=eap72-basic-s2i \
 -p IMAGE_STREAM_NAMESPACE=eap-demo \
 -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \
 -p SOURCE_REPOSITORY_REF=openshift \
 -p CONTEXT_DIR=kitchensink \
 -e ADMIN_USERNAME=myspecialuser \
 -e ADMIN_PASSWORD=myspecialp@ssw0rd
```

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

## 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 for JDK 8 image streams. It is recommended 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 JDK 11 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*.

### 3.3.1. S2I Artifacts

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

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

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

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

1. Include the artifact in the application source deployment directory. The artifact is downloaded during the build and injected into the image. This is similar to deploying an application on the JBoss EAP for OpenShift image.

2. Include the **CUSTOM_INSTALL_DIRECTORIES** environment variable, a list of comma-separated list of directories used for installation and configuration of artifacts for the image during the S2I process. There are two methods for including this information in the S2I:

   - An **install.sh** script in the nominated installation directory. The install script executes during the S2I process and operates with impunity.

     **install.sh Script Example**

     ```
     #!/bin/bash

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

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

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

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

     **Modules**

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

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

     **Example module.xml File**

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

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

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

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

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

**NOTE**

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

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

### Drivers

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

### Example **drivers.env** File

```
#DRIVER
DRIVERS=DERBY
DERBY_DRIVER_NAME=derby
DERBY_DRIVER_MODULE=org.apache.derby
DERBY_DRIVER_CLASS=org.apache.derby.jdbc.EmbeddedDriver
DERBY_XA_DATASOURCE_CLASS=org.apache.derby.jdbc.EmbeddedXADataSourc
e
```

The MySQL and PostgreSQL datasources are no longer provided as pre-configured internal datasources. However, these drivers can still be installed as modules as described in Modules, Drivers, and Generic Deployments.

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

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

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

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

```
#DRIVER
DRIVERS=POSTGRES
POSTGRES_DRIVER_NAME=postgres
POSTGRES_DRIVER_MODULE=org.postgresql
POSTGRES_DRIVER_CLASS=org.postgresql.Driver
POSTGRES_XA_DATASOURCE_CLASS=org.postgresql.xa.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 **$JBOSS_HOME/modules/system/layers/openshift**
  - **configuration/*** copied to **$JBOSS_HOME/standalone/configuration**
  - **deployments/*** copied to **$JBOSS_HOME/standalone/deployments**

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

## 3.3.2. Runtime Artifacts

### 3.3.2.1. Datasources

There are three types of datasources:

1. Default internal datasources. These are PostgreSQL, MySQL, and MongoDB. These datasources are available on OpenShift by default through the Red Hat Registry and do not require additional environment files to be configured for JDK 8 image streams. Set the DB_SERVICE_PREFIX_MAPPING environment variable to the name of the OpenShift service for the database to be discovered and used as a datasource.

2. Other internal datasources. These are datasources not available by default through the Red Hat Registry but run on OpenShift. Configuration of these datasources is provided by environment files added to OpenShift Secrets.

3. External datasources that are not run on OpenShift. Configuration of external datasources is provided by environment files added to OpenShift Secrets.

#### Example: Datasource Environment File

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

# Connection info for xa datasource
ACCOUNTS_DERBY_XA_CONNECTION_PROPERTY_DatabaseName=/home/jboss/source/data/dat
abases/derby/accounts

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

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

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

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

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

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

### 3.3.2.2. Resource Adapters

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

**Table 3.1. 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. |
| *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. DEPLOYMENT CONSIDERATIONS FOR THE JBOSS EAP FOR OPENSHIFT IMAGE

### 3.4.1. Scaling Up and Persistent Storage Partitioning

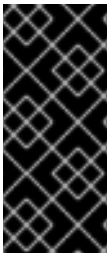
There are two methods for deploying JBoss EAP with persistent storage: single-node partitioning, and multi-node partitioning.

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

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

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

This is now the default setting in the **eap72-tx-recovery-s2i** template.



IMPORTANT

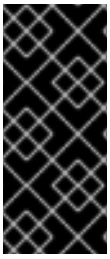
Due to the different storage methods of single-node and multi-node partitioning, changing a deployment from single-node to multi-node results in the application losing all data previously stored in the data directory, including messages, transaction logs, and so on. This is also true if changing a deployment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

### 3.4.2. Scaling Down and Transaction Recovery

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

In order to prevent transactions from remaining within the data store of the terminating pod until the cluster next scales up, the **eap72-tx-recovery-s2i** JBoss EAP template creates a second deployment

containing a migration pod that is responsible for managing the migration of transactions. The migration pod scans each independent **split-*n*** directory within the JBoss EAP persistent volume, identifies data stores associated with the pods that are terminating, and continues to run until all transactions on the terminating pod are completed.



IMPORTANT

Since the persistent volume needs to be accessed in read-write mode by both the JBoss EAP application pod and the migration pod, it needs to be created with the **ReadWriteMany** access mode. This access mode is currently only supported for persistent volumes using **GlusterFS** and **NFS** plug-ins. For details, see the *Supported Access Modes for Persistent Volumes* table.

For more information, see Example Workflow: Automated Transaction Recovery Feature When Scaling Down a Cluster, which demonstrates the automated transaction recovery feature of the JBoss EAP for OpenShift image when scaling down a cluster.

# CHAPTER 4. MIGRATING TO JBOSS EAP FOR OPENSHIFT JDK 11 IMAGE

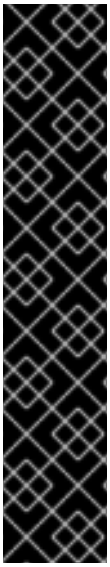
## 4.1. PREPARE OPENSHIFT FOR APPLICATION DEPLOYMENT USING JDK 11 IMAGE

Preparing OpenShift for application deployment using the JDK 11 image stream follows the same procedure as described in Prepare OpenShift for Application Deployment.

