



OpenShift Enterprise 3.1

Using Images

OpenShift Enterprise 3.1 Guide to Using Images

OpenShift Enterprise 3.1 Using Images

OpenShift Enterprise 3.1 Guide to Using Images

Legal Notice

Copyright © 2018 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Use these topics to find out what different S2I (Source-to-Image), database and Docker images are available for OpenShift Enterprise 3.1 users.

Table of Contents

CHAPTER 1. OVERVIEW	7
CHAPTER 2. SOURCE-TO-IMAGE (S2I)	8
2.1. OVERVIEW	8
2.2. NODE.JS	8
2.2.1. Overview	8
2.2.2. Versions	8
2.2.3. Images	8
2.2.4. Configuration	8
2.3. RUBY	8
2.3.1. Overview	9
2.3.2. Versions	9
2.3.3. Images	9
2.3.4. Configuration	9
2.3.5. Hot Deploying	10
2.4. PERL	11
2.4.1. Overview	11
2.4.2. Versions	11
2.4.3. Images	11
2.4.4. Configuration	12
2.4.5. Accessing Logs	12
2.4.6. Hot Deploying	12
2.5. PHP	13
2.5.1. Overview	13
2.5.2. Versions	13
2.5.3. Images	13
2.5.4. Configuration	14
2.5.4.1. Apache Configuration	15
2.5.5. Accessing Logs	15
2.5.6. Hot Deploying	15
2.6. PYTHON	16
2.6.1. Overview	16
2.6.2. Versions	16
2.6.3. Images	16
2.6.4. Configuration	16
2.6.5. Hot Deploying	17
CHAPTER 3. DATABASE IMAGES	19
3.1. OVERVIEW	19
3.2. MYSQL	19
3.2.1. Overview	19
3.2.2. Versions	19
3.2.3. Images	19
3.2.4. Configuration and Usage	19
3.2.4.1. Initializing the Database	20
3.2.4.2. Running MySQL Commands in Containers	20
3.2.4.3. Environment Variables	20
3.2.4.4. Volume Mount Points	22
3.2.4.5. Changing Passwords	22
3.2.5. Creating a Database Service from a Template	23
3.2.6. Using MySQL Replication	24
3.2.6.1. Creating the Deployment Configuration for the MySQL Master	24

3.2.6.2. Creating a Headless Service	27
3.2.6.3. Scaling the MySQL Slaves	28
3.2.7. Troubleshooting	28
3.2.7.1. Linux Native AIO Failure	28
3.3. POSTGRESQL	29
3.3.1. Overview	29
3.3.2. Versions	29
3.3.3. Images	29
3.3.4. Configuration and Usage	30
3.3.4.1. Initializing the Database	30
3.3.4.2. Running PostgreSQL Commands in Containers	30
3.3.4.3. Environment Variables	31
3.3.4.4. Volume Mount Points	32
3.3.4.5. Changing Passwords	32
3.3.5. Creating a Database Service from a Template	33
3.4. MONGODB	34
3.4.1. Overview	34
3.4.2. Versions	34
3.4.3. Images	34
3.4.4. Configuration and Usage	34
3.4.4.1. Initializing the Database	35
3.4.4.2. Running MongoDB Commands in Containers	35
3.4.4.3. Environment Variables	35
3.4.4.4. Volume Mount Points	36
3.4.4.5. Changing Passwords	36
3.4.5. Creating a Database Service from a Template	38
3.4.6. Using MongoDB Replication	38
3.4.6.1. Creating the Deployment Configuration	39
3.4.6.2. Creating the Service Pod	40
3.4.6.3. Creating a Headless Service	41
3.4.6.4. Creating the Final Replication Set	42
3.4.6.5. Scaling the MongoDB Replication Set	42
CHAPTER 4. DOCKER IMAGES	43
4.1. OVERVIEW	43
CHAPTER 5. OTHER IMAGES	44
5.1. OVERVIEW	44
5.2. JENKINS	44
5.2.1. Overview	44
5.2.2. Versions	44
5.2.3. Images	44
5.2.4. Configuration and Usage	44
5.2.4.1. Initializing Jenkins	44
5.2.4.2. Environment Variables	45
5.2.4.3. Volume Mount Points	45
5.2.5. Creating a Jenkins Service from a Template	45
5.2.6. Using Jenkins as a Source-To-Image builder	46
5.2.7. Using the Jenkins Kubernetes Plug-in to Run Jobs	47
5.2.8. Tutorial	47
CHAPTER 6. XPAAS MIDDLEWARE IMAGES	48
6.1. OVERVIEW	48
6.2. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (JBOSS EAP) XPAAS IMAGES	48

6.2.1. Overview	48
6.2.2. Comparing the Product and Image	48
6.2.3. Comparing the xPaaS JBoss EAP 6.4 and 7.0 Beta Images	48
6.2.4. Compatibility with xPaaS JBoss EAP	49
6.2.5. Setting Up the xPaaS JBoss EAP Image	49
6.2.6. Modifying the JDK Used by the xPaaS JBoss EAP Image	50
6.2.7. Getting Started Using xPaaS JBoss EAP Images	50
6.2.7.1. Configuring the xPaaS JBoss EAP Images	50
6.2.7.2. Configuring the xPaaS JBoss EAP Image using the S2I Templates	50
6.2.7.3. Using a Modified xPaaS JBoss EAP Image	51
6.2.7.4. Troubleshooting	52
6.3. RED HAT JBOSS A-MQ XPAAS IMAGE	52
6.3.1. Overview	52
6.3.2. Differences Between the JBoss A-MQ xPaaS Image and the Regular Release of JBoss A-MQ	53
6.3.3. Using the JBoss A-MQ xPaaS Image Streams and Application Templates	53
6.3.4. Configuring the JBoss A-MQ Image	53
6.3.4.1. Application Template Parameters	53
6.3.4.2. Configuration Using S2I	54
6.3.5. Configuring the JBoss A-MQ Persistent Image	54
6.3.5.1. Application Template Parameters	54
6.3.6. Security	55
6.3.7. High-Availability and Scalability	55
6.3.8. Logging	56
6.4. RED HAT JBOSS WEB SERVER XPAAS IMAGES	56
6.4.1. Overview	56
6.4.2. Functionality Differences in the OpenShift JBoss Web Server xPaaS Images	56
6.4.3. Using the JBoss Web Server xPaaS Image Streams and Application Templates	56
6.4.4. Using the JBoss Web Server xPaaS Image Source-to-Image (S2I) Process	57
6.4.5. Troubleshooting	57
6.5. RED HAT JBOSS FUSE INTEGRATION SERVICES	57
6.5.1. Overview	57
6.5.1.1. Differences Between Fuse Integration Services and JBoss Fuse	58
6.5.2. Using Fuse Integration Services	58
6.5.2.1. Maven Archetypes Catalog	58
6.5.2.2. Create an Application from the Maven Archetype Catalog	59
6.5.2.3. Fabric8 Maven Workflow	60
6.5.2.3.1. Authenticating Against a Registry	61
6.5.2.3.2. Plug-in Configuration	62
6.5.2.4. OpenShift Source-to-Image (S2I) Workflow	62
6.5.2.4.1. Create an Application Using Templates	63
6.5.2.5. Developing Applications	64
6.5.2.5.1. Injecting Kubernetes Services into Applications	64
6.5.2.5.1.1. CDI Injection	64
6.5.2.5.1.2. Using Environment Variables as Properties	64
6.6. DECISION SERVER XPAAS IMAGE	64
6.6.1. Overview	65
6.6.2. Comparing the Decision Server xPaaS Image to the Regular Release of JBoss BRMS	65
6.6.2.1. Functionality Differences for OpenShift Decision Server xPaaS Images	65
6.6.2.2. Managing OpenShift Decision Server xPaaS Images	65
6.6.2.3. Security in the OpenShift Decision Server xPaaS Image	66
6.6.3. Using the Decision Server xPaaS Image Streams and Application Templates	66
6.6.4. Running and Configuring the Decision Server xPaaS Image	66

6.6.4.1. Using the Decision Server xPaaS Image Source-to-Image (S2I) Process	66
6.6.4.2. Using a Modified Decision Server xPaaS Image	67
6.6.4.3. Updating Rules	68
6.6.5. Endpoints	68
6.6.5.1. REST	68
6.6.5.1.1. Browser	68
6.6.5.1.2. Java	68
6.6.5.1.3. Command Line	68
6.6.5.2. JMS	69
6.6.5.2.1. Java (HornetQ)	69
6.6.5.2.2. Java (ActiveMQ)	69
6.6.6. Troubleshooting	70
6.7. RED HAT JBOSS DATA GRID XPAAS IMAGE	70
6.7.1. Overview	70
6.7.2. Comparing the JBoss Data Grid xPaaS Image to the Regular Release of JBoss Data Grid	71
6.7.2.1. Functionality Differences for OpenShift JBoss Data Grid xPaaS Images	71
6.7.2.2. Forming a Cluster using the OpenShift JBoss Data Grid xPaaS Images	71
6.7.2.3. Endpoints	72
6.7.2.4. Configuring Caches	72
6.7.2.5. Datasources	72
6.7.2.5.1. JNDI Mappings for Datasources	72
6.7.2.5.2. Database Drivers	73
6.7.2.5.3. Examples	73
6.7.2.5.3.1. Single Mapping	73
6.7.2.5.3.2. Multiple Mappings	73
6.7.2.5.4. Environment Variables	74
6.7.2.6. Security Domains	74
6.7.2.7. Managing OpenShift JBoss Data Grid xPaaS Images	74
6.7.3. Using the JBoss Data Grid xPaaS Image Streams and Application Templates	74
6.7.4. Running and Configuring the JBoss Data Grid xPaaS Image	75
6.7.4.1. Using the JBoss Data Grid xPaaS Image Source-to-Image (S2I) Process	75
6.7.4.1.1. Using a Different JDK Version in the JBoss Data Grid xPaaS Image	75
6.7.4.2. Using a Modified JBoss Data Grid xPaaS Image	76
6.7.5. Environment Variables	76
6.7.5.1. Information Environment Variables	76
6.7.5.2. Configuration Environment Variables	77
6.7.5.3. Cache Environment Variables	80
6.7.5.4. Datasource Environment Variables	83
6.7.5.5. Security Environment Variables	84
6.7.6. Exposed Ports	84
6.7.7. Troubleshooting	85
6.8. RED HAT SINGLE SIGN-ON (SSO) XPAAS IMAGE	85
6.8.1. Overview	85
6.8.2. Differences Between the SSO xPaaS Application and Keycloak	86
6.8.3. Versioning for xPaaS Images	86
6.8.4. Prerequisites for Deploying the SSO xPaaS Image	86
6.8.5. Using the SSO Image Streams and Application Templates	87
6.8.6. Preparing and Deploying the SSO xPaaS Application Templates	87
6.8.6.1. Using the OpenShift CLI	87
6.8.6.2. Using the OpenShift Web Console	87
6.8.6.3. Deployment Process	88
6.8.7. Quickstart Example: Using the SSO xPaaS Image with the SSO-enabled JBoss EAP xPaaS Image	88

6.8.7.1. Deploy the SSO Application Template	88
6.8.7.2. Create SSO Credentials	88
6.8.7.3. Deploy the SSO-enabled JBoss EAP Image	89
6.8.7.3.1. Alternate Deployments	90
6.8.7.4. Log in to the JBoss EAP Server Using SSO	90
6.8.8. Known Issues	90
CHAPTER 7. REVISION HISTORY: USING IMAGES	92
7.1. TUE SEP 13 2016	92
7.2. WED JUL 27 2016	92
7.3. WED JUL 20 2016	92
7.4. MON JUN 13 2016	92
7.5. MON MAY 30 2016	92
7.6. TUE MAY 03 2016	93
7.7. MON APR 04 2016	93
7.8. TUE MAR 29 2016	93
7.9. TUE MAR 22 2016	93
7.10. MON MAR 21 2016	93
7.11. MON FEB 22 2016	94
7.12. MON FEB 01 2016	94
7.13. TUE JAN 26 2016	94
7.14. THU NOV 19 2015	94

CHAPTER 1. OVERVIEW

Use these topics to discover the different [S2I \(Source-to-Image\)](#), database, and other Docker images that are available for OpenShift Enterprise users.

Red Hat's official container images are provided in the Red Hat Registry at registry.access.redhat.com. OpenShift Enterprise's supported S2I, database, and Jenkins images are provided in the [openshift3 repository](#) in the Red Hat Registry. For example, **`registry.access.redhat.com/openshift3/nodejs-010-rhel7`** for the Node.js image.

The xPaaS middleware images are provided in their respective product repositories on the Red Hat Registry, but suffixed with a **-openshift**. For example, **`registry.access.redhat.com/jboss-eap-6/eap64-openshift`** for the JBoss EAP image.

CHAPTER 2. SOURCE-TO-IMAGE (S2I)

2.1. OVERVIEW

This topic group includes information on the different [S2I \(Source-to-Image\)](#) supported images available for OpenShift Enterprise users.

2.2. NODE.JS

2.2.1. Overview

OpenShift Enterprise provides [S2I](#) enabled Node.js images for building and running Node.js applications. The Node.js S2I builder image assembles your application source with any required dependencies to create a new image containing your Node.js application. This resulting image can be run either by OpenShift Enterprise or by Docker.

2.2.2. Versions

Currently, OpenShift Enterprise provides version [0.10](#) of Node.js.

2.2.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry using:

```
$ docker pull registry.access.redhat.com/openshift3/nodejs-010-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/nodejs-010-centos7
```

To use these images, you can either access them directly from these [image registries](#), or push them into your [OpenShift Enterprise Docker registry](#). Additionally, you can create an [image stream](#) that points to the image, either in your Docker registry or at the external location. Your OpenShift Enterprise resources can then reference the ImageStream. You can find [example image stream definitions](#) for all the provided OpenShift Enterprise images.

2.2.4. Configuration

The Node.js image does not offer any environment variable based configuration settings.

2.3. RUBY

2.3.1. Overview

OpenShift Enterprise provides [S2I](#) enabled Ruby images for building and running Ruby applications. The Ruby S2I builder image assembles your application source with any required dependencies to create a new image containing your Ruby application. This resulting image can be run either by OpenShift Enterprise or by Docker.

2.3.2. Versions

Currently, OpenShift Enterprise provides versions [2.0](#), [2.2](#), and [2.3](#) of Ruby.

2.3.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry using:

```
$ docker pull registry.access.redhat.com/openshift3/ruby-20-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/ruby-20-centos7
```

To use these images, you can either access them directly from these [image registries](#), or push them into your [OpenShift Enterprise Docker registry](#). Additionally, you can create an [image stream](#) that points to the image, either in your Docker registry or at the external location. Your OpenShift Enterprise resources can then reference the ImageStream. You can find [example image stream definitions](#) for all the provided OpenShift Enterprise images.

2.3.4. Configuration

The Ruby image supports a number of environment variables which can be set to control the configuration and behavior of the Ruby runtime.

To set these environment variables, you can place them into a [.sti/environment](#) file inside your source code repository, or define them in [the environment section](#) of the build configuration's **sourceStrategy** definition.

Table 2.1. Ruby Environment Variables

Variable name	Description
---------------	-------------

Variable name	Description
RACK_ENV	This variable specifies the environment within which the Ruby application is deployed; for example, production , development , or test . Each level has different behavior in terms of logging verbosity, error pages, and ruby gem installation. The application assets are only compiled if RACK_ENV is set to production ; the default value is production .
RAILS_ENV	This variable specifies the environment within which the Ruby on Rails application is deployed; for example, production , development , or test . Each level has different behavior in terms of logging verbosity, error pages, and ruby gem installation. The application assets are only compiled if RAILS_ENV is set to production . This variable is set to \${RACK_ENV} by default.
DISABLE_ASSET_COMPILATION	This variable set to true disables the process of asset compilation. Asset compilation only happens when the application runs in a production environment. Therefore, you can use this variable when assets have already been compiled.

2.3.5. Hot Deploying

Hot deployment allows you to quickly make and deploy changes to your application without having to generate a new S2I build. The method for enabling hot deployment in this image differs based on the application type.

Ruby on Rails Applications

For Ruby on Rails application, run the built Rails application with the **RAILS_ENV=development** environment variable passed to the running pod. For an existing deployment configuration, you can use [the oc env](#) command:

```
$ oc env dc/rails-app RAILS_ENV=development
```

Other Types of Ruby Applications (Sinatra, Padrino, etc.)

For other types of Ruby applications, your application must be built with a gem that can reload the server every time a change to the source code is made inside the running container. Those gems are:

- [Shotgun](#)
- [Rerun](#)
- [Rack-livereload](#)

In order to be able to run your application in development mode, you must modify the [S2I run script](#) so that the web server is launched by the chosen gem, which checks for changes in the source code.

After you build your application image with your version of the [S2I run script](#), run the image with the **RACK_ENV=development** environment variable. For example, see the [oc new-app](#) command. You can use the [oc env](#) command to update environment variables of existing objects.



WARNING

You should only use this option while developing or debugging; it is not recommended to turn this on in your production environment.

To change your source code in a running pod, use the [oc rsh](#) command to enter the container:

```
$ oc rsh <pod_id>
```

After you enter into the running container, your current directory is set to **/opt/app-root/src**, where the source code is located.