## 4.2. IMPORT JDK 11 IMAGE

Use the following command to import the JBoss EAP for OpenShift JDK 11 image stream and templates into your OpenShift project's namespace:

```
for resource in \
  eap72-openjdk11-image-stream.json \
  eap72-openjdk11-amq-persistent-s2i.json \
  eap72-openjdk11-amq-s2i.json \
  eap72-openjdk11-basic-s2i.json \
  eap72-openjdk11-https-s2i.json \
  eap72-openjdk11-sso-s2i.json \
  eap72-openjdk11-starter-s2i.json \
  eap72-openjdk11-third-party-db-s2i.json \
  eap72-openjdk11-tx-recovery-s2i.json
do
 oc replace --force -f \
https://raw.githubusercontent.com/jboss-container-images/jboss-eap-7-openshift-image/eap72-openjdk11-ubi8/templates/${resource}
done
```



**IMPORTANT**

The following internal datasources and drivers are not provided with the JBoss EAP for OpenShift JDK 11 image:

- MySQL
- PostgreSQL
- MongoDB

It is recommended that you use JDBC drivers obtained from your database vendor for your JBoss EAP applications.

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

## 4.3. DEPLOY A JBOSS EAP S2I APPLICATION TO OPENSHIFT USING JDK 11 IMAGE

Deploying a JBoss EAP S2I application to OpenShift follows the same procedures as described in Deploy a JBoss EAP Source-to-Image (S2I) Application to OpenShift .

The JDK 11 stream uses the **eap72-openjdk11-basic-s2i** for S2I builds, instead of the **eap72-basic-s2i** template used in JDK 8.

To deploy the **kitchensink** quickstart, use the following command to use the **eap72-openjdk11-basic-s2i** template with the **kitchensink** source code on GitHub:

```
oc new-app --template=eap72-openjdk11-basic-s2i \
 -p IMAGE_STREAM_NAMESPACE=eap-demo \
 -p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts.git \
 -p SOURCE_REPOSITORY_REF=openshift \
 -p CONTEXT_DIR=kitchensink \
```

The **eap72-openjdk11-basic-s2i** template in the **eap-demo** project was created in Prepare OpenShift for Application Deployment.

## 4.4. CONFIGURE JBOSS EAP FOR OPENSHIFT USING ENVIRONMENT VARIABLES FOR JDK 11 IMAGE

Configuring JBoss EAP for OpenShift using environment variables for JDK 11 follows the same procedures as described in Configuring JBoss EAP for OpenShift Using Environment Variables .

The JDK 11 image stream uses the **eap72-openjdk11-basic-s2i** template, instead of the **eap72-basic-s2i** template used in JDK 8.

To set the JBoss EAP instance's management username and password using environment variables, use the following command when creating your OpenShift application.

```
oc new-app --template=eap72-openjdk11-basic-s2i \
-p IMAGE_STREAM_NAMESPACE=eap-demo \
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-developer/jboss-eap-quickstarts \
-p SOURCE_REPOSITORY_REF=openshift \
-p CONTEXT_DIR=kitchensink \
-e ADMIN_USERNAME=myspecialuser \
-e ADMIN_PASSWORD=myspecialp@ssw0rd
```

# CHAPTER 5. MIGRATING APPLICATION TO OPENSHIFT 4

## 5.1. UPDATING LIVENESS AND READINESS PROBE CONFIGURATION FOR OPENSHIFT 4

The **YAML** configuration of probes must be adjusted when migrating to OpenShift 4.

On OpenShift 3.11, the default **YAML** configuration for liveness probes is similar to the following code example:

### Example YAML Configuration for OpenShift 3.11 Liveness Probe

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

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

The probe is considered a failure after 3 attempts to call the **livenessProbe.sh** script. The container is deemed unhealthy, and OpenShift will restart the JBoss EAP container in its respective pod.

On OpenShift 3.11, a single call lasts 5 seconds before it returns as a success or failure. On OpenShift 4, a single call lasts less than 1 second.

On OpenShift 3.11, a call to the probe lasts 5 seconds, followed by a 10 second waiting period. This means that 3 calls last approximately 35 seconds before the container inside the pod is restarted if the JBoss EAP image is unhealthy.

On OpenShift 4, 3 calls last approximately 23 seconds. The configuration of the probe for OpenShift 4 should be adjusted in the **YAML** configuration as follows:

### Example YAML Configuration for OpenShift 4 Liveness Probe

```
livenessProbe:
      exec:
        command:
          - /bin/bash
          - '-c'
          - /opt/eap/bin/livenessProbe.sh
      initialDelaySeconds: 60
      periodSeconds: 16
      successThreshold: 1
      failureThreshold: 3
```

In this example, **periodSeconds** has been increased by 6 seconds. Now the first call lasts 1 second, followed by a 16 second waiting period. Three calls would last approximately 34 seconds, which is nearly equivalent to the OpenShift 3.11 behavior of the probe.

For readiness probes, a similar adjustment must be made to the **YAML** configuration:

### Example YAML Configuration for OpenShift 4 Readiness Probe

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

### Additional Resources

Liveness and Readiness Probes

# CHAPTER 6. TROUBLESHOOTING

## 6.1. TROUBLESHOOTING POD RESTARTS

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

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

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

   ```
   $ oc get pods
   ```

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

   a. To see the JBoss EAP logs of the previous pod, use the following command.

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

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

   ```
   $ oc get events
   ```

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

## 6.2. TROUBLESHOOTING USING THE JBOSS EAP MANAGEMENT CLI

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

**IMPORTANT**

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

To make configuration changes to JBoss EAP for OpenShift, see Configuring the JBoss EAP for OpenShift Image for Your Java Application.

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

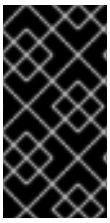
   ```
   $ oc rsh POD_NAME
   ```

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

```
$ /opt/eap/bin/jboss-cli.sh
```

# CHAPTER 7. ADVANCED TUTORIALS

## 7.1. EXAMPLE WORKFLOW: AUTOMATED TRANSACTION RECOVERY FEATURE WHEN SCALING DOWN A CLUSTER



IMPORTANT

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

This tutorial demonstrates the automated transaction recovery feature of the JBoss EAP for OpenShift image when scaling down a cluster. The **jta-crash-rec-eap7** quickstart example and the **eap72-tx-recovery-s2i** application template are used here to show how XA transactions issued on the OpenShift pod, when terminated within the cluster's scale down, are recovered by the dedicated migration pod.

NOTE

The **jta-crash-rec-eap7** quickstart uses the H2 database that is included with JBoss EAP. It is a lightweight, relational example datasource that is used for examples only. It is not robust or scalable, is not supported, and should not be used in a production environment.

### 7.1.1. Prepare for Deployment

1. Log in to your OpenShift instance using the **oc login** command.
2. Create a new project.

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

3. Add the view role to the **default** service account, which will be used to run the underlying pods. This enables the service account to view all the resources in the **eap-tx-demo** namespace, which is necessary for managing the cluster.

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

4. For automated transaction recovery to work, the JBoss EAP application must use a **ReadWriteMany** persistent volume.
   Provision the persistent volume expected by the **eap72-tx-recovery-s2i** application template to hold the data for the **${APPLICATION_NAME}-eap-claim** persistent volume claim.

   This example uses a persistent volume object provisioned using the NFS method with the following definition:

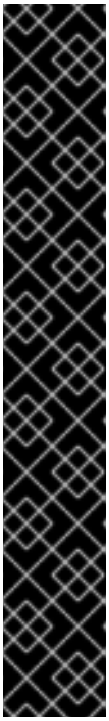
```
$ cat txpv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: txpv
spec:
  capacity:
```

```
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /mnt/mountpoint
    server: 192.168.100.175
```

Update the **path** and **server** fields in the above definition for your environment, and provision the persistent volume with the following command:

```
$ oc create -f txpv.yaml
persistentvolume "txpv" created
```

```
$ oc get pv
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS      CLAIM
STORAGECLASS   REASON    AGE
txpv      1Gi        RWX           Retain          Available                             26s
```



**IMPORTANT**

When using the NFS method to provision persistent volume objects for the **eap72-tx-recovery-s2i** application template, ensure the mount point is exported with sufficient permissions. On the host from which the mount point is exported, perform the following:

```
# chmod -R 777 /mnt/mountpoint
```

```
# cat /etc/exports
/mnt/mountpoint *(rw,sync,anonuid=185,anongid=185)
```

```
# exportfs -va
exporting *:/mnt/mountpoint
```

```
# setsebool -P virt_use_nfs 1
```

Replace **/mnt/mountpoint** path above as appropriate for your environment.

### 7.1.2. Deployment

1. Deploy the **jta-crash-rec-eap7** quickstart using the **eap72-tx-recovery-s2i** application template. Specify the following:

**Example: eap72-tx-recovery-s2i application template**

+

```
$ oc new-app --template=eap72-tx-recovery-s2i \
-p SOURCE_REPOSITORY_URL=https://github.com/jboss-openshift/openshift-quickstarts \
-p SOURCE_REPOSITORY_REF=master \
-p CONTEXT_DIR=jta-crash-rec-eap7 \
-e CUSTOM_INSTALL_DIRECTORIES=extensions/* \
```

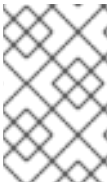
```
--name=eap-app
--> Deploying template "openshift/eap72-tx-recovery-s2i" to project eap-tx-demo

    JBoss EAP 7.0 (tx recovery)
    ---------
    An example EAP 7 application. For more information about using this template, see
https://github.com/jboss-openshift/application-templates.

    A new EAP 7 based application has been created in your project.

    * With parameters:
      * Application Name=eap-app
      * Custom http Route Hostname=
      * Git Repository URL=https://github.com/jboss-openshift/openshift-quickstarts
      * Git Reference=master
      * Context Directory=jta-crash-rec-eap7
      * Queues=
      * Topics=
      * A-MQ cluster password=nyneOXUm # generated
      * Github Webhook Secret=PUW8Tmov # generated
      * Generic Webhook Secret=o7uD7qrG # generated
      * ImageStream Namespace=openshift
      * JGroups Cluster Password=MoR1Jthf # generated
      * Deploy Exploded Archives=false
      * Maven mirror URL=
      * ARTIFACT_DIR=
      * MEMORY_LIMIT=1Gi
      * EAP Volume Size=1Gi
      * Split the data directory?=true

--> Creating resources ...
    service "eap-app" created
    service "eap-app-ping" created
    route "eap-app" created
    imagestream "eap-app" created
    buildconfig "eap-app" created
    deploymentconfig "eap-app" created
    deploymentconfig "eap-app-migration" created
    persistentvolumeclaim "eap-app-eap-claim" created
--> Success
    Build scheduled, use 'oc logs -f bc/eap-app' to track its progress.
    Run 'oc status' to view your app.
```



NOTE

The JDK 11 image stream uses the **eap72-openjdk11-tx-recovery-s2i** application template in the above example, instead of **eap72-tx-recovery-s2i** used in the JDK 8 image stream.