2.4. PERL

2.4.1. Overview

OpenShift Enterprise provides [S2I](#) enabled Perl images for building and running Perl applications. The Perl S2I builder image assembles your application source with any required dependencies to create a new image containing your Perl application. This resulting image can be run either by OpenShift Enterprise or by Docker.

2.4.2. Versions

Currently, OpenShift Enterprise supports version [5.16](#) of Perl.

2.4.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry using:

```
$ docker pull registry.access.redhat.com/openshift3/perl-516-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/perl-516-centos7
```

To use these images, you can either access them directly from these [image registries](#), or push them into your [OpenShift Enterprise Docker registry](#). Additionally, you can create an [image stream](#) that points to the image, either in your Docker registry or at the external location. Your OpenShift Enterprise resources can then reference the ImageStream. You can find [example image stream definitions](#) for all the provided OpenShift Enterprise images.

2.4.4. Configuration

The Perl image supports a number of environment variables which can be set to control the configuration and behavior of the Perl runtime.

To set these environment variables, you can place them into a [.sti/environment](#) file inside your source code repository, or define them in [the environment section](#) of the build configuration's **sourceStrategy** definition.

Table 2.2. Perl Environment Variables

Variable name	Description
ENABLE_CPAN_TEST	This variable installs all the cpan modules and runs their tests. By default, the testing of the modules is turned off.
CPAN_MIRROR	This variable specifies a mirror URL which cpanminus uses to install dependencies. By default, this URL is not specified.
PERL_APACHE2_RELOAD	Set this to true to enable automatic reloading of modified Perl modules. By default, automatic reloading is turned off.

2.4.5. Accessing Logs

Access logs are streamed to standard output and as such they can be viewed using the **oc logs** command. Error logs are stored in the **/tmp/error_log** file, which can be viewed using the **oc rsh** command to access the container.

2.4.6. Hot Deploying

Hot deployment allows you to quickly make and deploy changes to your application without having to generate a new S2I build. To enable hot deployment in this image, you must set the **PERL_APACHE2_RELOAD** environment variable to **true**. For example, see the **oc new-app** command. You can use the **oc env** command to update environment variables of existing objects.

**WARNING**

You should only use this option while developing or debugging; it is not recommended to turn this on in your production environment.

To change your source code in a running pod, use the **oc rsh** command to enter the container:

```
$ oc rsh <pod_id>
```

After you enter into the running container, your current directory is set to **/opt/app-root/src**, where the source code is located.

2.5. PHP

2.5.1. Overview

OpenShift Enterprise provides [S2I](#) enabled PHP images for building and running PHP applications. The PHP S2I builder image assembles your application source with any required dependencies to create a new image containing your PHP application. This resulting image can be run either by OpenShift Enterprise or by Docker.

2.5.2. Versions

Currently, OpenShift Enterprise provides version [5.5](#) and [5.6](#) of PHP.

2.5.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry using:

```
$ docker pull registry.access.redhat.com/openshift3/php-55-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/php-55-centos7
```

To use these images, you can either access them directly from these [image registries](#), or push them into your [OpenShift Enterprise Docker registry](#). Additionally, you can create an [image stream](#) that points to the image, either in your Docker registry or at the external

location. Your OpenShift Enterprise resources can then reference the image stream.

You can find [example image stream definitions](#) for all the provided OpenShift Enterprise images.

2.5.4. Configuration

The PHP image supports a number of environment variables which can be set to control the configuration and behavior of the PHP runtime.

To set these environment variables, you can place them into a [.sti/environment](#) file inside your source code repository, or define them in [the environment section](#) of the build configuration's **sourceStrategy** definition.

The following environment variables set their equivalent property value in the **php.ini** file:

Table 2.3. PHP Environment Variables

Variable name	Description	Default
ERROR_REPORTING	Informs PHP of the errors, warnings, and notices for which you would like it to take action.	E_ALL & ~E_NOTICE
DISPLAY_ERRORS	Controls if and where PHP outputs errors, notices, and warnings.	ON
DISPLAY_STARTUP_ERRORS	Causes any display errors that occur during PHP's startup sequence to be handled separately from display errors.	OFF
TRACK_ERRORS	Stores the last error/warning message in \$php_errormsg (boolean).	OFF
HTML_ERRORS	Links errors to documentation that is related to the error.	ON
INCLUDE_PATH	Path for PHP source files.	./opt/openshift/src:/opt/rh/php55/root/usr/share/pear
SESSION_PATH	Location for session data files.	/tmp/sessions

The following environment variable sets its equivalent property value in the **opcache.ini** file:

Table 2.4. Additional PHP settings

Variable name	Description	Default
OPCACHE_MEMORY_CONSUMPTION	The OPcache shared memory storage size.	16M
OPCACHE_REVALIDATE_FREQ	How often to check script time stamps for updates, in seconds. 0 results in OPcache checking for updates on every request.	2

You can also override the entire directory used to load the PHP configuration by setting:

Table 2.5. Additional PHP settings

Variable name	Description
PHPRC	Sets the path to the <i>php.ini</i> file.
PHP_INI_SCAN_DIR	Path to scan for additional <i>.ini</i> configuration files

2.5.4.1. Apache Configuration

If the **DocumentRoot** of the application is nested in the source directory */opt/openshift/src*, you can provide your own *.htaccess* file to override the default Apache behavior and specify how application requests should be handled. The *.htaccess* file must be located at the root of the application source.

2.5.5. Accessing Logs

Access logs are streamed to standard out and as such they can be viewed using the [oc logs](#) command. Error logs are stored in the */tmp/error_log* file, which can be viewed using the [oc rsh](#) command to access the container.

2.5.6. Hot Deploying

Hot deployment allows you to quickly make and deploy changes to your application without having to generate a new S2I build. In order to immediately pick up changes made in your application source code, you must run your built image with the **OPCACHE_REVALIDATE_FREQ=0** environment variable.

For example, see the [oc new-app](#) command. You can use the [oc env](#) command to update environment variables of existing objects.



WARNING

You should only use this option while developing or debugging; it is not recommended to turn this on in your production environment.

To change your source code in a running pod, use the **oc rsh** command to enter the container:

```
$ oc rsh <pod_id>
```

After you enter into the running container, your current directory is set to **/opt/app-root/src**, where the source code is located.

2.6. PYTHON

2.6.1. Overview

OpenShift Enterprise provides [S2I](#) enabled Python images for building and running Python applications. The Python S2I builder image assembles your application source with any required dependencies to create a new image containing your Python application. This resulting image can be run either by OpenShift Enterprise or by Docker.

2.6.2. Versions

Currently, OpenShift Enterprise provides versions [2.7](#), [3.3](#), [3.4](#), and [3.5](#) of Python.

2.6.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry using:

```
$ docker pull registry.access.redhat.com/openshift3/python-33-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/python-33-centos7
```

To use these images, you can either access them directly from these [image registries](#), or push them into your [OpenShift Enterprise Docker registry](#). Additionally, you can create an [image stream](#) that points to the image, either in your Docker registry or at the external location. Your OpenShift Enterprise resources can then reference the ImageStream. You can find [example image stream definitions](#) for all the provided OpenShift Enterprise images.

2.6.4. Configuration

The Python image supports a number of environment variables which can be set to control the configuration and behavior of the Python runtime.

To set these environment variables, you can place them into a [.sti/environment](#) file inside your source code repository, or define them in the [environment](#) section of the build configuration's `sourceStrategy` definition.

Table 2.6. Python Environment Variables

Variable name	Description
APP_FILE	This variable specifies the file name passed to the python interpreter which is responsible for launching the application. This variable is set to app.py by default.
APP_MODULE	This variable specifies the WSGI callable. It follows the pattern <code>\$(MODULE_NAME):\$(VARIABLE_NAME)</code> , where the module name is a full dotted path and the variable name refers to a function inside the specified module. If you use setup.py for installing the application, then the module name can be read from that file and the variable defaults to application . There is an example setup-test-app available.
APP_CONFIG	This variable indicates the path to a valid Python file with a gunicorn configuration .
DISABLE_COLLECTSTATIC	Set it to a nonempty value to inhibit the execution of manage.py collectstatic during the build. Only affects Django projects.
DISABLE_MIGRATE	Set it to a nonempty value to inhibit the execution of manage.py migrate when the produced image is run. Only affects Django projects.

2.6.5. Hot Deploying

Hot deployment allows you to quickly make and deploy changes to your application without having to generate a new S2I build. If you are using Django, hot deployment works out of the box.

To enable hot deployment while using Gunicorn, ensure you have a Gunicorn configuration file inside your repository with the [reload option](#) set to **true**. Specify your configuration file using the **APP_CONFIG** environment variable. For example, see the [oc new-app](#) command. You can use the [oc env](#) command to update environment variables of existing objects.



WARNING

You should only use this option while developing or debugging; it is not recommended to turn this on in your production environment.

To change your source code in a running pod, use the **oc rsh** command to enter the container:

```
$ oc rsh <pod_id>
```

After you enter into the running container, your current directory is set to ***/opt/app-root/src***, where the source code is located.

CHAPTER 3. DATABASE IMAGES

3.1. OVERVIEW

This topic group includes information on the different database images available for OpenShift Enterprise users.



NOTE

Enabling clustering for database images is currently in Technology Preview and not intended for production use.

3.2. MYSQL

3.2.1. Overview

OpenShift Enterprise provides a Docker image for running MySQL. This image can provide database services based on username, password, and database name settings provided via configuration.

3.2.2. Versions

Currently, OpenShift Enterprise provides version [5.5](#) and [5.6](#) of MySQL.

3.2.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry via:

```
$ docker pull registry.access.redhat.com/openshift3/mysql-55-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/mysql-55-centos7
```

To use these images, you can either access them directly from these registries or push them into your OpenShift Enterprise Docker registry. Additionally, you can create an ImageStream that points to the image, either in your Docker registry or at the external location. Your OpenShift Enterprise resources can then reference the ImageStream. You can find [example](#) ImageStream definitions for all the provided OpenShift Enterprise images.

3.2.4. Configuration and Usage

3.2.4.1. Initializing the Database

The first time you use the shared volume, the database is created along with the database administrator user and the MySQL root user (if you specify the **MYSQL_ROOT_PASSWORD** environment variable). Afterwards, the MySQL daemon starts up. If you are re-attaching the volume to another container, then the database, database user, and the administrator user are not created, and the MySQL daemon starts.

The following command creates a new database [pod](#) with MySQL running in a container:

```
$ oc new-app -e \
    MYSQL_USER=<username>,MYSQL_PASSWORD=<password>,MYSQL_DATABASE=
<database_name> \
    registry.access.redhat.com/openshift3/mysql-55-rhel7
```

3.2.4.2. Running MySQL Commands in Containers

OpenShift Enterprise uses [Software Collections](#) (SCLs) to install and launch MySQL. If you want to execute a MySQL command inside of a running container (for debugging), you must invoke it using `bash`.

To do so, first identify the name of the pod. For example, you can view the list of pods in your current project:

```
$ oc get pods
```

Then, open a remote shell session to the pod:

```
$ oc rsh <pod>
```

When you enter the container, the required SCL is automatically enabled.

You can now run the **mysql** command from the bash shell to start a MySQL interactive session and perform normal MySQL operations. For example, to authenticate as the database user:

```
bash-4.2$ mysql -u $MYSQL_USER -p$MYSQL_PASSWORD -h $HOSTNAME
$MYSQL_DATABASE
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.37 MySQL Community Server (GPL)
...
mysql>
```

When you are finished, enter **quit** or **exit** to leave the MySQL session.

3.2.4.3. Environment Variables

The MySQL user name, password, and database name must be configured with the following environment variables:

Table 3.1. MySQL Environment Variables

Variable Name	Description
MYSQL_USER	Specifies the user name for the database user that is created for use by your application.
MYSQL_PASSWORD	Password for the MYSQL_USER .
MYSQL_DATABASE	Name of the database to which MYSQL_USER has full rights.
MYSQL_ROOT_PASSWORD	Optional password for the root user. If this is not set, then remote login to the root account is not possible. Local connections from within the container are always permitted without a password.
MYSQL_SERVICE_HOST	Service host variable automatically created by Kubernetes.
MYSQL_SERVICE_PORT	Service port variable automatically created by Kubernetes.

**WARNING**

You must specify the user name, password, and database name. If you do not specify all three, the pod will fail to start and OpenShift Enterprise will continuously try to restart it.

MySQL settings can be configured with the following environment variables:

Table 3.2. Additional MySQL Settings

Variable Name	Description	Default
MYSQL_LOWER_CASE_TABLE_NAMES	Sets how the table names are stored and compared.	0
MYSQL_MAX_CONNECTIONS	The maximum permitted number of simultaneous client connections.	151
MYSQL_FT_MIN_WORD_LENGTH	The minimum length of the word to be included in a FULLTEXT index.	4
MYSQL_FT_MAX_WORD_LENGTH	The maximum length of the word to be included in a FULLTEXT index.	20

Variable Name	Description	Default
MYSQL_AIO	Controls the innodb_use_native_aio setting value if the native AIO is broken.	1

3.2.4.4. Volume Mount Points

The MySQL image can be run with mounted volumes to enable persistent storage for the database:

- **/var/lib/mysql/data** - This is the data directory where MySQL stores database files.

3.2.4.5. Changing Passwords

Passwords are part of the image configuration, therefore the only supported method to change passwords for the database user (**MYSQL_USER**) and **root** user is by changing the environment variables **MYSQL_PASSWORD** and **MYSQL_ROOT_PASSWORD**, respectively.

You can view the current passwords by viewing the pod or deployment configuration in the web console or by listing the environment variables with the CLI:

```
$ oc env pod <pod_name> --list
```

Whenever **MYSQL_ROOT_PASSWORD** is set, it enables remote access for the **root** user with the given password, and whenever it is unset, remote access for the **root** user is disabled. This does not affect the regular user **MYSQL_USER**, who always has remote access. This also does not affect local access by the **root** user, who can always log in without a password in **localhost**.

Changing database passwords through SQL statements or any way other than through the environment variables aforementioned causes a mismatch between the values stored in the variables and the actual passwords. Whenever a database container starts, it resets the passwords to the values stored in the environment variables.

To change these passwords, update one or both of the desired environment variables for the related deployment configuration(s) using the **oc env** command. If multiple deployment configurations utilize these environment variables, for example in the case of an application created from a template, you must update the variables on each deployment configuration so that the passwords are in sync everywhere. This can be done all in the same command:

```
$ oc env dc <dc_name> [<dc_name_2> ...] \
  MYSQL_PASSWORD=<new_password> \
  MYSQL_ROOT_PASSWORD=<new_root_password>
```



IMPORTANT

Depending on your application, there may be other environment variables for passwords in other parts of the application that should also be updated to match. For example, there could be a more generic **DATABASE_USER** variable in a front-end pod that should match the database user's password. Ensure that passwords are in sync for all required environment variables per your application, otherwise your pods may fail to redeploy when triggered.

Updating the environment variables triggers the redeployment of the database server if you have a [configuration change trigger](#). Otherwise, you must manually start a new deployment in order to apply the password changes.

To verify that new passwords are in effect, first open a remote shell session to the running MySQL pod:

```
$ oc rsh <pod>
```

From the bash shell, verify the database user's new password:

```
bash-4.2$ mysql -u $MYSQL_USER -p<new_password> -h $HOSTNAME
$MYSQL_DATABASE -te "SELECT * FROM (SELECT database()) db CROSS JOIN
(SELECT user()) u"
```

If the password was changed correctly, you should see a table like this:

```
+-----+-----+
| database() | user() |
+-----+-----+
| sampled    | user0PG@172.17.42.1 |
+-----+-----+
```

To verify the **root** user's new password:

```
bash-4.2$ mysql -u root -p<new_root_password> -h $HOSTNAME $MYSQL_DATABASE
-te "SELECT * FROM (SELECT database()) db CROSS JOIN (SELECT user()) u"
```

If the password was changed correctly, you should see a table like this:

```
+-----+-----+
| database() | user() |
+-----+-----+
| sampled    | root@172.17.42.1 |
+-----+-----+
```

3.2.5. Creating a Database Service from a Template

OpenShift Enterprise provides a [template](#) to make creating a new database service easy. The template provides parameter fields to define all the mandatory environment variables (user, password, database name, etc) with predefined defaults including auto-generation of password values. It will also define both a [deployment configuration](#) and a [service](#).

The MySQL templates should have been registered in the default **openshift** project by your cluster administrator during the initial cluster setup. See [Loading the Default Image Streams and Templates](#) for more details, if required.

There are two templates available:

- **mysql-ephemeral** is for development or testing purposes only because it uses ephemeral storage for the database content. This means that if the database pod is restarted for any reason, such as the pod being moved to another node or the deployment configuration being updated and triggering a redeploy, all data will be lost.
- **mysql-persistent** uses a persistent volume store for the database data which means the data will survive a pod restart. Using persistent volumes requires a persistent volume pool be defined in the OpenShift Enterprise deployment. Cluster administrator instructions for setting up the pool are located [here](#).

You can find instructions for instantiating templates by following these [instructions](#).

Once you have instantiated the service, you can copy the user name, password, and database name environment variables into a deployment configuration for another component that intends to access the database. That component can then access the database via the service that was defined.