1. Wait for the build to finish. You can see the status of the build using the **oc logs -f bc/eap-app** command.

2. Modify the **eap-app** deployment configuration with the definition of **JAVA_OPTS_APPEND** and **JBOSS_MODULES_SYSTEM_PKGS_APPEND** environment variables.

```
$ oc get dc
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
eap-app           1          1         1         config,image(eap-app:latest)
eap-app-migration 1          1         1         config,image(eap-app:latest)
```

```
$ oc set env dc/eap-app \
-e JBOSS_MODULES_SYSTEM_PKGS_APPEND="org.jboss.byteman" \
-e JAVA_OPTS_APPEND="-
javaagent:/tmp/src/extensions/byteman/byteman.jar=script:/tmp/src/src/main/scripts/xa.btm"
deploymentconfig "eap-app" updated
```

This setting will notify the Byteman tracing and monitoring tool to modify the XA transactions processing in the following way:

- The first transaction is always allowed to succeed.
- When an XA resource executes phase 2 of the second transaction, the JVM process of the particular pod is halted.

### 7.1.3. Using the JTA Crash Recovery Application

1. List running pods in the current namespace:

```
$ oc get pods | grep Running
NAME                       READY     STATUS      RESTARTS   AGE
eap-app-2-r00gm            1/1       Running     0          1m
eap-app-migration-1-lvfdt  1/1       Running     0          2m
```

2. Issue a new XA transaction.
   a. Launch the application by opening a browser and navigating to http://eap-app-eap-tx-demo.openshift.example.com/jboss-jta-crash-rec.
   b. Enter **Mercedes** into the **Key** field, and **Benz** into the **Value** field. Click the **Submit** button.
   c. Wait for a moment, then click the **Refresh Table** link.
   d. Notice how the table row containing the **Mercedes** entry is updated with **updated via JMS.** If it has not yet updated, click the **Refresh Table** link couple of times. Alternatively, you can inspect the log of the **eap-app-2-r00gm** pod to verify the transaction was handled properly:

```
$ oc logs eap-app-2-r00gm | grep 'updated'
INFO  [org.jboss.as.quickstarts.xa.DbUpdaterMDB] (Thread-0 (ActiveMQ-client-global-
threads-1566836606)) JTA Crash Record Quickstart: key value pair updated via JMS.
```

3. Issue a second XA transaction using your browser at http://eap-app-eap-tx-demo.openshift.example.com/jboss-jta-crash-rec.
   a. Enter **Land** into the **Key** field, and **Rover** into the **Value** field. Click the **Submit** button.
   b. Wait for a moment, then click the **Refresh Table** link.
   c. Notice how the **Land Rover** entry was added without the **updated via ...** suffix.
4. Scale the cluster down.

```
$ oc scale --replicas=0 dc/eap-app
deploymentconfig "eap-app" scaled
```

a. Notice how the **eap-app-2-r00gm** pod was scheduled for termination.

```
$ oc get pods
NAME                       READY     STATUS        RESTARTS   AGE
eap-app-1-build            0/1       Completed     0          4m
eap-app-2-r00gm            1/1       Terminating   0          2m
eap-app-migration-1-lvfdt  1/1       Running       0          3m
```

5. Watch the log of the migration pod and notice how transaction recovery is performed. Wait for the recovery to finish:

```
$ oc logs -f eap-app-migration-1-lvfdt
Finished Migration Check cycle, pausing for 30 seconds before resuming
...
Finished, recovery terminated successfully
Migration terminated with status 0 (T)
Releasing lock: (/opt/eap/standalone/partitioned_data/split-1)
Finished Migration Check cycle, pausing for 30 seconds before resuming
...
```

6. Scale the cluster back up.

```
$ oc scale --replicas=1 dc/eap-app
deploymentconfig "eap-app" scaled
```

7. Using the browser navigate back to http://eap-app-eap-tx-demo.openshift.example.com/jboss-jta-crash-rec.

8. Notice the table contains entries for both transactions. It looks similar to the following output:

**Table 7.1. Example: Database Table Contents**

| Database Table Contents | |
|---|---|
| Key | Value |
| Mercedes | Benz updated via JMS. |
| Land | Rover updated via JMS. |

The content in the above table indicates that, although the cluster was scaled down before the second XA transaction had chance to finish, the migration pod performed the transaction recovery and the transaction was successfully completed.

# CHAPTER 8. REFERENCE INFORMATION

**NOTE**

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

## 8.1. PERSISTENT TEMPLATES

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

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

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

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

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

## 8.2. INFORMATION ENVIRONMENT VARIABLES

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

**Table 8.1. Information Environment Variables**

| Variable Name | Description and Value |
|---|---|
| JBOSS_IMAGE_NAME | The image name.<br><br>Value: **jboss-eap-7/eap72-openshift** |
| JBOSS_IMAGE_RELEASE | The image release label.<br><br>Value: **dev** |
| JBOSS_IMAGE_VERSION | The image version.<br><br>Value: This is the image version number. See the Red Hat Container Catalog for the latest value. |

| Variable Name | Description and Value |
| --- | --- |
| JBOSS_MODULES_SYSTEM_PKGS | A comma-separated list of JBoss EAP system modules packages that are available to applications.<br><br>Value: **org.jboss.logmanager**, **jdk.nashorn.api** |
| STI_BUILDER | Provides OpenShift S2I support for **jee** project types.<br><br>Value: **jee** |

## 8.3. CONFIGURATION ENVIRONMENT VARIABLES

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

**Table 8.2. Configuration Environment Variables**

| Variable Name | Description |
| --- | --- |
| AB_JOLOKIA_AUTH_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**.<br><br>● Set to **false** to disable client authentication for OpenShift TLS communication.<br><br>● Set to **true** to enable client authentication for OpenShift TLS communication using the default CA certificate and client principal.<br><br>● 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. |
| 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.<br><br>If not set, **/opt/jolokia/etc/jolokia.properties** is created using the settings as defined in the Jolokia reference documentation.<br><br>Example value: **/opt/jolokia/custom.properties** |
| AB_JOLOKIA_DISCOVERY_ENABLED | Enable Jolokia discovery.<br><br>Defaults to **false**. |

| Variable Name | Description |
|---|---|
| AB_JOLOKIA_HOST | Host address to bind to.<br><br>Defaults to **0.0.0.0**.<br><br>Example value: **127.0.0.1** |
| AB_JOLOKIA_HTTPS | Switch on secure communication with HTTPS.<br><br>By default self-signed server certificates are generated if no **serverCert** configuration is given in **AB_JOLOKIA_OPTS**.<br><br>Example value: **true** |
| AB_JOLOKIA_ID | Agent ID to use.<br><br>The default value is the **$HOSTNAME**, which is the container id.<br><br>Example value: **openjdk-app-1-xqlsj** |
| AB_JOLOKIA_OFF | If set to **true**, disables activation of Jolokia, which echos an empty value.<br><br>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, …**.<br><br>Example value: **backlog=20** |
| AB_JOLOKIA_PASSWORD | The password for basic authentication.<br><br>By default, authentication is switched off.<br><br>Example value: **mypassword** |
| AB_JOLOKIA_PASSWORD_RANDOM | Determines if a random **AB_JOLOKIA_PASSWORD** should be generated.<br><br>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.<br><br>Defaults to **8778**.<br><br>Example value: **5432** |

| Variable Name | Description |
|---|---|
| AB_JOLOKIA_USER | The name of the user to use for basic authentication.<br><br>Defaults to **jolokia**.<br><br>Example value: **myusername** |
| 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.<br><br>Example value: **true** |
| CONTAINER_HEAP_PERCENT | Set the maximum Java heap size, as a percentage of available container memory.<br><br>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.<br><br>Example value: **custom,shared** |
| DEFAULT_JMS_CONNECTION_FACTORY | This value is used to specify the default JNDI binding for the JMS connection factory, for example **jms-connection-factory='java:jboss/DefaultJMSConnectionFactory'**.<br><br>Example value: **java:jboss/DefaultJMSConnectionFactory** |
| ENABLE_ACCESS_LOG | Enable logging of access messages to the standard output channel.<br><br>Logging of access messages is implemented using following methods:<br><br>● The JBoss EAP 6.4 OpenShift image uses a custom JBoss Web Access Log Valve.<br><br>● The JBoss EAP for OpenShift image uses the Undertow **AccessLogHandler**.<br><br>Defaults to **false**. |
| INITIAL_HEAP_PERCENT | Set the initial Java heap size, as a percentage of the maximum heap size.<br><br>Example value: **0.5** |
| JAVA_OPTS_APPEND | Server startup options.<br><br>Example value: **-Dfoo=bar** |

| Variable Name | Description |
|---|---|
| JBOSS_MODULES_SYSTEM_PKGS_APPEND | A comma-separated list of package names that will be appended to the **JBOSS_MODULES_SYSTEM_PKGS** environment variable.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>Example value: **jgroups** |
| 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.<br><br>Example value: **mypassword** |
| JGROUPS_ENCRYPT_PROTOCOL | JGroups protocol to use for encryption of cluster traffic. Can be either **SYM_ENCRYPT** or **ASYM_ENCRYPT**.<br><br>Defaults to **SYM_ENCRYPT**.<br><br>Example value: **ASYM_ENCRYPT** |

| Variable Name | Description |
|---|---|
| JGROUPS_ENCRYPT_SECRET | Name of the secret, containing the JGroups keystore file used for securing the JGroups communications, when using **SYM_ENCRYPT** JGroups cluster traffic encryption protocol. If not set, cluster communication is not encrypted and a warning is issued.<br><br>Example value: **eap7-app-secret** |
| JGROUPS_PING_PROTOCOL | JGroups protocol to use for node discovery. Can be either **openshift.DNS_PING** or **openshift.KUBE_PING**. |
| MQ_SIMPLE_DEFAULT_PHYSICAL_DESTINATION | For backwards compatibility, set to **true** to use **MyQueue** and **MyTopic** as physical destination name defaults instead of **queue/MyQueue** and **topic/MyTopic**. |
| OPENSHIFT_DNS_PING_SERVICE_NAME | Name of the service exposing the ping port on the servers for the DNS discovery mechanism.<br><br>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 will be made to discover the port number from the SRV records for the service, otherwise the default **8888** will be used.<br><br>Defaults to **8888**. |
| OPENSHIFT_KUBE_PING_LABELS | Clustering labels selector for the Kubernetes discovery mechanism.<br><br>Example value: **app=eap-app** |
| OPENSHIFT_KUBE_PING_NAMESPACE | Clustering project namespace for the Kubernetes discovery mechanism.<br><br>Example value: **myproject** |
| SCRIPT_DEBUG | If set to **true**, ensures that the Bash scripts are executed with the **-x** option, printing the commands and their arguments as they are executed. |

NOTE

Other environment variables not listed above that can influence the product can be found in the JBoss EAP documentation.

## 8.4. APPLICATION TEMPLATES

**Table 8.3. Application Templates**

| Variable Name | Description |
| --- | --- |
| AUTO_DEPLOY_EXPLODED | Controls whether exploded deployment content should be automatically deployed.<br><br>Example value: **false** |

## 8.5. EXPOSED PORTS

**Table 8.4. Exposed Ports**

| Port Number | Description |
| --- | --- |
| 8443 | HTTPS |
| 8778 | Jolokia Monitoring |

## 8.6. DATASOURCES

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

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

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

### 8.6.1. JNDI Mappings for Datasources

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

NOTE

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

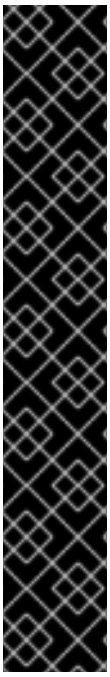
The ***DATABASETYPE*** determines the driver for the datasource.