3.2.6. Using MySQL Replication



NOTE

Enabling clustering for database images is currently in Technology Preview and not intended for production use.

Red Hat provides a proof-of-concept [template](#) for MySQL master-slave replication (clustering); you can obtain the [example template from GitHub](#).

To upload the example template into the current project's template library:

```
$ oc create -f \
https://raw.githubusercontent.com/openshift/mysql/master/5.5/examples/replica/mysql_replica.json
```

The following sections detail the objects defined in the example template and describe how they work together to start a cluster of MySQL servers implementing master-slave replication. This is the recommended replication strategy for MySQL.

3.2.6.1. Creating the Deployment Configuration for the MySQL Master

To set up MySQL replication, a [deployment configuration](#) is defined in the example template that defines a [replication controller](#). For MySQL master-slave replication, two deployment configurations are needed. One deployment configuration defines the MySQL *master* server and second the MySQL *slave* servers.

To tell a MySQL server to act as the master, the **command** field in the container's definition in the deployment configuration must be set to **run-mysqld-master**. This script acts as an alternative entrypoint for the MySQL image and configures the MySQL server to run as the

master in replication.

MySQL replication requires a special user that relays data between the master and slaves. The following environment variables are defined in the template for this purpose:

Variable Name	Description	Default
MYSQL_MASTER_USER	The user name of the replication user	master
MYSQL_MASTER_PASSWORD	The password for the replication user	generated

Example 3.1. MySQL Master Deployment Configuration Object Definition in the Example Template

```
kind: "DeploymentConfig"
apiVersion: "v1"
metadata:
  name: "mysql-master"
spec:
  strategy:
    type: "Recreate"
  triggers:
    - type: "ConfigChange"
  replicas: 1
  selector:
    name: "mysql-master"
  template:
    metadata:
      labels:
        name: "mysql-master"
    spec:
      volumes:
        - name: "mysql-master-data"
          persistentVolumeClaim:
            claimName: "mysql-master"
      containers:
        - name: "server"
          image: "openshift/mysql-55-centos7"
          command:
            - "run-mysqld-master"
          ports:
            - containerPort: 3306
              protocol: "TCP"
          env:
            - name: "MYSQL_MASTER_USER"
              value: "${MYSQL_MASTER_USER}"
            - name: "MYSQL_MASTER_PASSWORD"
              value: "${MYSQL_MASTER_PASSWORD}"
            - name: "MYSQL_USER"
              value: "${MYSQL_USER}"
            - name: "MYSQL_PASSWORD"
```

```

        value: "${MYSQL_PASSWORD}"
      - name: "MYSQL_DATABASE"
        value: "${MYSQL_DATABASE}"
      - name: "MYSQL_ROOT_PASSWORD"
        value: "${MYSQL_ROOT_PASSWORD}"
    volumeMounts:
      - name: "mysql-master-data"
        mountPath: "/var/lib/mysql/data"
    resources: {}
    terminationMessagePath: "/dev/termination-log"
    imagePullPolicy: "IfNotPresent"
    securityContext:
      capabilities: {}
      privileged: false
    restartPolicy: "Always"
    dnsPolicy: "ClusterFirst"

```

Since we claimed a persistent volume in this deployment configuration to have all data persisted for the MySQL master server, you must ask your cluster administrator to create a persistent volume that you can claim the storage from.

After the deployment configuration is created and the pod with MySQL master server is started, it will create the database defined by **MYSQL_DATABASE** and configure the server to replicate this database to slaves.

The example provided defines only one replica of the MySQL master server. This causes OpenShift Enterprise to start only one instance of the server. Multiple instances (multi-master) is not supported and therefore you can not scale this replication controller.

To replicate the database created by the [MySQL master](#), a deployment configuration is defined in the template. This deployment configuration creates a replication controller that launches the MySQL image with the **command** field set to **run-mysqld-slave**. This alternative entrypoints skips the initialization of the database and configures the MySQL server to connect to the **mysql-master** service, which is also defined in example template.

Example 3.2. MySQL Slave Deployment Configuration Object Definition in the Example Template

```

kind: "DeploymentConfig"
apiVersion: "v1"
metadata:
  name: "mysql-slave"
spec:
  strategy:
    type: "Recreate"
  triggers:
    - type: "ConfigChange"
  replicas: 1
  selector:
    name: "mysql-slave"
  template:
    metadata:
      labels:
        name: "mysql-slave"
    spec:

```

```

containers:
  - name: "server"
    image: "openshift/mysql-55-centos7"
    command:
      - "run-mysqld-slave"
    ports:
      - containerPort: 3306
        protocol: "TCP"
    env:
      - name: "MYSQL_MASTER_USER"
        value: "${MYSQL_MASTER_USER}"
      - name: "MYSQL_MASTER_PASSWORD"
        value: "${MYSQL_MASTER_PASSWORD}"
      - name: "MYSQL_DATABASE"
        value: "${MYSQL_DATABASE}"
    resources: {}
    terminationMessagePath: "/dev/termination-log"
    imagePullPolicy: "IfNotPresent"
    securityContext:
      capabilities: {}
      privileged: false
    restartPolicy: "Always"
    dnsPolicy: "ClusterFirst"

```

This example deployment configuration starts the replication controller with the initial number of replicas set to **1**. You can [scale this replication controller](#) in both directions, up to the resources capacity of your account.

3.2.6.2. Creating a Headless Service

The pods created by the MySQL slave replication controller must reach the MySQL master server in order to register for replication. The example template defines a headless service named **mysql-master** for this purpose. This service is not used only for replication, but the clients can also send the queries to **mysql-master:3306** as the MySQL host.

To have a headless service, the **portName** parameter in the service definition is set to **None**. Then you can use a DNS query to get a list of the pod IP addresses that represents the current endpoints for this service.

Example 3.3. Headless Service Object Definition in the Example Template

```

kind: "Service"
apiVersion: "v1"
metadata:
  name: "mysql-master"
  labels:
    name: "mysql-master"
spec:
  ports:
    - protocol: "TCP"
      port: 3306
      targetPort: 3306
      nodePort: 0
  selector:

```

```
    name: "mysql-master"
    portalIP: "None"
    type: "ClusterIP"
    sessionAffinity: "None"
    status:
      loadBalancer: {}
```

3.2.6.3. Scaling the MySQL Slaves

To [increase the number of members](#) in the cluster:

```
$ oc scale rc mysql-slave-1 --replicas=<number>
```

This tells [the replication controller](#) to create a new MySQL slave pod. When a new slave is created, the slave endpoint first attempts to contact the **mysql-master** service and register itself to the replication set. Once that is done, the MySQL master server sends the slave the replicated database.

When scaling down, the MySQL slave is shut down and, because the slave does not have any persistent storage defined, all data on the slave is lost. The MySQL master server then discovers that the slave is not reachable anymore, and it automatically removes it from the replication.

3.2.7. Troubleshooting

This section describes some troubles you might encounter and presents possible resolutions.

3.2.7.1. Linux Native AIO Failure

Symptom

The MySQL container fails to start and the logs show something like:

```
151113 5:06:56 InnoDB: Using Linux native AIO
151113 5:06:56 InnoDB: Warning: io_setup() failed with EAGAIN. Will make
5 attempts before giving up.
InnoDB: Warning: io_setup() attempt 1 failed.
InnoDB: Warning: io_setup() attempt 2 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 3 failed.
InnoDB: Warning: io_setup() attempt 4 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 5 failed.
151113 5:06:59 InnoDB: Error: io_setup() failed with EAGAIN after 5
attempts.
InnoDB: You can disable Linux Native AIO by setting innodb_use_native_aio
= 0 in my.cnf
151113 5:06:59 InnoDB: Fatal error: cannot initialize AIO sub-system
151113 5:06:59 [ERROR] Plugin 'InnoDB' init function returned error.
151113 5:06:59 [ERROR] Plugin 'InnoDB' registration as a STORAGE ENGINE
```

```
failed.
151113 5:06:59 [ERROR] Unknown/unsupported storage engine: InnoDB
151113 5:06:59 [ERROR] Aborting
```

Explanation

MySQL's storage engine was unable to use the kernel's AIO (Asynchronous I/O) facilities due to resource limits.

Resolution

1. Turn off AIO usage entirely, by setting environment variable **MYSQL_AIO** to have value **0**. On subsequent deployments, this arranges for the MySQL configuration variable **innodb_use_native_aio** to have value **0**.
2. Increase the **aio-max-nr** kernel resource. The following example examines the current value of **aio-max-nr** and doubles it.

```
$ sysctl fs.aio-max-nr
fs.aio-max-nr = 1048576
# sysctl -w fs.aio-max-nr=2097152
```

This is a per-node resolution and lasts until the next node reboot.

3.3. POSTGRESQL

3.3.1. Overview

OpenShift Enterprise provides a Docker image for running PostgreSQL. This image can provide database services based on username, password, and database name settings provided via configuration.

3.3.2. Versions

Currently, OpenShift Enterprise supports version [9.2](#) and [9.4](#) of PostgreSQL.

3.3.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry via:

```
$ docker pull registry.access.redhat.com/openshift3/postgresql-92-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/postgresql-92-centos7
```

or

```
$ docker pull centos/postgresql-94-centos7
```

To use these images, you can either access them directly from these registries or push them into your OpenShift Enterprise Docker registry. Additionally, you can create an ImageStream that points to the image, either in your Docker registry or at the external location. Your OpenShift Enterprise resources can then reference the ImageStream. You can find [example](#) ImageStream definitions for all the provided OpenShift Enterprise images.

3.3.4. Configuration and Usage

3.3.4.1. Initializing the Database

The first time you use the shared volume, the database is created along with the database administrator user and the PostgreSQL postgres user (if you specify the **POSTGRESQL_ADMIN_PASSWORD** environment variable). Afterwards, the PostgreSQL daemon starts up. If you are re-attaching the volume to another container, then the database, the database user, and the administrator user are not created, and the PostgreSQL daemon starts.

The following command creates a new database [pod](#) with PostgreSQL running in a container:

```
$ oc new-app -e \
    POSTGRESQL_USER=<username>,POSTGRESQL_PASSWORD=
    <password>,POSTGRESQL_DATABASE=<database_name> \
    registry.access.redhat.com/rhsc1/postgresql-94-rhel7
```

3.3.4.2. Running PostgreSQL Commands in Containers

OpenShift Enterprise uses [Software Collections](#) (SCLs) to install and launch PostgreSQL. If you want to execute a PostgreSQL command inside of a running container (for debugging), you must invoke it using `bash`.

To do so, first identify the name of the running PostgreSQL pod. For example, you can view the list of pods in your current project:

```
$ oc get pods
```

Then, open a remote shell session to the desired pod:

```
$ oc rsh <pod>
```

When you enter the container, the required SCL is automatically enabled.

You can now run the **psql** command from the bash shell to start a PostgreSQL interactive session and perform normal PostgreSQL operations. For example, to authenticate as the database user:

```
bash-4.2$ PGPASSWORD=$POSTGRESQL_PASSWORD psql -h postgresql
```

```
$POSTGRESQL_DATABASE $POSTGRESQL_USER
psql (9.2.8)
Type "help" for help.

default=>
```

When you are finished, enter `\q` to leave the PostgreSQL session.

3.3.4.3. Environment Variables

The PostgreSQL user name, password, and database name must be configured with the following environment variables:

Table 3.3. PostgreSQL Environment Variables

Variable Name	Description
POSTGRESQL_USER	User name for the PostgreSQL account to be created. This user has full rights to the database.
POSTGRESQL_PASSWORD	Password for the user account.
POSTGRESQL_DATABASE	Database name.
POSTGRESQL_ADMIN_PASSWORD	Optional password for the postgres administrator user. If this is not set, then remote login to the postgres account is not possible. Local connections from within the container are always permitted without a password.



WARNING

You must specify the user name, password, and database name. If you do not specify all three, the pod will fail to start and OpenShift Enterprise will continuously try to restart it.

PostgreSQL settings can be configured with the following environment variables:

Table 3.4. Additional PostgreSQL settings

Variable Name	Description	Default
POSTGRESQL_MAX_CONNECTIONS	The maximum number of client connections allowed. This also sets the maximum number of prepared transactions.	100

Variable Name	Description	Default
POSTGRESQL_SHARED_BUFFERS	Configures how much memory is dedicated to PostgreSQL for caching data.	32M

3.3.4.4. Volume Mount Points

The PostgreSQL image can be run with mounted volumes to enable persistent storage for the database:

- **/var/lib/pgsql/data** - This is the database cluster directory where PostgreSQL stores database files.

3.3.4.5. Changing Passwords

Passwords are part of the image configuration, therefore the only supported method to change passwords for the database user (**POSTGRESQL_USER**) and **postgres** administrator user is by changing the environment variables **POSTGRESQL_PASSWORD** and **POSTGRESQL_ADMIN_PASSWORD**, respectively.

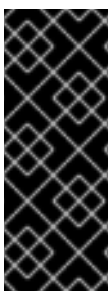
You can view the current passwords by viewing the pod or deployment configuration in the web console or by listing the environment variables with the CLI:

```
$ oc env pod <pod_name> --list
```

Changing database passwords through SQL statements or any way other than through the environment variables aforementioned will cause a mismatch between the values stored in the variables and the actual passwords. Whenever a database container starts, it resets the passwords to the values stored in the environment variables.

To change these passwords, update one or both of the desired environment variables for the related deployment configuration(s) using the **oc env** command. If multiple deployment configurations utilize these environment variables, for example in the case of an application created from a template, you must update the variables on each deployment configuration so that the passwords are in sync everywhere. This can be done all in the same command:

```
$ oc env dc <dc_name> [<dc_name_2> ...] \
  POSTGRESQL_PASSWORD=<new_password> \
  POSTGRESQL_ADMIN_PASSWORD=<new_admin_password>
```



IMPORTANT

Depending on your application, there may be other environment variables for passwords in other parts of the application that should also be updated to match. For example, there could be a more generic **DATABASE_USER** variable in a front-end pod that should match the database user's password. Ensure that passwords are in sync for all required environment variables per your application, otherwise your pods may fail to redeploy when triggered.

Updating the environment variables triggers the redeployment of the database server if you have a [configuration change trigger](#). Otherwise, you must manually start a new deployment in order to apply the password changes.

To verify that new passwords are in effect, first open a remote shell session to the running PostgreSQL pod:

```
$ oc rsh <pod>
```

From the bash shell, verify the database user's new password:

```
bash-4.2$ PGPASSWORD=<new_password> psql -h postgresql
$POSTGRESQL_DATABASE $POSTGRESQL_USER -c "SELECT * FROM (SELECT
current_database()) cdb CROSS JOIN (SELECT current_user) cu"
```

If the password was changed correctly, you should see a table like this:

```
current_database | current_user
-----+-----
default          | django
(1 row)
```

From the bash shell, verify the **postgres** administrator user's new password:

```
bash-4.2$ PGPASSWORD=<new_admin_password> psql -h postgresql
$POSTGRESQL_DATABASE postgres -c "SELECT * FROM (SELECT
current_database()) cdb CROSS JOIN (SELECT current_user) cu"
```

If the password was changed correctly, you should see a table like this:

```
current_database | current_user
-----+-----
default          | postgres
(1 row)
```

3.3.5. Creating a Database Service from a Template

OpenShift provides a [template](#) to make creating a new database service easy. The template provides parameter fields to define all the mandatory environment variables (user, password, database name, etc) with predefined defaults including auto-generation of password values. It will also define both a [deployment configuration](#) and a [service](#).

The PostgreSQL templates should have been registered in the default **openshift** project by your cluster administrator during the initial cluster setup. See [Loading the Default Image Streams and Templates](#) for more details, if required.

There are two templates available:

- **PostgreSQL-ephemeral** is for development or testing purposes only because it uses ephemeral storage for the database content. This means that if the database pod is restarted for any reason, such as the pod being moved to another node or the deployment configuration being updated and triggering a redeploy, all data will be lost.

- **PostgreSQL-persistent** uses a persistent volume store for the database data which means the data will survive a pod restart. Using persistent volumes requires a persistent volume pool be defined in the OpenShift Enterprise deployment. Cluster administrator instructions for setting up the pool are located [here](#).

You can find instructions for instantiating templates by following these [instructions](#).

Once you have instantiated the service, you can copy the user name, password, and database name environment variables into a deployment configuration for another component that intends to access the database. That component can then access the database via the service that was defined.

3.4. MONGODB

3.4.1. Overview

OpenShift Enterprise provides a Docker image for running MongoDB. This image can provide database services based on username, password, and database name settings provided via configuration.

3.4.2. Versions

Currently, OpenShift Enterprise provides version [2.4](#) and [2.6](#) of MongoDB.

3.4.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry via:

```
$ docker pull registry.access.redhat.com/openshift3/mongodb-24-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/mongodb-24-centos7
```

```
$ docker pull centos/mongodb-26-centos7
```

To use these images, you can either access them directly from these registries or push them into your OpenShift Enterprise Docker registry. Additionally, you can create an ImageStream that points to the image, either in your Docker registry or at the external location. Your OpenShift Enterprise resources can then reference the ImageStream. You can find [example](#) ImageStream definitions for all the provided OpenShift Enterprise images.