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

**WARNING**

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

#### 8.6.1.1. Database Drivers



**IMPORTANT**

Support for using the Red Hat-provided internal datasource drivers with the JBoss EAP for OpenShift image is now deprecated for JDK 8 image streams. It is recommended 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 JDK 11 image:

- MySQL
- PostgreSQL

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

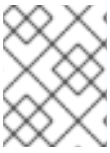
For more information on configuring JDBC drivers with JBoss EAP, see JDBC drivers in the JBoss EAP *Configuration Guide*.

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

**IMPORTANT**

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

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



**NOTE**

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

#### 8.6.1.2. Datasource Configuration Environment Variables

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

IMPORTANT

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

| Variable Name | Description |
|---|---|
| *POOLNAME_DATABASETYPE*_SERVICE_HOST | Defines the database server's host name or IP address to be used in the datasource's **connection-url** property.<br><br>Example value: **192.168.1.3** |
| *POOLNAME_DATABASETYPE*_SERVICE_PORT | Defines the database server's port for the datasource.<br><br>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_MILLIS | 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.<br><br>Example value: **org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker** |
| *PREFIX*_DATABASE | Defines the database name for the datasource.<br><br>Example value: **myDatabase** |
| *PREFIX*_DRIVER | Defines Java database driver for the datasource.<br><br>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.<br><br>Example value: **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter** |

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

#### 8.6.1.3. Examples

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

#### 8.6.1.3.1. Single Mapping

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

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

#### 8.6.1.3.2. Multiple Mappings

You can specify multiple datasource mappings.

NOTE

Always separate multiple datasource mappings with a comma.

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

This creates the following two datasources:

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

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

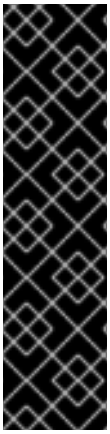
## 8.7. CLUSTERING

### 8.7.1. Configuring a JGroups Discovery Mechanism

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

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

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



IMPORTANT

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

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

#### 8.7.1.1. Configuring KUBE_PING

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

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

   ```
   JGROUPS_PING_PROTOCOL=kubernetes.KUBE_PING
   ```

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

   ```
   KUBERNETES_NAMESPACE=PROJECT_NAME
   ```

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

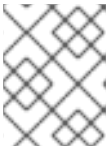
   ```
   KUBERNETES_LABELS=application=APP_NAME
   ```

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

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

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

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



NOTE

See Prepare OpenShift for Application Deployment for more information on adding policies to service accounts.

#### 8.7.1.2. Configuring DNS_PING

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

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

```
JGROUPS_PING_PROTOCOL=openshift.DNS_PING
```

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

```
OPENSHIFT_DNS_PING_SERVICE_NAME=PING_SERVICE_NAME
```

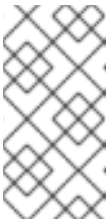
3. The **OPENSHIFT_DNS_PING_SERVICE_PORT** environment variable should be set to the port number on which the ping service is exposed. The **DNS_PING** protocol attempts to discern the port from the SRV records, otherwise it defaults to **8888**.

```
OPENSHIFT_DNS_PING_SERVICE_PORT=PING_PORT
```

4. A ping service which exposes the ping port must be defined. This service should be headless (ClusterIP=None) and must have the following:

   a. The port must be named.

   b. The service must be annotated with **service.alpha.kubernetes.io/tolerate-unready-endpoints** set to **"true"**.



NOTE

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

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

NOTE

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

### 8.7.2. Configuring JGroups to Encrypt Cluster Traffic

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

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

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

IMPORTANT

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

#### 8.7.2.1. Configuring SYM_ENCRYPT

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

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

   ```
   JGROUPS_ENCRYPT_PROTOCOL=SYM_ENCRYPT
   ```

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

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

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

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

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

   ```
   JGROUPS_ENCRYPT_KEYSTORE=jgroups.jceks
   ```

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

```
JGROUPS_ENCRYPT_NAME=jgroups
```

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

```
JGROUPS_ENCRYPT_PASSWORD=mypassword
```

#### 8.7.2.2. Configuring ASYM_ENCRYPT

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

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

```
JGROUPS_ENCRYPT_PROTOCOL=ASYM_ENCRYPT
```

2. The **JGROUPS_CLUSTER_PASSWORD** environment variable must be set to the password used to authenticate the node so it is allowed to join the JGroups cluster. If not set, authentication is disabled, cluster communication is not encrypted and a warning is issued. For example:

```
JGROUPS_CLUSTER_PASSWORD=mypassword
```

**NOTE**

If you configure **ASYM_ENCRYPT** but also define any of the environment variables required for **SYM_ENCRYPT**, the **SYM_ENCRYPT** environment variables are ignored and a warning is issued. In this situation, cluster communication is still encrypted using **ASYM_ENCRYPT**.

## 8.8. HEALTH CHECKS

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

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

**Table 8.5. Liveness and Readiness Checks**

| Performed Test | Liveness | Readiness |
|---|---|---|
| Server Status | Any status | Running |

| Performed Test | Liveness | Readiness |
|---|---|---|
| Boot Errors | None | None |
| 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.

## 8.9. MESSAGING

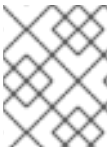
### 8.9.1. Configuring External Red Hat AMQ Brokers

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

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

**Example: eap72-amq-s2i application template**

```
oc new-app eap72-amq-s2i \
-p APPLICATION_NAME=eap72-mq \
-p MQ_USERNAME=MY_USERNAME \
-p MQ_PASSWORD=MY_PASSWORD
```



NOTE

The JDK 11 image stream uses the **eap72-openjdk11-amq-s2i** application template in the above example, instead of **eap72-amq-s2i** used in the JDK 8 image stream.

IMPORTANT

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

## 8.10. SECURITY DOMAINS

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

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

Table 8.6. Security Domains

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

## 8.11. HTTPS ENVIRONMENT VARIABLES

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

## 8.12. ADMINISTRATION ENVIRONMENT VARIABLES

Table 8.7. Administration Environment Variables

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

## 8.13. S2I

The image includes S2I scripts and Maven.

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

Only WAR deployments are supported at this time.

### 8.13.1. Custom Configuration

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

#### 8.13.1.1. Custom Modules

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

### 8.13.2. Deployment Artifacts

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

### 8.13.3. Artifact Repository Mirrors

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

Benefits of using a mirror are:

- Availability of a synchronized mirror, which is geographically closer and faster.
- Ability to have greater control over the repository content.

- Possibility to share artifacts across different teams (developers, CI), without the need to rely on public servers and repositories.
- Improved build times.

Often, a repository manager can serve as local cache to a mirror. Assuming that the repository manager is already deployed and reachable externally at **http://10.0.0.1:8080/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
```

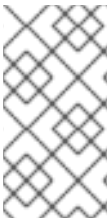
2. Update build configuration of **eap** with a **MAVEN_MIRROR_URL** environment variable.

```
oc env bc/eap MAVEN_MIRROR_URL="http://10.0.0.1:8080/repository/internal/"
buildconfig "eap" updated
```

3. Verify the setting.

```
oc env bc/eap --list
# buildconfigs eap
MAVEN_MIRROR_URL=http://10.0.0.1:8080/repository/internal/
```

4. Schedule new build of the application.



NOTE

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

### 8.13.4. Scripts

**run**

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

**assemble**

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

### 8.13.5. Environment Variables

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

**Table 8.8. s2i Environment Variables**

| Variable name | Description |
|---|---|
| ARTIFACT_DIR | The **.war**, **.ear**, and **.jar** files from this directory will be copied into the **deployments/** directory.<br><br>Example value: **target** |
| HTTP_PROXY_HOST | Host name or IP address of a HTTP proxy for Maven to use.<br><br>Example value: **192.168.1.1** |
| HTTP_PROXY_PORT | TCP Port of a HTTP proxy for Maven to use.<br><br>Example value: **8080** |
| HTTP_PROXY_USERNAME | If supplied with **HTTP_PROXY_PASSWORD**, use credentials for HTTP proxy.<br><br>Example value: **myusername** |
| HTTP_PROXY_PASSWORD | If supplied with **HTTP_PROXY_USERNAME**, use credentials for HTTP proxy.<br><br>Example value: **mypassword** |
| HTTP_PROXY_NONPROXYHOSTS | If supplied, a configured HTTP proxy will ignore these hosts.<br><br>Example value: **some.example.org\|*.example.net** |
| MAVEN_ARGS | Overrides the arguments supplied to Maven during build.<br><br>Example value: **-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package** |
| MAVEN_ARGS_APPEND | Appends user arguments supplied to Maven during build.<br><br>Example value: **-Dfoo=bar** |
| MAVEN_MIRROR_URL | URL of a Maven Mirror/repository manager to configure.<br><br>Example value: **http://10.0.0.1:8080/repository/internal/** |
| MAVEN_CLEAR_REPO | Optionally clear the local Maven repository after the build.<br><br>Example value: **true** |
| APP_DATADIR | If defined, directory in the source from where data files are copied.<br><br>Example value: **mydata** |

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

**NOTE**

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

## 8.14. SSO

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

**NOTE**

See the Red Hat JBoss SSO for OpenShift documentation for more information on how to deploy the Red Hat JBoss SSO for OpenShift image with the JBoss EAP for OpenShift image.

**Table 8.9. SSO Environment Variables**

| Variable name | Description |
|---|---|
| SSO_URL | URL of the SSO server. |
| SSO_REALM | SSO realm for the deployed applications. |
| SSO_PUBLIC_KEY | Public key of the SSO Realm. This field is optional but if omitted can leave the applications vulnerable to man-in-middle attacks. |
| SSO_USERNAME | SSO User required to access the SSO REST API.<br><br>Example value: **mySsoUser** |
| SSO_PASSWORD | Password for the SSO user defined by the **SSO_USERNAME** variable.<br><br>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 | SSO Client Access Type. (Optional)<br><br>Example value: **true** |
| SSO_CLIENT | Path for SSO redirects back to the application. Defaults to match **module-name**. |
| SSO_ENABLE_CORS | If **true**, enable CORS for SSO applications. (Optional) |
| SSO_SECRET | The SSO Client Secret for Confidential Access.<br><br>Example value: **KZ1Qylq4** |
| SSO_DISABLE_SSL_CERTIFICATE_VALIDATION | If **true** the SSL/TLS communication between JBoss EAP and the RH-SSO server will be unsecure, for example, the certificate validation is disabled with **curl**. Not set by default.<br><br>Example value: **true** |

## 8.15. TRANSACTION RECOVERY

When a cluster is scaled down, it is possible for transaction branches to be in doubt. There is a technology preview automated recovery pod that is meant to complete these branches, but there are rare scenarios, such as a network split, where the recovery may fail. In these cases, manual transaction recovery might be necessary.

### 8.15.1. Unsupported Transaction Recovery Scenarios

- JTS transactions
  Because the network endpoint of the parent is encoded in recovery coordinator IORs, recovery cannot work reliably if either the child or parent node recovers with either a new IP address, or if it is intended to be accessed using a virtualized IP address.
- XTS transactions
  XTS does not work in a clustered scenario for recovery purposes. See JBTM-2742 for details.
- Transactions propagated over JBoss Remoting
- Transactions propagated over XATerminator
  Because the EIS is intended to be connected to a single instance of a Java EE application server, there are no well-defined ways to couple these processes.

### 8.15.2. Manual Transaction Recovery Process

The goal of the following procedure is to find and manually resolve in-doubt branches in cases where automated recovery has failed.

#### 8.15.2.1. Caveats

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

**IMPORTANT**

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

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

The deployment must use a JDBC object store.

**IMPORTANT**

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

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

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

However, this process might impact the uniqueness of the identifier. For example, the names **aaa12345678901234567 8m0jwh** and **bbb12345678901234567 8m0jwh** are both truncated to **12345678901234567 8m0jwh**, which breaks the uniqueness of the names that are expected. In another example, **this-pod-is-m0jwh** and **thispod-is-m0jwh** are both truncated to **thispodism0jwh**, again breaking the uniqueness of the names.

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

### 8.15.2.2. Prerequisite

It is assumed the OpenShift instance has been configured with a JDBC store, and that the store tables are partitioned using a table prefix corresponding to the pod name. This should be automatic whenever a JBoss EAP deployment is in use. This is different from the automated recovery example, which uses a file store with split directories on a shared volume. You can verify that the JBoss EAP instance is using a JDBC object store by looking at the configuration of the transactions subsystem in a running pod:

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

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

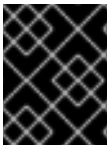
2. If the JDBC object store is in use, then there is an entry similar to the following:

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

NOTE

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

#### 8.15.2.3. Procedure



IMPORTANT

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

1. Use the database vendor tooling to list the XIDs (transaction branch identifiers) for in-doubt branches. It is necessary to list XIDs *for all datasources that were in use by any deployments running on the pod* that failed or was scaled down. Refer to the vendor documentation for the database product in use.

2. For each such XID, determine which pod created the transaction and check to see if that pod is still running.

   a. If it is running, then leave the branch alone.

   b. If the pod is not running, assume it was removed from the cluster and you must apply the manual resolution procedure described here. Look in the transaction log storage that was used by the failed pod to see if there is a corresponding transaction log:

      i. If there is a log, then manually commit the XID using the vendor tooling.

      ii. If there is not a log, assume it is an orphaned branch and roll back the XID using the vendor tooling.

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

##### 8.15.2.3.1. Resolving In-doubt Branches

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

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

1. Open a terminal on a pod running a JBoss EAP instance in the cluster of the failed pod. If there is no such pod, scale up to one.

2. Create a management user using the **/opt/eap/bin/add-user.sh** script.

3. Log into the management CLI using the **/opt/eap/bin/jboss-cli.sh** script.

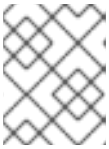
4. List the datasources configured on the server. These are the ones that may contain in-doubt transaction branches.

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

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

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

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



NOTE

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

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

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

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

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

sampledb=# select gid from pg_prepared_xacts;
131077_AAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGHAtanRhLWNyYXNoLXJlY0zLXAy
Y2N3_AAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGgAAAAEAAAAA
```