3.4.4. Configuration and Usage

3.4.4.1. Initializing the Database

The first time you use the shared volume, the database is created along with the database administrator user. Afterwards, the MongoDB daemon starts up. If you are re-attaching the volume to another container, then the database, database user, and the administrator user are not created, and the MongoDB daemon starts.

The following command creates a new database `pod` with MongoDB running in a container:

```
$ oc new-app -e \
  MONGODB_USER=<username>,MONGODB_PASSWORD=<password>,MONGODB_DATABASE=
  <database_name>,MONGODB_ADMIN_PASSWORD=<admin_password> \
  registry.access.redhat.com/rhsc1/mongodb-26-rhel7
```

3.4.4.2. Running MongoDB Commands in Containers

OpenShift Enterprise uses [Software Collections](#) (SCLs) to install and launch MongoDB. If you want to execute a MongoDB command inside of a running container (for debugging), you must invoke it using `bash`.

To do so, first identify the name of the running MongoDB pod. For example, you can view the list of pods in your current project:

```
$ oc get pods
```

Then, open a remote shell session to the desired pod:

```
$ oc rsh <pod>
```

When you enter the container, the required SCL is automatically enabled.

You can now run **mongo** commands from the bash shell to start a MongoDB interactive session and perform normal MongoDB operations. For example, to switch to the **sampledb** database and authenticate as the database user:

```
bash-4.2$ mongo -u $MONGODB_USER -p $MONGODB_PASSWORD $MONGODB_DATABASE
MongoDB shell version: 2.4.9
connecting to: sampledb
>
```

When you are finished, press **CTRL+D** to leave the MongoDB session.

3.4.4.3. Environment Variables

The MongoDB user name, password, database name, and **admin** password must be configured with the following environment variables:

Table 3.5. MongoDB Environment Variables

Variable Name	Description
MONGODB_USER	User name for MongoDB account to be created.

Variable Name	Description
MONGODB_PASSWORD	Password for the user account.
MONGODB_DATABASE	Database name.
MONGODB_ADMIN_PASSWORD	Password for the admin user.

**WARNING**

You must specify the user name, password, database name, and **admin** password. If you do not specify all four, the pod will fail to start and OpenShift Enterprise will continuously try to restart it.

**NOTE**

The administrator user name is set to **admin** and you must specify its password by setting the **MONGODB_ADMIN_PASSWORD** environment variable. This process is done upon database initialization.

MongoDB settings can be configured with the following environment variables:

Table 3.6. Additional MongoDB Settings

Variable Name	Description	Default
MONGODB_NOPREALLOC	Disable data file preallocation.	true
MONGODB_SMALLFILES	Set MongoDB to use a smaller default data file size.	true
MONGODB_QUIET	Runs MongoDB in a quiet mode that attempts to limit the amount of output.	true

3.4.4.4. Volume Mount Points

The MongoDB image can be run with mounted volumes to enable persistent storage for the database:

- **/var/lib/mongodb** - This is the database directory where MongoDB stores database files.

3.4.4.5. Changing Passwords

Passwords are part of the image configuration, therefore the only supported method to change passwords for the database user (**MONGODB_USER**) and **admin** user is by changing the environment variables **MONGODB_PASSWORD** and **MONGODB_ADMIN_PASSWORD**, respectively.

You can view the current passwords by viewing the pod or deployment configuration in the web console or by listing the environment variables with the CLI:

```
$ oc env pod <pod_name> --list
```

Changing database passwords directly in MongoDB causes a mismatch between the values stored in the variables and the actual passwords. Whenever a database container starts, it resets the passwords to the values stored in the environment variables.

To change these passwords, update one or both of the desired environment variables for the related deployment configuration(s) using the **oc env** command. If multiple deployment configurations utilize these environment variables, for example in the case of an application created from a template, you must update the variables on each deployment configuration so that the passwords are in sync everywhere. This can be done all in the same command:

```
$ oc env dc <dc_name> [<dc_name_2> ...] \
  MONGODB_PASSWORD=<new_password> \
  MONGODB_ADMIN_PASSWORD=<new_admin_password>
```



IMPORTANT

Depending on your application, there may be other environment variables for passwords in other parts of the application that should also be updated to match. For example, there could be a more generic **DATABASE_USER** variable in a front-end pod that should match the database user's password. Ensure that passwords are in sync for all required environment variables per your application, otherwise your pods may fail to redeploy when triggered.

Updating the environment variables triggers the redeployment of the database server if you have a [configuration change trigger](#). Otherwise, you must manually start a new deployment in order to apply the password changes.

To verify that new passwords are in effect, first open a remote shell session to the running MongoDB pod:

```
$ oc rsh <pod>
```

From the bash shell, verify the database user's new password:

```
bash-4.2$ mongo -u $MONGODB_USER -p <new_password> $MONGODB_DATABASE --
eval "db.version()"
```

If the password was changed correctly, you should see output like this:

```
MongoDB shell version: 2.6.9
connecting to: sampled
2.6.9
```

To verify the **admin** user's new password:

```
bash-4.2$ mongo -u admin -p <new_admin_password> admin --eval  
"db.version()"
```

If the password was changed correctly, you should see output like this:

```
MongoDB shell version: 2.4.9  
connecting to: admin  
2.4.9
```

3.4.5. Creating a Database Service from a Template

OpenShift Enterprise provides a [template](#) to make creating a new database service easy. The template provides parameter fields to define all the mandatory environment variables (user, password, database name, etc) with predefined defaults including auto-generation of password values. It will also define both a [deployment configuration](#) and a [service](#).

The MongoDB templates should have been registered in the default **openshift** project by your cluster administrator during the initial cluster setup. See [Loading the Default Image Streams and Templates](#) for more details, if required.

There are two templates available:

- **mongodb-ephemeral** is for development/testing purposes only because it uses ephemeral storage for the database content. This means that if the database pod is restarted for any reason, such as the pod being moved to another node or the deployment configuration being updated and triggering a redeploy, all data will be lost.
- **mongodb-persistent** uses a persistent volume store for the database data which means the data will survive a pod restart. Using persistent volumes requires a persistent volume pool be defined in the OpenShift Enterprise deployment. Cluster administrator instructions for setting up the pool are located [here](#).

You can find instructions for instantiating templates by following these [instructions](#).

Once you have instantiated the service, you can copy the user name, password, and database name environment variables into a deployment configuration for another component that intends to access the database. That component can then access the database via the service that was defined.

3.4.6. Using MongoDB Replication



NOTE

Enabling clustering for database images is currently in Technology Preview and not intended for production use.

Red Hat provides a proof-of-concept [template](#) for MongoDB replication (clustering); you can obtain the [example template from GitHub](#).

For example, to upload the example template into the current project's template library:

```
$ oc create -f \
```

```
https://raw.githubusercontent.com/openshift/mongodb/master/2.4/examples/replica/mongodb-clustered.json
```



IMPORTANT

The example template does not use persistent storage. When you lose all members of the replication set, your data will be lost.

The following sections detail the objects defined in the example template and describe how they work together to start a cluster of MongoDB servers implementing master-slave replication and automated failover. This is the recommended replication strategy for MongoDB.

3.4.6.1. Creating the Deployment Configuration

To set up MongoDB replication, a [deployment configuration](#) is defined in the example template that defines a [replication controller](#). The replication controller manages the members of the MongoDB cluster.

To tell a MongoDB server that the member will be part of the cluster, additional environment variables are provided for the container defined in the replication controller pod template:

Variable Name	Description	Default
MONGODB_REPLICA_NAME	Specifies the name of the replication set.	rs0
MONGODB_KEYFILE_VALUE	See: Generate a Key File	generated

Example 3.4. Deployment Configuration Object Definition in the Example Template

```
kind: DeploymentConfig
apiVersion: v1
metadata:
  name: "${MONGODB_SERVICE_NAME}"
spec:
  strategy:
    type: Recreate
    resources: {}
  triggers:
    - type: ConfigChange
  replicas: 3
  selector:
    name: mongodb-replica
  template:
    metadata:
      labels:
```

```

      name: mongodb-replica
spec:
  containers:
  - name: member
    image: openshift/mongodb-24-centos7
    env:
      - name: MONGODB_USER
        value: "${MONGODB_USER}"
      - name: MONGODB_PASSWORD
        value: "${MONGODB_PASSWORD}"
      - name: MONGODB_DATABASE
        value: "${MONGODB_DATABASE}"
      - name: MONGODB_ADMIN_PASSWORD
        value: "${MONGODB_ADMIN_PASSWORD}"
      - name: MONGODB_REPLICA_NAME
        value: "${MONGODB_REPLICA_NAME}"
      - name: MONGODB_SERVICE_NAME
        value: "${MONGODB_SERVICE_NAME}"
      - name: MONGODB_KEYFILE_VALUE
        value: "${MONGODB_KEYFILE_VALUE}"
    ports:
      - containerPort: 27017
        protocol: TCP
  restartPolicy: Never
  dnsPolicy: ClusterFirst

```

After the deployment configuration is created and the pods with MongoDB cluster members are started, they will not be initialized. Instead, they start as part of the **rs0** replication set, as the value of **MONGODB_REPLICA_NAME** is set to **rs0** by default.

3.4.6.2. Creating the Service Pod

To initialize members created by [the deployment configuration](#), the pods are started with the **initiate** argument, which instructs the startup script to behave [slightly differently](#) than a regular, stand-alone MongoDB database.

Example 3.5. Deployment Configuration Object Definition in the Example Template

```

- kind: DeploymentConfig
  apiVersion: v1
  metadata:
    name: "${MONGODB_SERVICE_NAME}"
  spec:
    strategy:
      type: Recreate
      recreateParams:
        post:
          failurePolicy: Retry
          execNewPod:
            command: ["run-mongod", "initiate"]
            containerName: mongodb
            env:
              - name: MONGODB_INITIAL_REPLICA_COUNT

```

```

        value: '3'
triggers:
- type: ConfigChange
replicas: 3
selector:
  name: mongodb-replica
template:
  metadata:
    labels:
      name: mongodb-replica
  spec:
    containers:
    - name: mongodb
      image: openshift/mongodb-24-centos7
      readinessProbe:
        tcpSocket:
          port: 27017
        initialDelaySeconds: 15
        timeoutSeconds: 1
      env:
      - name: MONGODB_USER
        value: "${MONGODB_USER}"
      - name: MONGODB_PASSWORD
        value: "${MONGODB_PASSWORD}"
      - name: MONGODB_DATABASE
        value: "${MONGODB_DATABASE}"
      - name: MONGODB_ADMIN_PASSWORD
        value: "${MONGODB_ADMIN_PASSWORD}"
      - name: MONGODB_REPLICA_NAME
        value: "${MONGODB_REPLICA_NAME}"
      - name: MONGODB_SERVICE_NAME
        value: "${MONGODB_SERVICE_NAME}"
      - name: MONGODB_KEYFILE_VALUE
        value: "${MONGODB_KEYFILE_VALUE}"
    ports:
    - containerPort: 27017

```

3.4.6.3. Creating a Headless Service

The **initiate** argument in the [container specification above](#) instructs the container to first discover all running member pods within the MongoDB cluster. To achieve this, a *headless service* is defined named **mongodb** in the example template.

To have a headless service, the **portalIP** parameter in the service definition is set to **None**. Then you can use a DNS query to get a list of the pod IP addresses that represents the current endpoints for this service.

Example 3.6. Headless Service Object Definition in the Example Template

```

kind: "Service"
apiVersion: "v1"
metadata:
  name: "${MONGODB_SERVICE_NAME}"
  labels:

```

```
    name: "${MONGODB_SERVICE_NAME}"
spec:
  ports:
    - protocol: "TCP"
      port: 27017
      targetPort: 27017
      nodePort: 0
  selector:
    name: "mongodb-replica"
  portalIP: "None"
  type: "ClusterIP"
  sessionAffinity: "None"
status:
  loadBalancer: {}
```

3.4.6.4. Creating the Final Replication Set

When the script that runs as the container entrypoint has the IP addresses of all running MongoDB members, it creates a MongoDB replication set configuration where it lists all member IP addresses. It then initiates the replication set using **rs.initiate(config)**. The script waits until MongoDB elects the **PRIMARY** member of the cluster.

Once the **PRIMARY** member has been elected, the entrypoint script starts creating MongoDB users and databases.

Clients can then start using the MongoDB instance by sending the queries to the **mongodb** service. As this service is a headless service, they do not need to provide the IP address. Clients can use **mongodb:27017** for connections. The service then sends the query to one of the members in the replication set.

3.4.6.5. Scaling the MongoDB Replication Set

To [increase the number of members](#) in the cluster:

```
$ oc scale rc mongodb-1 --replicas=<number>
```

This tells [the replication controller](#) to create a new MongoDB member pod. When a new member is created, the member entrypoint first attempts to discover other running members in the cluster. It then chooses one and adds itself to the list of members. Once the replication configuration is updated, the other members replicate the data to a new pod and start a new election.

CHAPTER 4. DOCKER IMAGES

4.1. OVERVIEW

You can use arbitrary Docker images in your OpenShift Enterprise instance, for example those found on the [Docker Hub](#). For instructions on how to enable images to run with **USER** in the Dockerfile, see [Managing Security Context Constraints](#).

CHAPTER 5. OTHER IMAGES

5.1. OVERVIEW

This topic group includes information on other Docker images available for OpenShift Enterprise users.

5.2. JENKINS

5.2.1. Overview

OpenShift Enterprise provides a Docker image for running Jenkins. This image provides a Jenkins server instance which can be used to set up a basic flow for continuous testing, integration, and delivery.

This image also includes a sample Jenkins job which triggers a new build of a **BuildConfig** defined in OpenShift Enterprise, tests the output of that build, and then on successful build, retags the output to indicate the build is ready for production.

5.2.2. Versions

OpenShift Enterprise follows the [LTS](#) releases of Jenkins.

5.2.3. Images

This image comes in two flavors, depending on your needs:

- RHEL 7
- CentOS 7

RHEL 7 Based Image

The RHEL 7 image is available through Red Hat's subscription registry:

```
$ docker pull registry.access.redhat.com/openshift3/jenkins-1-rhel7
```

CentOS 7 Based Image

This image is available on DockerHub. To download it:

```
$ docker pull openshift/jenkins-1-centos7
```

To use these images, you can either access them directly from these registries or push them into your OpenShift Enterprise Docker registry. Additionally, you can create an ImageStream that points to the image, either in your Docker registry or at the external location. Your OpenShift Enterprise resources can then reference the ImageStream. You can find [example](#) ImageStream definitions for all the provided OpenShift Enterprise images.

5.2.4. Configuration and Usage

5.2.4.1. Initializing Jenkins

The first time you start Jenkins, the configuration is created along with the administrator user and password. The default login is **admin/password**. The default password can be configured by setting the **JENKINS_PASSWORD** environment variable.

The following command creates a new Jenkins [pod](#) with Jenkins running in a container:

```
$ oc new-app -e \
  JENKINS_PASSWORD=<password> \
  openshift/jenkins-1-centos7
```

5.2.4.2. Environment Variables

The Jenkins password can be configured with the following environment variable:

Table 5.1. Jenkins Environment Variables

Variable name	Description
JENKINS_PASSWORD	Password for the admin user.

5.2.4.3. Volume Mount Points

The Jenkins image can be run with mounted volumes to enable persistent storage for the configuration:

- **/var/lib/jenkins** - This is the data directory where Jenkins stores configuration files including job definitions.

5.2.5. Creating a Jenkins Service from a Template

[Templates](#) provide parameter fields to define all the environment variables (password) with predefined defaults. OpenShift Enterprise provides templates to make creating a new Jenkins service easy. The Jenkins templates should have been registered in the default **openshift** project by your cluster administrator during the initial cluster setup. See [Loading the Default Image Streams and Templates](#) for more details, if required.

The two available templates both define a [deployment configuration](#) and a [service](#), but differ in their storage strategy, which affects whether or not the Jenkins content persists across a pod restart.



NOTE

A pod may be restarted when it is moved to another node, or when an update of the deployment configuration triggers a redeployment.

- **jenkins-ephemeral** uses ephemeral storage. On pod restart, all data is lost. This template is useful for development or testing only.
- **jenkins-persistent** uses a persistent volume store. Data survives a pod restart. To use a persistent volume store, the cluster administrator must define a persistent volume pool in the OpenShift Enterprise deployment.

Once selected, you must [instantiate](#) the template to be able to use Jenkins.

5.2.6. Using Jenkins as a Source-To-Image builder

To customize the official OpenShift Enterprise Jenkins image, you have two options:

- Use Docker layering.
- Use the image as a Source-To-Image builder, described here.

You can use [S2I](#) to copy your custom Jenkins Jobs definitions, additional plugins or replace the provided **config.xml** file with your own, custom, configuration.

In order to include your modifications in the Jenkins image, you need to have a Git repository with the following directory structure:

plugins

This directory contains those binary Jenkins plugins you want to copy into Jenkins.

plugins.txt

This file lists the plugins you want to install (see the section above).

configuration/jobs

This directory contains the Jenkins job definitions.

configuration/config.xml

This file contains your custom Jenkins configuration.

The contents of the **configuration/** directory will be copied into the **/var/lib/jenkins/** directory, so you can also include additional files, such as **credentials.xml**, there.

The following is an example build configuration that customizes the Jenkins image in OpenShift Enterprise:

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source: ❶
    git:
      uri: https://github.com/custom/repository
      type: Git
  strategy: ❷
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jenkins:latest
        namespace: openshift
      type: Source
  output: ❸
    to:
      kind: ImageStreamTag
      name: custom-jenkins:latest
```

❶ The **source** field defines the source Git repository with the layout described above.

❷ The **strategy** field defines the original Jenkins image to use as a source image for the build.

- 3 The **output** field defines the resulting, customized Jenkins image you can use in deployment configuration instead of the official Jenkins image.

5.2.7. Using the Jenkins Kubernetes Plug-in to Run Jobs

The official OpenShift Enterprise Jenkins image includes the pre-installed [Kubernetes plug-in](#) that allows Jenkins slaves to be dynamically provisioned on multiple Docker hosts using Kubernetes and OpenShift Enterprise.

The Jenkins image entrypoint also provides auto-discovery and auto-configuration of the Kubernetes plug-ins by scanning the project Jenkins is deployed in for existing image streams with the label **role** set to **jenkins-slave**.

When an image stream with this label is found, the entrypoint generates the corresponding Kubernetes plug-in configuration so you can assign your Jenkins jobs to run in a pod running the Docker image provided by the image stream.

To use a Docker image as an Jenkins slave, the image must run the slave agent as an entrypoint. For more details about this, refer to the official [Jenkins documentation](#).

Alternatively, you can use [a provided OpenShift Enterprise template](#) to convert an existing image stream to a Jenkins slave.

5.2.8. Tutorial

For more details on the sample job included in this image, see this [tutorial](#).

CHAPTER 6. XPAAS MIDDLEWARE IMAGES

6.1. OVERVIEW

This topic group includes information on the different xPaaS middleware images available for OpenShift users.

6.2. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (JBOSS EAP) XPAAS IMAGES

6.2.1. Overview

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

6.2.2. Comparing the Product and Image

The xPaaS JBoss EAP images differ from the JBoss EAP product in several ways:

1. The image does not include the JBoss EAP Management Console used to manage xPaaS JBoss EAP images.
2. The JBoss EAP Management CLI is included in the xPaaS JBoss EAP image, but can only access the Management CLI of a container from within the pod.
3. Domain mode is not supported in the xPaaS JBoss EAP image. Instead, OpenShift manages the creation and distribution of applications in the containers.
4. The image's default root page is disabled. Deploy your own application to the root context as **ROOT.war**.
5. The EAP 6.4 image supports A-MQ for inter-pod and remote messaging. HornetQ is only supported for intra-pod messaging and only enabled when A-MQ is absent. The EAP 7 Beta image includes Artemis as a replacement for HornetQ.

For further information about JBoss EAP functionality and features independent from the JBoss EAP image, see the [JBoss EAP documentation](#) on the Red Hat Customer Portal.

6.2.3. Comparing the xPaaS JBoss EAP 6.4 and 7.0 Beta Images

Red Hat offers two xPaaS EAP images for use with OpenShift. The first is based on JBoss EAP 6.4 and the second is based on JBoss EAP 7 Beta. There are several differences between the two images:

JBoss Web is replaced by Undertow

- The xPaaS JBoss EAP 6.4 image uses JBoss Web.
- The xPaaS JBoss EAP 7 Beta image uses Undertow instead of JBoss Web. This change only affects users implementing custom JBoss Web Valves in their applications. Affected users must refer to the Red Hat JBoss EAP 7 Beta documentation for details about [migrating JBoss EAP Web Valve handlers](#).

HornetQ is replaced by Artemis

- The EAP 6.4 image only uses HornetQ for intra-pod messaging when A-MQ is absent.
- The EAP 7 Beta image uses Artemis instead of HornetQ. This change resulted in renaming the **HORNETQ_QUEUES** and **HORNETQ_TOPICS** environment variables to **MQ_QUEUES** and **MQ_TOPICS** respectively. For complete instructions to deal with migrating applications from JBoss EAP 6.4 to 7 Beta, see the [JBoss EAP 7 Beta Migration Guide](#).

6.2.4. Compatibility with xPaaS JBoss EAP

See the xPaaS section of the [OpenShift and Atomic Platform Tested Integrations page](#) for details about OpenShift EAP image version compatibility.

6.2.5. Setting Up the xPaaS JBoss EAP Image

The following is a list of prerequisites for using the xPaaS JBoss EAP images:

1. **Acquire Red Hat Subscriptions** - Ensure that you have the relevant subscriptions for OpenShift as well as a subscription for xPaaS Middleware.
2. **Install OpenShift** - Before using the xPaaS JBoss EAP images, you must have an OpenShift environment installed and configured:
 - a. The [Quick Installation](#) method allows you to install OpenShift using an interactive CLI utility.
 - b. The [Advanced Installation](#) method allows you to install OpenShift using a reference configuration. This method is best suited for production environments.
3. **Install and Deploy Docker Registry** - Install the Docker Registry and then ensure that the Docker Registry is deployed to locally manage images as follows:

```
$ oadm registry --config=/etc/origin/master/admin.kubeconfig --
credentials=/etc/origin/master/openshift-registry.kubeconfig
```

For further information, see [Deploying a Docker Registry](#)

4. **Deploy a Router** - Use the instructions at the [Deploying a Router](#) page for this step.
5. **Privileges** - Ensure that you can run the **oc create** command with [cluster-admin](#) privileges.
6. **Create Image Streams** - Image streams are configured during the Quick or Advanced OpenShift Installation. If required, manually create the image streams for both versions of the xPaaS JBoss EAP image as follows:

```
$ oc create -f /usr/share/ansible/openshift-
ansible/roles/openshift_examples/files/examples/v1.1/xpaas-
streams/jboss-image-streams.json -n openshift
```

**NOTE**

For further information about creating image streams, see [Loading the Default Image Streams and Templates](#)

7. **Create Instant App Templates** - Instant App templates define a full set of objects for running applications and are configured during the Quick or Advanced OpenShift Installation. If required, create Instant App templates as follows:

- a. Create the core Instant App templates:

```
$ oc create -f \ openshift-
ansible/roles/openshift_examples/files/examples/quickstart-
templates -n openshift
```

- b. Register Instant App templates for xPaaS Middleware products:

```
$ oc create -f \ openshift-
ansible/roles/openshift_examples/files/examples/xPaaS-templates -
n openshift
```

6.2.6. Modifying the JDK Used by the xPaaS JBoss EAP Image

The xPaaS JBoss EAP 6.4 image includes OpenJDK 1.7 and 1.8, with OpenJDK 1.8 as the default. The xPaaS JBoss EAP 7 Beta image only includes and supports OpenJDK 1.8.

To change the JDK version used by the xPaaS JBoss EAP 6.4 image:

1. Ensure that the ***pom.xml*** file specifies that the code must be built using the intended JDK version.
2. In the S2I application template, configure the image's **JAVA_HOME** environment variable to point to the intended JDK version. For example:

Example 6.1. Setting the JDK version

Change the defined value to point to the required version of the JDK.

```
name: "JAVA_HOME"
value: "/usr/lib/jvm/java-1.7.0"
```

6.2.7. Getting Started Using xPaaS JBoss EAP Images

6.2.7.1. Configuring the xPaaS JBoss EAP Images

You can change the configuration for the xPaaS JBoss EAP images by either using the S2I (Source to Image) templates, or by using a modified xPaaS JBoss EAP image. Red Hat recommends using the S2I method to configure the xPaaS JBoss EAP image.

6.2.7.2. Configuring the xPaaS JBoss EAP Image using the S2I Templates

The recommended method to run and configure the xPaaS JBoss EAP image is to use the OpenShift S2I process together with the application template parameters and environment variables.



NOTE

The variable **EAP_HOME** is used to denote the path to the JBoss EAP installation. Replace this variable with the actual path to your JBoss EAP installation.

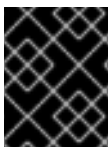
The S2I process for the xPaaS JBoss EAP image works as follows:

1. If a **pom.xml** file is present in the source repository, a Maven build using the contents of the **\$MAVEN_ARGS** environment variable is triggered. 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.redhatga**). The results of a successful Maven build are copied to **EAP_HOME/standalone/deployments**. 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 target directory.
2. Any JAR, WAR, and EAR in the deployment's source repository directory are copied to the **EAP_HOME/standalone/deployments** directory.
3. All files in the configuration source repository directory are copied to **EAP_HOME/standalone/configuration**. 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 **EAP_HOME/modules**.

6.2.7.3. Using a Modified xPaaS JBoss EAP Image

You can make changes to an image or create a custom image to use in OpenShift.

The JBoss EAP configuration file used by OpenShift in the xPaaS JBoss EAP image is **EAP_HOME/standalone/configuration/standalone-openshift.xml**. The script to start JBoss EAP is **EAP_HOME/bin/openshift-launch.sh**.



IMPORTANT

Ensure that you have read the [guidelines for creating images](#) and follow them when creating a modified image.

To use a modified image in OpenShift:

**WARNING**

This procedure results in losing configuration placeholders for various settings such as datasources, messaging, HTTPS, KeyCloak, etc. A workaround for this issue is to create a duplicate copy of the ***standalone.xml*** file to edit. The original and edited versions can be compared after all edits are complete and placeholder values can be copied to the edited version from the original version to retain these values.

1. Run the xPaaS JBoss EAP image using Docker.
2. Make the required changes using the JBoss EAP Management CLI by running the script at ***EAP_HOME/bin/jboss-cli.sh***.
3. Commit the changed container as a new image and then use the modified image in OpenShift.

6.2.7.4. Troubleshooting

If an application is not starting, use the following command to view details to locate and troubleshoot the problem:

```
$ oc describe po <pod_name>
```

To troubleshoot running xPaaS JBoss EAP containers, you can either view the OpenShift logs, or view the JBoss EAP logs displayed to the container's console. Use the following command to view the JBoss EAP logs:

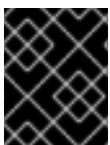
```
$ oc logs -f <pod_name> <container_name>
```

**NOTE**

By default, the xPaaS JBoss EAP image does not have a file log handler configured. Logs are therefore only sent to the console.

6.3. RED HAT JBOSS A-MQ XPAAS IMAGE**6.3.1. Overview**

Red Hat JBoss A-MQ (JBoss A-MQ) is available as a containerized xPaaS image that is designed for use with OpenShift. It allows developers to quickly deploy an A-MQ message broker in a hybrid cloud environment.

**IMPORTANT**

There are significant differences in supported configurations and functionality in the JBoss A-MQ image compared to the regular release of JBoss A-MQ.

This topic details the differences between the JBoss A-MQ xPaaS image and the regular release of JBoss A-MQ, and provides instructions specific to running and configuring the JBoss A-MQ xPaaS image. Documentation for other JBoss A-MQ functionality not specific to the JBoss A-MQ xPaaS image can be found in the [JBoss A-MQ documentation on the Red Hat Customer Portal](#).

6.3.2. Differences Between the JBoss A-MQ xPaaS Image and the Regular Release of JBoss A-MQ

There are several major functionality differences in the OpenShift JBoss A-MQ xPaaS image:

- The Karaf shell is not available.
- The Fuse Management Console (Hawtio) is not available.
- Configuration of the broker can be performed:
 - using parameters specified in the A-MQ application template, as described in [Application Template Parameters](#).
 - using the S2I (Source-to-image) tool, as described in [Configuration Using S2I](#).

6.3.3. Using the JBoss A-MQ xPaaS Image Streams and Application Templates

The Red Hat xPaaS middleware images were [automatically created during the installation](#) of OpenShift along with the other default image streams and templates.

6.3.4. Configuring the JBoss A-MQ Image

6.3.4.1. Application Template Parameters

Basic configuration of the JBoss A-MQ xPaaS image is performed by specifying values of application template parameters. The following parameters can be configured:

AMQ_RELEASE

The JBoss A-MQ release version. This determines which JBoss A-MQ image will be used as a basis for the application. At the moment, only version 6.2 is available.

APPLICATION_NAME

The name of the application used internally in OpenShift. It is used in names of services, pods, and other objects within the application.

MQ_USERNAME

The user name used for authentication to the broker. In a standard non-containerized JBoss A-MQ, you would specify the user name in the **AMQ_HOME/opt/user.properties** file. If no value is specified, a random user name is generated.

MQ_PASSWORD

The password used for authentication to the broker. In a standard non-containerized JBoss A-MQ, you would specify the password in the **AMQ_HOME/opt/user.properties** file. If no value is specified, a random password is generated.

AMQ_ADMIN_USERNAME

The user name used as an admin authentication to the broker. If no value is specified, a random user name is generated.

AMQ_ADMIN_PASSWORD

The password used for authentication to the broker. If no value is specified, a random password is generated.

MQ_PROTOCOL

Comma-separated list of the messaging protocols used by the broker. Available options are *amqp*, *mqtt*, *openwire*, and *stomp*. If left empty, all available protocols will be available. Please note that for integration of the image with Red Hat JBoss Enterprise Application Platform, the *openwire* protocol must be specified, while other protocols can be optionally specified as well.

MQ_QUEUES

Comma-separated list of queues available by default on the broker on its startup.

MQ_TOPICS

Comma-separated list of topics available by default on the broker on its startup.

AMQ_SECRET

The name of a secret containing SSL related files. If no value is specified, a random password is generated.

AMQ_TRUSTSTORE

The SSL trust store filename. If no value is specified, a random password is generated.

AMQ_KEYSTORE

The SSL key store filename. If no value is specified, a random password is generated.

6.3.4.2. Configuration Using S2I

Configuration of the JBoss A-MQ image can also be modified using the Source-to-image feature, described in full detail at [S2I Requirements](#).

Custom A-MQ broker configuration can be specified by creating an ***openshift-activemq.xml*** file inside the ***git*** directory of your application's Git project root. On each commit, the file will be copied to the ***conf*** directory in the A-MQ root and its contents used to configure the broker.

6.3.5. Configuring the JBoss A-MQ Persistent Image

6.3.5.1. Application Template Parameters

Basic configuration of the JBoss A-MQ Persistent xPaaS image is performed by specifying values of application template parameters. The following parameters can be configured:

AMQ_RELEASE

The JBoss A-MQ release version. This determines which JBoss A-MQ image will be used as a basis for the application. At the moment, only version 6.2 is available.

APPLICATION_NAME

The name of the application used internally in OpenShift. It is used in names of services, pods, and other objects within the application.

MQ_PROTOCOL

Comma-separated list of the messaging protocols used by the broker. Available options are *amqp*, *mqtt*, *openwire*, and *stomp*. If left empty, all available protocols will be available. Please note that for integration of the image with Red Hat JBoss Enterprise

Application Platform, the *openwire* protocol must be specified, while other protocols can be optionally specified as well.

MQ_QUEUES

Comma-separated list of queues available by default on the broker on its startup.

MQ_TOPICS

Comma-separated list of topics available by default on the broker on its startup.

VOLUME_CAPACITY

The size of the persistent storage for database volumes.

MQ_USERNAME

The user name used for authentication to the broker. In a standard non-containerized JBoss A-MQ, you would specify the user name in the **AMQ_HOME/opt/user.properties** file. If no value is specified, a random user name is generated.

MQ_PASSWORD

The password used for authentication to the broker. In a standard non-containerized JBoss A-MQ, you would specify the password in the **AMQ_HOME/opt/user.properties** file. If no value is specified, a random password is generated.

AMQ_ADMIN_USERNAME

The user name used as an admin authentication to the broker. If no value is specified, a random user name is generated.

AMQ_ADMIN_PASSWORD

The password used for authentication to the broker. If no value is specified, a random password is generated.

AMQ_SECRET

The name of a secret containing SSL related files. If no value is specified, a random password is generated.

AMQ_TRUSTSTORE

The SSL trust store filename. If no value is specified, a random password is generated.

AMQ_KEYSTORE

The SSL key store filename. If no value is specified, a random password is generated.

For more information, see [Using Persistent Volumes](#).

6.3.6. Security

Only SSL connections can connect from outside of the OpenShift instance, regardless of the protocol specified in the **MQ_PROTOCOL** property of the A-MQ application templates. The non-SSL version of the protocols can only be used inside the OpenShift instance.

For security reasons, using the default KeyStore and TrustStore generated by the system is discouraged. It is recommended to generate your own KeyStore and TrustStore and supply them to the image using the OpenShift secrets mechanism or S2I.

6.3.7. High-Availability and Scalability

The JBoss xPaaS A-MQ image is supported in two modes:

1. A single A-MQ pod mapped to a Persistent Volume for message persistence. This mode provides message High Availability and guaranteed messaging but does not provide scalability.

2. Multiple A-MQ pods using local message persistence (i.e. no mapped Persistent Volume). This mode provides scalability but does not provide message High Availability or guaranteed messaging.

6.3.8. Logging

In addition to viewing the OpenShift logs, you can troubleshoot a running JBoss A-MQ image by viewing the JBoss A-MQ logs that are outputted to the container's console:

```
$ oc logs -f <pod_name> <container_name>
```



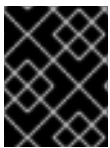
NOTE

By default, the OpenShift JBoss A-MQ xPaaS image does not have a file log handler configured. Logs are only sent to the console.

6.4. RED HAT JBOSS WEB SERVER XPAAS IMAGES

6.4.1. Overview

The Apache Tomcat 7 and Apache Tomcat 8 components of Red Hat JBoss Web Server 3 are available as containerized xPaaS images that are designed for use with OpenShift. Developers can use these images to quickly build, scale, and test Java web applications deployed across hybrid environments.



IMPORTANT

There are significant differences in the functionality between the JBoss Web Server xPaaS images and the regular release of JBoss Web Server.

This topic details the differences between the JBoss Web Server xPaaS images and the regular release of JBoss Web Server, and provides instructions specific to running and configuring the JBoss Web Server xPaaS images. Documentation for other JBoss Web Server functionality not specific to the JBoss Web Server xPaaS images can be found in the [JBoss Web Server documentation on the Red Hat Customer Portal](#).

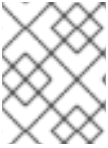
The location of ***JWS_HOME/tomcat<version>/*** inside a JBoss Web Server xPaaS image is: ***/opt/webserver/***.

6.4.2. Functionality Differences in the OpenShift JBoss Web Server xPaaS Images

A major functionality difference compared to the regular release of JBoss Web Server is that there is no Apache HTTP Server in the OpenShift JBoss Web Server xPaaS images. All load balancing in OpenShift is handled by the OpenShift router, so there is no need for a load-balancing Apache HTTP Server with `mod_cluster` or `mod_jk` connectors.

6.4.3. Using the JBoss Web Server xPaaS Image Streams and Application Templates

The Red Hat xPaaS middleware images were [automatically created during the installation](#) of OpenShift along with the other default image streams and templates.

**NOTE**

The JBoss Web Server xPaaS application templates are distributed as two sets: one set for Tomcat 7, and another for Tomcat 8. endif::[]

6.4.4. Using the JBoss Web Server xPaaS Image Source-to-Image (S2I) Process

To run and configure the OpenShift JBoss Web Server xPaaS images, use the OpenShift S2I process with the application template parameters and environment variables.

The S2I process for the JBoss Web Server xPaaS images works as follows:

1. If there is a ***pom.xml*** file in the source repository, a Maven build is triggered with the contents of ***\$MAVEN_ARGS*** environment variable.
By default the ***package*** goal is used with the ***openshift*** profile, including the system properties for skipping tests (***-DskipTests***) and enabling the Red Hat GA repository (***-Dcom.redhat.xpaas.repo.redhatga***).

The results of a successful Maven build are copied to ***/opt/webserver/webapps***. This includes all WAR files from the source repository directory specified by the ***\$ARTIFACT_DIR*** environment variable. The default value of ***\$ARTIFACT_DIR*** is the ***target*** directory.

2. All WAR files from the ***deployments*** source repository directory are copied to ***/opt/webserver/webapps***.
3. All files in the ***configuration*** source repository directory are copied to ***/opt/webserver/conf***.

**NOTE**

If you want to use custom Tomcat configuration files, the file names should be the same as for a normal Tomcat installation. For example, ***context.xml*** and ***server.xml***.

6.4.5. Troubleshooting

In addition to viewing the OpenShift logs, you can troubleshoot a running JBoss Web Server container by viewing the logs that are outputted to the container's console:

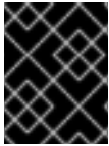
```
$ oc logs -f <pod_name> <container_name>
```

Additionally, access logs are written to ***/opt/webserver/logs/***.

6.5. RED HAT JBOSS FUSE INTEGRATION SERVICES

6.5.1. Overview

Red Hat JBoss Fuse Integration Services provides a set of tools and containerized xPaaS images that enable development, deployment, and management of integration microservices within OpenShift.

**IMPORTANT**

There are significant differences in supported configurations and functionality in Fuse Integration Services compared to the standalone JBoss Fuse product.

6.5.1.1. Differences Between Fuse Integration Services and JBoss Fuse

There are several major functionality differences:

- Fuse Management Console is not included as Fuse administration views have been integrated directly within the OpenShift Web Console.
- An application deployment with Fuse Integration Services consists of an application and all required runtime components packaged inside a Docker image. Applications are not deployed to a runtime as with Fuse, the application image itself is a complete runtime environment deployed and managed through OpenShift.
- Patching in an OpenShift environment is different from standalone Fuse since each application image is a complete runtime environment. To apply a patch, the application image is rebuilt and redeployed within OpenShift. Core OpenShift management capabilities allow for rolling upgrades and side-by-side deployment to maintain availability of your application during upgrade.
- Provisioning and clustering capabilities provided by Fabric in Fuse have been replaced with equivalent functionality in Kubernetes and OpenShift. There is no need to create or configure individual child containers as OpenShift automatically does this for you as part of deploying and scaling your application.
- Messaging services are created and managed using the A-MQ xPaaS images for OpenShift and not included directly within Fuse. Fuse Integration Services provides an enhanced version of the camel-amq component to allow for seamless connectivity to messaging services in OpenShift through Kubernetes.
- Live updates to running Karaf instances using the Karaf shell is strongly discouraged as updates will not be preserved if an application container is restarted or scaled up. This is a fundamental tenet of immutable architecture and essential to achieving scalability and flexibility within OpenShift.

Additional details on technical differences and support scope are documented in an associated [KCS article](#).

6.5.2. Using Fuse Integration Services

You can start using Fuse Integration Services by creating an application and deploying it to OpenShift using one of the following application development workflows:

- Fabric8 Maven Workflow
- OpenShift Source-to-Image (S2I) Workflow

Both workflows begin with creating a new project from a Maven archetype.

6.5.2.1. Maven Archetypes Catalog

The Maven Archetype catalog includes the following examples:

cdi-camel-http-archetype	Creates a new Camel route using CDI in a standalone Java Container calling the remote camel-servlet quickstart
cdi-cxf-archetype	Creates a new CXF JAX-RS using CDI running in a standalone Java Container
cdi-camel-archetype	Creates a new Camel route using CDI in a standalone Java Container
cdi-camel-jetty-archetype	Creates a new Camel route using CDI in a standalone Java Container using Jetty as HTTP server
java-simple-mainclass-archetype	Creates a new Simple standalone Java Container (main class)
java-camel-spring-archetype	Creates a new Camel route using Spring XML in a standalone Java container
karaf-cxf-rest-archetype	Creates a new RESTful WebService Example using JAX-RS
karaf-camel-rest-sql-archetype	Creates a new Camel Example using Rest DSL with SQL Database
karaf-camel-log-archetype	Creates a new Camel Log Example

Begin by selecting the archetype which matches the type of application you would like to create.

6.5.2.2. Create an Application from the Maven Archetype Catalog

You must configure the Maven repositories, which hold the archetypes and artifacts you may need, before creating a sample project:

- JBoss Fuse repository: <https://repo.fusesource.com/nexus/content/groups/public/>
- RedHat GA repository: <https://maven.repository.redhat.com/ga>

Use the maven archetype catalog to create a sample project with the required resources. The command to create a sample project is:

```
$ mvn archetype:generate \
-
DarchetypeCatalog=https://repo.fusesource.com/nexus/content/groups/public/
archetype-catalog.xml \
-DarchetypeGroupId=io.fabric8.archetypes \
-DarchetypeVersion=2.2.0.redhat-079 \
-DarchetypeArtifactId=<archetype-name>
```

**NOTE**

Replace `<archetype-name>` with the name of the archetype that you want to use. For example, **karaf-camel-log-archetype** creates a new Camel log example.

This will create a maven project with all required dependencies. Maven properties and plugins that are used to create Docker images are added to the ***pom.xml*** file.

6.5.2.3. Fabric8 Maven Workflow

Creates a new project based off a Maven application template created through Archetype catalog. This catalog provides examples of Java and Karaf projects and supports S2I and Maven deployment workflows.

1. Set the following environment variables to communicate with OpenShift and a Docker daemon:

DOCKER_HOST	Specifies the connection to a Docker daemon used to build an application Docker image	tcp://10.1.2.2:2375
KUBERNETES_MASTER	Specifies the URL for contacting the OpenShift API server	https://10.1.2.2:8443
KUBERNETES_DOMAIN	Domain used for creating routes. Your OpenShift API server must be mapped to all hosts of this domain.	openshift.dev

2. Login to OpenShift using CLI and select the project to which to deploy.

```
$ oc login
$ oc project <projectname>
```

3. Create a sample project as described in [Create an Application from the Maven Archetype Catalog](#).
4. Build and push the project to OpenShift. You can use following maven goals for building and pushing docker images.

docker:build	Builds the docker image for your maven project.
docker:push	Pushes the locally built docker image to the global or a local docker registry. This step is optional when developing on a single node OpenShift cluster.

fabric8:json	Generates kubernetes json file for your maven project. This goal is bound to the package phase and doesn't need to be called explicitly when running mvn install
fabric8:apply	Applies the kubernetes json file to the current Kubernetes environment and namespace.

There are few pre-configured maven profiles that you can use to build the project. These profiles are combinations of above maven goals that simplify the build process.

mvn -Pf8-build	Comprises of clean , install , docker:build , and fabric8:json . This will build dockerfile and JSON template for a project.
mvn -Pf8-local-deploy	Comprises of clean , install , docker:build , fabric8:json , and fabric8:apply . This will create docker and JSON templates and then apply them to OpenShift.
mvn -Pf8-deploy:	Comprises of clean , docker:build , fabric8:json , docker:push , and fabric8:apply . This will create docker and JSON templates, push them to docker registry and apply to OpenShift.

In this example, we will build it locally by running the command:

```
$ mvn -Pf8-local-deploy
```

5. Login to OpenShift Web Console. A pod is created for the newly created application. You can view the status of this pod, deployments and services that the application is creating.

6.5.2.3.1. Authenticating Against a Registry

For multi node OpenShift setups, the image created must be pushed to the OpenShift registry. This registry must be reachable from the outside through a route. Authentication against this registry reuses the OpenShift authentication with **oc login**. Assuming that your OpenShift registry is exposed as **registry.openshift.dev:80**, the project image can be deployed to the registry with following command:

```
$ mvn docker:push -Ddocker.registry=registry.openshift.dev:80 \
-Ddocker.username=$(oc whoami) \
```

```
-Ddocker.password=$(oc whoami -t)
```

To push changes to the registry, the OpenShift project must exist and the users of Docker image must be connected to the OpenShift project. All the examples uses the property **fabric8.dockerUser** as Docker image user which has **fabric8/** as default (note the trailing slash). When this user is used unaltered an OpenShift project 'fabric8' must exist. This can be created with 'oc new-project fabric8'.

6.5.2.3.2. Plug-in Configuration

Plug-ins **docker-maven-plugin** and **fabric8-maven-plugin** are responsible for creating Docker images and OpenShift API objects which can be configured flexibly. The examples from the archetypes introduces some extra properties which can be changed when running Maven:

docker.registry	Registry to use for docker:push and -Pf8-deploy
docker.username	Username for authentication against the registry
docker.password	Password for authentication against the registry
docker.from	Base image for the application Docker image
fabric8.dockerUser	User used in the image's name as user part. It must contain a / as trailing part. The default value is fabric8/ .
docker.image	The final Docker image name. Default value is \${fabric8.dockerUser}\${project.artifactId}:\${project.version}

6.5.2.4. OpenShift Source-to-Image (S2I) Workflow

Applications are created through OpenShift Admin Console and CLI using application templates. If you have a JSON or YAML file that defines a template, you can upload the template to the project using the CLI. This saves the template to the project for repeated use by users with appropriate access to that project. You can add the remote Git repository location to the template using template parameters. This allows you to pull the application source from remote repository and built using source-to-image (S2I) method.

JBoss Fuse Integration Services application templates depend on S2I builder **ImageStreams**, which **MUST** be created **ONCE**. The OpenShift installer creates them automatically. For users existing OpenShift setups, it can be achieved with the following command:

```
$ oc create -n openshift -f /usr/share/openshift/examples/xpaas-streams/fis-image-streams.json
```

The **ImageStreams** may be created in a namespace other than **openshift** by changing it in the command and corresponding template parameter **IMAGE_STREAM_NAMESPACE** when creating applications.

6.5.2.4.1. Create an Application Using Templates

1. Create an application template using command **mvn archetype:generate**. To create an application, upload the template to your current project's template library with the following command:

```
$ oc create -f quickstart-template.json -n <project>
```

The template is now available for selection using the web console or the CLI.

2. Login to OpenShift Web Console. In the desired project, click **Add to Project** to create the objects from an uploaded template.
3. Select the template from the list of templates in your project or from the global template library.
4. Edit template parameters and then click **Create**. For example, template parameters for a camel-spring quickstart are:

Parameter	Description	Default
APP_NAME	Application Name	Artifact name of the project
GIT_REPO	Git repository, required	
GIT_REF	Git ref to build	master
SERVICE_NAME	Exposed Service name	
BUILDER_VERSION	Builder version	1.0
APP_VERSION	Application version	Maven project version
MAVEN_ARGS	Arguments passed to mvn in the build	package -DskipTests -e
MAVEN_ARGS_APPEND	Extra arguments passed to mvn, e.g. for multi-module builds use -pl groupId:module-artifactId -am	
ARTIFACT_DIR	Maven build directory	target/
IMAGE_STREAM_NAMESPACE	Namespace in which the JBoss Fuse ImageStreams are installed.	

Parameter	Description	Default
BUILD_SECRET	generated if empty. The secret needed to trigger a build.	

- After successful creation of the application, you can view the status of application by clicking **Pods** tab or by running the following command:

```
$ oc get pods
```

For more information, see [Application Templates](#).

6.5.2.5. Developing Applications

6.5.2.5.1. Injecting Kubernetes Services into Applications

You can inject Kubernetes services into applications by labeling the pods and use those labels to select the required pods to provide a logical service. These labels are simple key, value pairs.

6.5.2.5.1.1. CDI Injection

Fabric8 provides a CDI extension that you can use to inject Kubernetes resources into your applications. To use the CDI extension, first add the dependency to the project's **pom.xml** file.

```
<dependency>
  <groupId>io.fabric8</groupId>
  <artifactId>fabric8-cdi</artifactId>
  <version>{$fabric8.version}</version>
</dependency>
```

Next step is to identify the field that requires the service and then inject the service by adding a **@ServiceName** annotation to it. For example,

```
@Inject
@ServiceName("my-service")
private String service.
```

The **@PortName** annotation is used to select a specific port by name when multiple ports are defined for a service.

6.5.2.5.1.2. Using Environment Variables as Properties

You can use to access a service by using environment variables to expose the fixed IP address and port. These are, **SERVICE_HOST** and **SERVICE_PORT**. **SERVICE_HOST** is the host (IP) address of the service and **SERVICE_PORT** is the port of the service.

6.6. DECISION SERVER XPAAS IMAGE

6.6.1. Overview

Decision Server is available as a containerized xPaaS image that is designed for use with OpenShift as an execution environment for business rules. Developers can quickly build, scale, and test applications deployed across hybrid environments.



IMPORTANT

There are significant differences in supported configurations and functionality in the Decision Server xPaaS image compared to the regular release of JBoss BRMS.

This topic details the differences between the Decision Server xPaaS image and the full, non-PaaS release of JBoss BRMS, and provides instructions specific to running and configuring the Decision Server xPaaS image. Documentation for other JBoss BRMS functionality not specific to the Decision Server xPaaS image can be found in the [JBoss BRMS documentation on the Red Hat Customer Portal](#).

EAP_HOME in this documentation, as in the [JBoss BRMS documentation](#), is used to refer to the JBoss EAP installation directory where the decision server is deployed. The location of **EAP_HOME** inside a Decision Server xPaaS image is **/opt/eap/**, which the **JBOSS_HOME** environment variable is also set to by default.

6.6.2. Comparing the Decision Server xPaaS Image to the Regular Release of JBoss BRMS

6.6.2.1. Functionality Differences for OpenShift Decision Server xPaaS Images

There are several major functionality differences in the OpenShift Decision Server xPaaS image:

- The Decision Server image extends the OpenShift EAP image, and any capabilities or limitations it has are also found in the Decision Server image.
- Only stateless scenarios are supported.
- Authoring of any content through the BRMS Console or API is not supported.

6.6.2.2. Managing OpenShift Decision Server xPaaS Images

As the Decision Server image is built off the OpenShift JBoss EAP xPaaS image, the JBoss EAP Management CLI is accessible from within the container for troubleshooting purposes.

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