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

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

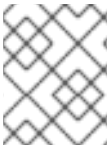
For example, the following Bash script can be used to perform this conversion. Assuming that **$PG_XID** holds the XID from the select statement above, then the JBoss EAP transaction ID can be obtained as follows:

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

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

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

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



NOTE

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

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

- If there is an entry in the table that matches the global transaction ID, then the in-doubt branch needs to be committed using the datasource vendor tooling as described below.
- If there is no such entry, then the branch is an orphan and can safely be rolled back.

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

```
$ oc rsh postgresql-2-vwf9n
sh-4.2$ psql sampledb
psql (9.5.7)
Type "help" for help.
psql sampledb
commit prepared '131077_AAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGHAtanRh
 ----
LWNyYXNoLXJlYy0zLXAyY2N3_AAAAAAAAAAAAP//rBEAB440GK1aJ72oAAAAGgAAAAEAAAAA';
```

**IMPORTANT**

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

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

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

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

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

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

**IMPORTANT**

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

#### 8.15.2.3.4. Find the Transaction Logs

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

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

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

NOTE

The *ostxrecoveryapp426p4rjbosststxtable* table name listed in the following command has been chosen since it follows the pattern for JDBC table names holding the log storage entries. In your environment the table name will have similar form:

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

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

sampledb=# select uidstring from ostxrecoveryapp426p4rjbosststxtable where
TYPENAME='StateManager/BasicAction/TwoPhaseCoordinator/AtomicAction'
;
            uidstring
 ------------------------------------
 0:ffff0a81009d:33789827:5a68b2bf:40
 (1 row)
```

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

WARNING

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

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

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

```
DELETE FROM ostxrecoveryapp426p4rjbosststxtable where uidstring = UIDSTRING
```

**IMPORTANT**

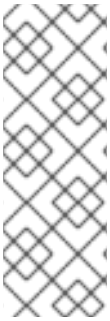
If you do not delete the log, then completed transactions which failed after prepare, but which have now been resolved, will never be removed from the transaction log storage. The consequence of this is that unnecessary storage is used and future manual reconciliation will be more difficult.

## 8.16. INCLUDED JBOSS MODULES

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

**Table 8.10. Included JBoss Modules**

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



**NOTE**

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

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

*Revised on 2019-09-12 10:36:37 UTC*