```
$ oc rsh <pod_name>
```

2. Then run the following from the remote shell session to launch the JBoss EAP Management CLI:

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

**WARNING**

Any configuration changes made using the JBoss EAP Management CLI on a running container will be lost when the container restarts.

Making configuration changes to the JBoss EAP instance inside the JBoss EAP xPaaS image is different from the process you may be used to for a regular release of JBoss EAP.

6.6.2.3. Security in the OpenShift Decision Server xPaaS Image

Access is limited to users with the **kie-server** authorization role. A user with this role can be specified via the **KIE_SERVER_USER** and **KIE_SERVER_PASSWORD** environment variables.

**NOTE**

The HTTP/REST endpoint is configured to only allow the execution of KIE containers and querying of KIE Server resources. Administrative functions like creating or disposing Containers, updating Releases or Scanners, etc. are restricted. The JMS endpoint currently does not support these restrictions. In the future, more fine-grained security configuration should be available for both endpoints.

6.6.3. Using the Decision Server xPaaS Image Streams and Application Templates

The Red Hat xPaaS middleware images were [automatically created during the installation](#) of OpenShift along with the other default image streams and templates.

6.6.4. Running and Configuring the Decision Server xPaaS Image

You can make changes to the Decision Server configuration in the xPaaS image using either the S2I templates, or by using a modified Decision Server image.

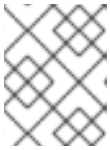
6.6.4.1. Using the Decision Server xPaaS Image Source-to-Image (S2I) Process

The recommended method to run and configure the OpenShift Decision Server xPaaS image is to use the OpenShift S2I process together with the application template parameters and environment variables.

The S2I process for the Decision Server xPaaS image works as follows:

1. If there is a **pom.xml** file in the source repository, a Maven build is triggered with the contents of **\$MAVEN_ARGS** environment variable.
 - By default, the **package** goal is used with the **openshift** profile, including the system properties for skipping tests (**-DskipTests**) and enabling the Red Hat GA repository (**-Dcom.redhat.xpaas.repo.redhatga**).

2. The results of a successful Maven build are installed into the local Maven repository, **/home/jboss/.m2/repository/**, along with all dependencies for offline usage. The Decision Server xPaaS Image will load the created k jars from this local repository.
 - In addition to k jars resulting from the Maven build, any k jars found in the deployments source directory will also be installed into the local Maven repository. K jars do not end up in the **EAP_HOME/standalone/deployments/** directory.
3. Any JAR (that is not a k jar) , WAR, and EAR in the **deployments** source repository directory will be copied to the **EAP_HOME/standalone/deployments** directory and subsequently deployed using the JBoss EAP deployment scanner.
4. All files in the **configuration** source repository directory are copied to **EAP_HOME/standalone/configuration**.



NOTE

If you want to use a custom JBoss EAP configuration file, it should be named **standalone-openshift.xml**.

5. All files in the **modules** source repository directory are copied to **EAP_HOME/modules**.

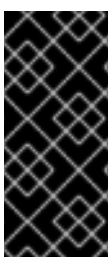
6.6.4.2. Using a Modified Decision Server xPaaS Image

An alternative method is to make changes to the image, and then use that modified image in OpenShift. The templates currently provided, along with the interfaces they support, are listed below:

Table 6.1. Provided Templates

Template Name	Supported Interfaces
decisionserver62-basic-s2i.json	http-rest, jms-hornetq
decisionserver62-https-s2i.json	http-rest, https-rest, jms-hornetq
decisionserver62-amq-s2i.json	http-rest, https-rest, jms-activemq

You can run the Decision Server xPaaS image in Docker, make the required configuration changes using the JBoss EAP Management CLI (**EAP_HOME/bin/jboss-cli.sh**) included in the Decision Server xPaaS image, and then commit the changed container as a new image. You can then use that modified image in OpenShift.



IMPORTANT

It is recommended that you do not replace the OpenShift placeholders in the JBoss EAP xPaaS configuration file, as they are used to automatically configure services (such as messaging, datastores, HTTPS) during a container's deployment. These configuration values are intended to be set using environment variables.

**NOTE**

Ensure that you follow the [guidelines for creating images](#).

6.6.4.3. Updating Rules

As each image is built from a snapshot of a specific Maven repository, whenever a new rule is added, or an existing rule modified, a new image must be created and deployed for the rule modifications to take effect.

6.6.5. Endpoints

Clients can access the Decision Server xPaaS Image via multiple endpoints; by default the provided templates include support for REST, HornetQ, and ActiveMQ.

6.6.5.1. REST

Clients can use the [REST API](#) in various ways:

6.6.5.1.1. Browser

1. Current server state: <http://host/kie-server/services/rest/server>
2. List of containers: <http://host/kie-server/services/rest/server/containers>
3. Specific container state: <http://host/kie-server/services/rest/server/containers/HelloRulesContainer>

6.6.5.1.2. Java

```
// HelloRulesClient.java
KieServicesConfiguration config = KieServicesFactory.newRestConfiguration(
    "http://host/kie-server/services/rest/server", "kieserverUser",
    "kieserverPassword");
config.setMarshallingFormat(MarshallingFormat.XSTREAM);
RuleServicesClient client =

KieServicesFactory.newKieServicesClient(config).getServicesClient(RuleServ
icesClient.class);
ServiceResponse<String> response =
client.executeCommands("HelloRulesContainer", myCommands);
```

6.6.5.1.3. Command Line

```
# request.sh
#!/bin/sh
curl -X POST \
  -d @request.xml \
  -H "Accept:application/xml" \
  -H "X-KIE-ContentType:XSTREAM" \
  -H "Content-Type:application/xml" \
  -H "Authorization:Basic a2llc2VydmVy0mtpZXNlcnZlcjEh" \
  -H "X-KIE-
```

```
ClassType:org.drools.core.command.runtime.BatchExecutionCommandImpl" \
http://host/kie-
server/services/rest/server/containers/instances/HelloRulesContainer
```

```
<!-- request.xml -->
<batch-execution lookup="HelloRulesSession">
  <insert>
    <org.openshift.quickstarts.decisionserver.hellorules.Person>
      <name>errantepiphany</name>
    </org.openshift.quickstarts.decisionserver.hellorules.Person>
  </insert>
  <fire-all-rules/>
  <query out-identifier="greetings" name="get greeting"/>
</batch-execution>
```

6.6.5.2. JMS

Client can also use the Java Messaging Service, as demonstrated below:

6.6.5.2.1. Java (HornetQ)

```
// HelloRulesClient.java
Properties props = new Properties();
props.setProperty(Context.INITIAL_CONTEXT_FACTORY,
    "org.jboss.naming.remote.client.InitialContextFactory");
props.setProperty(Context.PROVIDER_URL, "remote://host:4447");
props.setProperty(Context.SECURITY_PRINCIPAL, "kieserverUser");
props.setProperty(Context.SECURITY_CREDENTIALS, "kieserverPassword");
InitialContext context = new InitialContext(props);
KieServicesConfiguration config =
    KieServicesFactory.newJMSConfiguration(context, "hornetqUser",
    "hornetqPassword");
config.setMarshallingFormat(MarshallingFormat.XSTREAM);
RuleServicesClient client =

KieServicesFactory.newKieServicesClient(config).getServicesClient(RuleServ
icesClient.class);
ServiceResponse<String> response =
    client.executeCommands("HelloRulesContainer", myCommands);
```

6.6.5.2.2. Java (ActiveMQ)

```
// HelloRulesClient.java
props.setProperty(Context.INITIAL_CONTEXT_FACTORY,
    "org.apache.activemq.jndi.ActiveMQInitialContextFactory");
props.setProperty(Context.PROVIDER_URL, "tcp://host:61616");
props.setProperty(Context.SECURITY_PRINCIPAL, "kieserverUser");
props.setProperty(Context.SECURITY_CREDENTIALS, "kieserverPassword");
InitialContext context = new InitialContext(props);
ConnectionFactory connectionFactory =
    (ConnectionFactory)context.lookup("ConnectionFactory");
Queue requestQueue =
    (Queue)context.lookup("dynamicQueues/queue/KIE.SERVER.REQUEST");
Queue responseQueue =
```

```
(Queue)context.lookup("dynamicQueues/queue/KIE.SERVER.RESPONSE");
KieServicesConfiguration config = KieServicesFactory.newJMSConfiguration(
    connectionFactory, requestQueue, responseQueue, "activemqUser",
    "activemqPassword");
config.setMarshallingFormat(MarshallingFormat.XSTREAM);
RuleServicesClient client =

KieServicesFactory.newKieServicesClient(config).getServicesClient(RuleServ
icesClient.class);
ServiceResponse<String> response =
client.executeCommands("HelloRulesContainer", myCommands);
```

6.6.6. Troubleshooting

In addition to viewing the OpenShift logs, you can troubleshoot a running Decision Server xPaaS Image container by viewing its logs. These are outputted to the container's standard out, and are accessible with the following command:

```
$ oc logs -f <pod_name> <container_name>
```



NOTE

By default, the OpenShift Decision Server xPaaS image does not have a file log handler configured. Logs are only sent to the container's standard out.

6.7. RED HAT JBOSS DATA GRID XPAAS IMAGE

6.7.1. Overview

Red Hat JBoss Data Grid is available as a containerized xPaaS image that is designed for use with OpenShift. This image provides an in-memory distributed database so that developers can quickly access large amounts of data in a hybrid environment.



IMPORTANT

There are significant differences in supported configurations and functionality in the JBoss Data Grid xPaaS image compared to the full, non-PaaS release of JBoss Data Grid.

This topic details the differences between the JBoss Data Grid xPaaS image and the full, non-PaaS release of JBoss Data Grid, and provides instructions specific to running and configuring the JBoss Data Grid xPaaS image. Documentation for other JBoss Data Grid functionality not specific to the JBoss Data Grid xPaaS image can be found in the [JBoss Data Grid documentation on the Red Hat Customer Portal](#).



NOTE

OpenShift Container Platform release 3.1 supports JBoss Data Grid release 6.6 and earlier. If you are using OpenShift release 3.2 and JBoss Data Grid release 7.1 or later, refer to the documentation in the [Red Hat Customer Portal](#)

6.7.2. Comparing the JBoss Data Grid xPaaS Image to the Regular Release of JBoss Data Grid

6.7.2.1. Functionality Differences for OpenShift JBoss Data Grid xPaaS Images

There are several major functionality differences in the OpenShift JBoss Data Grid xPaaS image:

- The JBoss Data Grid Management Console is not available [to manage OpenShift JBoss Data Grid xPaaS images](#).
- The JBoss Data Grid Management CLI is only bound locally. This means that you can only access the Management CLI of a container from within the pod.
- Library mode is not supported.
- Only JDBC is supported for a backing cache-store. Support for remote cache stores are present only for data migration purposes.

6.7.2.2. Forming a Cluster using the OpenShift JBoss Data Grid xPaaS Images

Clustering is achieved through one of two discovery mechanisms: Kubernetes or DNS. This is accomplished by configuring the JGroups protocol stack in ***clustered-openshift.xml*** with either the ***<openshift.KUBE_PING/>*** or ***<openshift.DNS_PING/>*** elements. By default ***KUBE_PING*** is the pre-configured and supported protocol.

For ***KUBE_PING*** to work the following steps must be taken:

1. The ***OPENSHIFT_KUBE_PING_NAMESPACE*** environment variable must be set (as seen in the [Configuration Environment Variables](#)). If this variable is not set, then the server will act as if it is a single-node cluster, or a cluster that consists of only one node.
2. The ***OPENSHIFT_KUBE_PING_LABELS*** environment variable must be set (as seen in the [Configuration Environment Variables](#)). If this variable is not set, then pods outside the application (but in the same namespace) will attempt to join.
3. Authorization must be granted to the service account the pod is running under to be allowed to Kubernetes' REST api. This is done on the command line:

Example 6.2. Policy commands

Using the ***default*** service account in the ***myproject*** 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 ***myproject*** namespace:

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

Once the above is configured images will automatically join the cluster as they are deployed; however, removing images from an active cluster, and therefore shrinking the cluster, is not supported.

6.7.2.3. Endpoints

Clients can access JBoss Data Grid via REST, HotRod, and memcached endpoints defined as usual in the cache's configuration.

If a client attempts to access a cache via HotRod and is in the same project it will be able to receive the full cluster view and make use of consistent hashing; however, if it is in another project then the client will be unable to receive the cluster view. Additionally, if the client is located outside of the project that contains the HotRod cache there will be additional latency due to extra network hops being required to access the cache.



IMPORTANT

Only caches with an exposed REST endpoint will be accessible outside of OpenShift.

6.7.2.4. Configuring Caches

A list of caches may be defined by the **CACHE_NAMES** environment variable. By default the following caches are created:

- **default**
- **memcached**

Each cache's behavior may be controlled through the use of cache-specific environment variables, with each environment variable expecting the cache's name as the prefix. For instance, consider the **default** cache, any configuration applied to this cache must begin with the **DEFAULT_** prefix. To define the number of cache entry owners for each entry in this cache the **DEFAULT_CACHE_OWNERS** environment variable would be used.

A full list of these is found at [Cache Environment Variables](#).

6.7.2.5. Datasources

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

The most important variable is the **DB_SERVICE_PREFIX_MAPPING** which defines JNDI mappings for datasources. It must be set to a comma-separated list of **<name> <database_type>=<PREFIX> triplet**, where ***name** is used as the pool-name in the datasource, **database_type** determines which database driver to use, and **PREFIX** is the prefix used in the names of environment variables, which are used to configure the datasource.

6.7.2.5.1. JNDI Mappings for Datasources

For each **<name>-<database_type>=<PREFIX>** triplet in the **DB_SERVICE_PREFIX_MAPPING** environment variable, a separate datasource will be created by the launch script, which is executed when running the image.

The **<database_type>** will determine the driver for the datasource. Currently, only **postgresql** and **mysql** are supported.

The **<name>** parameter can be chosen on your own. Do not use any special characters.



NOTE

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

6.7.2.5.2. Database Drivers

The JBoss Data Grid xPaaS image contains Java drivers for MySQL, PostgreSQL, and MongoDB databases deployed. Datasources are **generated only for MySQL and PostgreSQL databases**.



NOTE

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

6.7.2.5.3. Examples

The following examples demonstrate how datasources may be defined using the **DB_SERVICE_PREFIX_MAPPING** environment variable.

6.7.2.5.3.1. Single Mapping

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

This will create a datasource named **java:jboss/datasources/test_postgresql**. Additionally, all of the required settings, such as username and password, will be expected to be provided as environment variables with the **TEST_** prefix, such as **TEST_USERNAME** and **TEST_PASSWORD**.

6.7.2.5.3.2. Multiple Mappings

Multiple database mappings may also be specified; for instance, considering the following value for the **DB_SERVICE_PREFIX_MAPPING** environment variable: **cloud-postgresql=CLOUD,test-mysql=TEST_MYSQL**.



NOTE

Multiple datasource mappings should be separated with commas, as seen in the above example.

This will create two datasources:

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

MySQL datasource configuration, such as the username and password, will be expected with the **TEST_MYSQL** prefix, for example **TEST_MYSQL_USERNAME**. Similarly the

PostgreSQL datasource will expect to have environment variables defined with the **CLOUD_** prefix, such as **CLOUD_USERNAME**.

6.7.2.5.4. Environment Variables

A full list of datasource environment variables may be found at [Datasource Environment Variables](#).

6.7.2.6. Security Domains

To configure a new Security Domain the **SECDOMAIN_NAME** environment variable must be defined, which will result in the creation of a security domain named after the passed in value. This domain may be configured through the use of the [Security Environment Variables](#).

6.7.2.7. Managing OpenShift JBoss Data Grid xPaaS Images

A major difference in managing an OpenShift JBoss Data Grid xPaaS image is that there is no Management Console exposed for the JBoss Data Grid installation inside the image. Because images are intended to be immutable, with modifications being written to a non-persistent file system, the Management Console is not exposed.

However, the JBoss Data Grid Management CLI (**JDG_HOME/bin/jboss-cli.sh**) is still accessible from within the container for troubleshooting purposes.

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

```
$ oc rsh <pod_name>
```

2. Then run the following from the remote shell session to launch the JBoss Data Grid Management CLI:

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



WARNING

Any configuration changes made using the JBoss Data Grid Management CLI on a running container will be lost when the container restarts.

[Making configuration changes to the JBoss Data Grid instance inside the JBoss Data Grid xPaaS image](#) is different from the process you may be used to for a regular release of JBoss Data Grid.

6.7.3. Using the JBoss Data Grid xPaaS Image Streams and Application Templates

The Red Hat xPaaS middleware images were [automatically created during the installation](#) of OpenShift along with the other default image streams and templates.

6.7.4. Running and Configuring the JBoss Data Grid xPaaS Image

You can make changes to the JBoss Data Grid configuration in the xPaaS image using either the S2I templates, or by using a modified JBoss Data Grid xPaaS image.

6.7.4.1. Using the JBoss Data Grid xPaaS Image Source-to-Image (S2I) Process

The recommended method to run and configure the OpenShift JBoss Data Grid xPaaS image is to use the OpenShift S2I process together with the application template parameters and environment variables.

The S2I process for the JBoss Data Grid xPaaS image works as follows:

1. If there is a ***pom.xml*** file in the source repository, a Maven build is triggered with the contents of ***\$MAVEN_ARGS*** environment variable.
2. By default the ***package*** goal is used with the ***openshift*** profile, including the system properties for skipping tests (***-DskipTests***) and enabling the Red Hat GA repository (***-Dcom.redhat.xpaas.repo.redhatga***).
3. The results of a successful Maven build are copied to ***JDG_HOME/standalone/deployments***. This includes all JAR, WAR, and EAR files from the directory within the source repository specified by ***\$ARTIFACT_DIR*** environment variable. The default value of ***\$ARTIFACT_DIR*** is the ***target*** directory.
 - Any JAR, WAR, and EAR in the ***deployments*** source repository directory are copied to the ***JDG_HOME/standalone/deployments*** directory.
 - All files in the ***configuration*** source repository directory are copied to ***JDG_HOME/standalone/configuration***.



NOTE

If you want to use a custom JBoss Data Grid configuration file, it should be named ***clustered-openshift.xml***.

4. All files in the ***modules*** source repository directory are copied to ***JDG_HOME/modules***.

6.7.4.1.1. Using a Different JDK Version in the JBoss Data Grid xPaaS Image

The JBoss Data Grid xPaaS image may come with multiple versions of OpenJDK installed, but only one is the default. For example, the JBoss Data Grid 6.5 xPaaS image comes with OpenJDK 1.7 and 1.8 installed, but OpenJDK 1.8 is the default.

If you want the JBoss Data Grid xPaaS image to use a different JDK version than the default, you must:

- Ensure that your ***pom.xml*** specifies to build your code using the intended JDK version.
- In the S2I application template, configure the image's ***JAVA_HOME*** environment variable to point to the intended JDK version. For example:

```
name: "JAVA_HOME"
value: "/usr/lib/jvm/java-1.7.0"
```

6.7.4.2. Using a Modified JBoss Data Grid xPaaS Image

An alternative method is to make changes to the image, and then use that modified image in OpenShift.

The JBoss Data Grid configuration file that OpenShift uses inside the JBoss Data Grid xPaaS image is ***JDG_HOME/standalone/configuration/clustered-openshift.xml***, and the JBoss Data Grid startup script is ***JDG_HOME/bin/openshift-launch.sh***.

You can run the JBoss Data Grid xPaaS image in Docker, make the required configuration changes using the JBoss Data Grid Management CLI (***JDG_HOME/bin/jboss-cli.sh***), and then commit the changed container as a new image. You can then use that modified image in OpenShift.



IMPORTANT

It is recommended that you do not replace the OpenShift placeholders in the JBoss Data Grid xPaaS configuration file, as they are used to automatically configure services (such as messaging, datastores, HTTPS) during a container's deployment. These configuration values are intended to be set using environment variables.



NOTE

Ensure that you follow the [guidelines for creating images](#)

6.7.5. Environment Variables

6.7.5.1. Information Environment Variables

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

Table 6.2. Information Environment Variables

Variable Name	Description	Value
<i>JBOSS_DATAGRID_VERSION</i>	The full, non-PaaS release that the xPaaS image is based from.	<i>6.5.1.GA</i>
<i>JBOSS_HOME</i>	The directory where the JBoss distribution is located.	<i>/opt/datagrid</i>
<i>JBOSS_IMAGE_NAME</i>	Image name, same as <i>Name</i> label	<i>jboss-datagrid-6/datagrid65-openshift</i>
<i>JBOSS_IMAGE_RELEASE</i>	Image release, same as <i>Release</i> label	Example: dev

Variable Name	Description	Value
<i>JBOSS_IMAGE_VERSION</i>	Image version, same as <i>Version</i> label	Example: 1.2
<i>JBOSS_MODULES_SYSTEM_PKGS</i>		<i>org.jboss.logmanager</i>
<i>JBOSS_PRODUCT</i>		<i>datagrid</i>
<i>LAUNCH_JBOSS_IN_BACKGROUND</i>	Allows the data grid server to be gracefully shutdown even when there is no terminal attached.	<i>true</i>

6.7.5.2. Configuration Environment Variables

Configuration environment variables are designed to conveniently adjust the image without requiring a rebuild, and should be set by the user as desired.

Table 6.3. Configuration Environment Variables

Variable Name	Description	Value
<i>CACHE_CONTAINER_START</i>	Should this cache container be started on server startup, or lazily when requested by a service or deployment. Defaults to <i>LAZY</i>	Example: <i>EAGER</i>
<i>CACHE_CONTAINER_STATISTICS</i>	Determines if the cache container collects statistics. Disable for optimal performance. Defaults to <i>true</i> .	Example: <i>false</i>
<i>CACHE_NAMES</i>	List of caches to configure. Defaults to <i>default, memcached</i> , and each defined cache will be configured as a distributed-cache with a mode of <i>SYNC</i> .	Example: <i>addressbook, addressbook_indexed</i>
<i>CONTAINER_SECURITY_CUSTOM_ROLE_MAPPER_CLASS</i>	Class of the custom principal to role mapper.	Example: <i>com.acme.CustomRoleMapper</i>

Variable Name	Description	Value
CONTAINER_SECURITY_IDENTITY_ROLE_MAPPER	Set a role mapper for this cache container. Valid values are: identity-role-mapper,common-name-role-mapper,cluster-role-mapper,custom-role-mapper .	Example: identity-role-mapper
CONTAINER_SECURITY_ROLES	Define role names and assign permissions to them.	Example: admin=ALL,reader=READ,writer=WRITE
DB_SERVICE_PREFIX_MAPPING	Define a comma-separated list of datasources to configure.	Example: test-mysql=TEST_MYSQL
DEFAULT_CACHE	Indicates the default cache for this cache container.	Example: addressbook
ENCRYPTION_REQUIRE_SSL_CLIENT_AUTH	Whether to require client certificate authentication. Defaults to false .	Example: true
HOTROD_AUTHENTICATION	If defined the hotrod-connectors will be configured with authentication in the ApplicationRealm .	Example: true
HOTROD_ENCRYPTION	If defined the hotrod-connectors will be configured with encryption in the ApplicationRealm .	Example: true
HOTROD_SERVICE_NAME	Name of the OpenShift service used to expose HotRod externally.	Example: DATAGRID_APP_HOTROD
INFINISPAN_CONNECTORS	Comma separated list of connectors to configure. Defaults to hotrod,memcached,rest . Note that if authorization or authentication is enabled on the cache then memcached should be removed as this protocol is inherently insecure.	Example: hotrod

Variable Name	Description	Value
JAVA_OPTS_APPEND	The contents of JAVA_OPTS_APPEND is appended to JAVA_OPTS on startup.	Example: -Dfoo=bar
JGROUPS_CLUSTER_PASSWORD	A password to control access to JGroups. Needs to be set consistently cluster-wide. The image default is to use the OPENSIFT_KUBE_PING_LABELS variable value; however, the JBoss application templates generate and supply a random value.	Example: miR0JaDR
MEMCACHED_CACHE	The name of the cache to use for the Memcached connector.	Example: memcached
OPENSIFT_KUBE_PING_LABELS	Clustering labels selector.	Example: application=eap-app
OPENSIFT_KUBE_PING_NAMESPACE	Clustering project namespace.	Example: myproject
PASSWORD	Password for the JDG user.	Example: p@ssw0rd
REST_SECURITY_DOMAIN	The security domain to use for authentication and authorization purposes. Defaults to none (no authentication).	Example: other
TRANSPORT_LOCK_TIMEOUT	Infinispan uses a distributed lock to maintain a coherent transaction log during state transfer or rehashing, which means that only one cache can be doing state transfer or rehashing at the same time. This constraint is in place because more than one cache could be involved in a transaction. This timeout controls the time to wait to acquire a distributed lock. Defaults to 240000 .	Example: 120000
USERNAME	Username for the JDG user.	Example: openshift

6.7.5.3. Cache Environment Variables

The following environment variables all control behavior of individual caches; when defining these values for a particular cache substitute the cache's name for **CACHE_NAME**.

Table 6.4. Cache Environment Variables

Variable Name	Description	Example Value
<CACHE_NAME>_CACHE_TYPE	Determines whether this cache should be distributed or replicated. Defaults to distributed .	replicated
<CACHE_NAME>_CACHE_START	Determines if this cache should be started on server startup, or lazily when requested by a service or deployment. Defaults to LAZY .	EAGER
<CACHE_NAME>_CACHE_BATCHING	Enables invocation batching for this cache. Defaults to false .	true
<CACHE_NAME>_CACHE_STATISTICS	Determines whether or not the cache collects statistics. Disable for optimal performance. Defaults to true .	false
<CACHE_NAME>_CACHE_MODE	Sets the clustered cache mode, ASYNC for asynchronous operations, or SYNC for synchronous operations.	ASYNC
<CACHE_NAME>_CACHE_QUEUE_SIZE	In ASYNC mode this attribute can be used to trigger flushing of the queue when it reaches a specific threshold. Defaults to 0 , which disables flushing.	100
<CACHE_NAME>_CACHE_QUEUE_FLUSH_INTERVAL	In ASYNC mode this attribute controls how often the asynchronous thread runs to flush the replication queue. This should be a positive integer that represents thread wakeup time in milliseconds. Defaults to 10 .	20

Variable Name	Description	Example Value
<CACHE_NAME>_CACHE_REMOTE_TIMEOUT	In SYNC mode the timeout, in milliseconds, used to wait for an acknowledgement when making a remote call, after which the call is aborted and an exception is thrown. Defaults to 17500 .	25000
<CACHE_NAME>_CACHE_OWNERS	Number of cluster-wide replicas for each cache entry. Defaults to 2 .	5
<CACHE_NAME>_CACHE_SEGMENTS	Number of hash space segments per cluster. The recommended value is 10 * cluster size. Defaults to 80 .	30
<CACHE_NAME>_CACHE_L1_LIFESPAN	Maximum lifespan, in milliseconds, of an entry placed in the L1 cache. Defaults to 0 , indicating that L1 is disabled.	100 .
<CACHE_NAME>_CACHE_EVICTION_STRATEGY	Sets the cache eviction strategy. Available options are UNORDERED , FIFO , LRU , LIRS , and NONE (to disable eviction). Defaults to NONE .	FIFO
<CACHE_NAME>_CACHE_EVICTION_MAX_ENTRIES	Maximum number of entries in a cache instance. If selected value is not a power of two the actual value will default to the least power of two larger than the selected value. A value of -1 indicates no limit. Defaults to 10000 .	-1
<CACHE_NAME>_CACHE_EXPIRATION_LIFESPAN	Maximum lifespan, in milliseconds, of a cache entry, after which the entry is expired cluster-wide. Defaults to -1 , indicating that the entries never expire.	10000

Variable Name	Description	Example Value
<CACHE_NAME>_CACHE_EXPIRATION_MAX_IDLE	Maximum idle time, in milliseconds, a cache entry will be maintained in the cache. If the idle time is exceeded, then the entry will be expired cluster-wide. Defaults to -1 , indicating that the entries never expire.	10000
<CACHE_NAME>_CACHE_EXPIRATION_INTERVAL	Interval, in milliseconds, between subsequent runs to purge expired entries from memory and any cache stores. If you wish to disable the periodic eviction process altogether, then set the interval to -1 . Defaults to 5000 .	-1
<CACHE_NAME>_CACHE_COMPATIBILITY_ENABLED	Enables compatibility mode for this cache. Disabled by default.	true
<CACHE_NAME>_CACHE_COMPATIBILITY_MARSHALLER	A marshaller to use for compatibility conversions.	com.acme.CustomMarshaller
<CACHE_NAME>_JDBC_STORE_TYPE	Type of JDBC store to configure. This value may either be string or binary .	string
<CACHE_NAME>_JDBC_STORE_DATASOURCE	Defines the jndiname of the datasource.	java:jboss/datasources/ExampleDS
<CACHE_NAME>_KEYED_TABLE_PREFIX	Defines the prefix prepended to the cache name used when composing the name of the cache entry table. Defaults to ispn_entry .	JDG
<CACHE_NAME>_CACHE_INDEX	The indexing mode of the cache. Valid values are NONE , LOCAL , and ALL . Defaults to NONE .	ALL
<CACHE_NAME>_CACHE_INDEXING_PROPERTIES	Comma separated list of properties to pass on to the indexing system.	default.directory_provider=ram

Variable Name	Description	Example Value
<CACHE_NAME>_CACHE_SECURITY_AUTHORIZATION_ENABLED	Enables authorization checks for this cache. Defaults to false .	true
<CACHE_NAME>_CACHE_SECURITY_AUTHORIZATION_ROLES	Sets the valid roles required to access this cache.	admin,reader,writer
<CACHE_NAME>_CACHE_PARTITION_HANDLING_ENABLED	If enabled, then the cache will enter degraded mode when it loses too many nodes. Defaults to true .	false

6.7.5.4. Datasource Environment Variables

Datasource properties may be configured with the following environment variables:

Table 6.5. Datasource Environment Variables

Variable Name	Description	Example Value
<NAME>_<DATABASE_TYPE>_SERVICE_HOST_	Defines the database server's hostname or IP to be used in the datasource's connection_url property.	192.168.1.3
<NAME>_DATABASE_TYPE_SERVICE_PORT	Defines the database server's port for the datasource.	5432
<PREFIX>_JNDI	Defines the JNDI name for the datasource. Defaults to java:jboss/datasources/<name><database_type>_ , where name and database_type are taken from the triplet definition. This setting is useful if you want to override the default generated JNDI name.	java:jboss/datasources/test-postgresql
<PREFIX>_USERNAME	Defines the username for the datasource.	admin
<PREFIX>_PASSWORD	Defines the password for the datasource.	password
<PREFIX>_DATABASE	Defines the database name for the datasource.	myDatabase

Variable Name	Description	Example Value
<PREFIX>_TX_ISOLATION	Defines the java.sql.Connection transaction isolation level for the database.	TRANSACTION_READ_UNCOMMITTED
<PREFIX>_TX_MIN_POOL_SIZE	Defines the minimum pool size option for the datasource.	1
<PREFIX>_TX_MAX_POOL_SIZE	Defines the maximum pool size option for the datasource.	20

6.7.5.5. Security Environment Variables

The following environment variables may be defined to customize the environment's security domain:

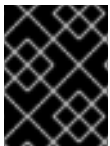
Table 6.6. Security Environment Variables

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

6.7.6. Exposed Ports

The following ports are exposed by default in the JBoss Data Grid xPaaS Image:

Value	Description
8443	Secure Web
8778	-
11211	memcached
11222	internal hotrod
11333	external hotrod



IMPORTANT

The external hotrod connector is only available if the ***HOTROD_SERVICE_NAME*** environment variable has been defined.

6.7.7. Troubleshooting

In addition to viewing the OpenShift logs, you can troubleshoot a running JBoss Data Grid xPaaS Image container by viewing its logs. These are outputted to the container's standard out, and are accessible with the following command:

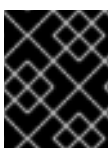
```
$ oc logs -f <pod_name> <container_name>
```



NOTE

By default, the OpenShift JBoss Data Grid xPaaS Image does not have a file log handler configured. Logs are only sent to the container's standard out.

6.8. RED HAT SINGLE SIGN-ON (SSO) XPAAS IMAGE



IMPORTANT

This image is currently in [Technical Preview](#) and not intended for production use.

6.8.1. Overview

Red Hat Single Sign-On (SSO) is an integrated sign-on solution available as a containerized xPaaS image designed for use with OpenShift. This image provides an authentication server for users to centrally log in, log out, register, and manage user accounts for web applications, mobile applications, and RESTful web services.

Red Hat offers five SSO application templates:

- ***sso70-basic***: SSO backed by a H2 database on the same pod
- ***sso70-mysql***: SSO backed by a MySQL database on a separate pod

- **sso70-mysql-persistent:** SSO backed by a persistent PostgreSQL database on a separate pod
- **sso70-postgresql:** SSO backed by a MySQL database on a separate pod
- **sso70-postgresql-persistent:** SSO backed by a persistent PostgreSQL database on a separate pod

An SSO-enabled Red Hat JBoss Enterprise Application Platform (JBoss EAP) Image is also offered, which enables users to deploy a JBoss EAP instance that can be used with SSO for authentication:

- **eap64-sso-s2i:** SSO-enabled JBoss EAP

6.8.2. Differences Between the SSO xPaaS Application and Keycloak

The SSO xPaaS application is based on Keycloak, a JBoss community project. There are some differences in functionality between the Red Hat Single Sign-On xPaaS Application and Keycloak:

- This image is currently available as a Technical Preview for use only with SSO-enabled Red Hat JBoss Enterprise Application Platform (JBoss EAP) applications.
- The SSO xPaaS Technical Preview Application includes all of the functionality of Keycloak 1.8.1. In addition, the SSO-enabled JBoss EAP image automatically handles OpenID Connect or SAML client registration and configuration for **.war** deployments that contain **<auth-method>KEYCLOAK</auth-method>** or **<auth-method>KEYCLOAK-SAML</auth-method>** in their respective **web.xml** files.

6.8.3. Versioning for xPaaS Images

See the xPaaS part of the [OpenShift and Atomic Platform Tested Integrations](#) page for details about OpenShift image version compatibility.

6.8.4. Prerequisites for Deploying the SSO xPaaS Image

The following is a list of prerequisites for using the SSO xPaaS image:

1. **Acquire Red Hat Subscriptions:** Ensure that you have the relevant OpenShift subscriptions as well as a subscription for xPaaS Middleware.
2. **Install OpenShift:** Before using the OpenShift xPaaS images, you must have an OpenShift environment installed and configured:
 - a. The [Quick Installation](#) method allows you to install OpenShift using an interactive CLI utility.
 - b. The [Advanced Installation](#) method allows you to install OpenShift using a reference configuration. This method is best suited for production environments.
3. Ensure the **DNS** has been configured. This is required for communication between JBoss EAP and SSO, and for the requisite redirection. See [DNS](#) for more information.
4. **Install and Deploy Docker Registry:** Install the Docker Registry and then ensure that the Docker Registry is deployed to locally manage images:

```
$ oadm registry --config=/etc/origin/master/admin.kubeconfig \
  --credentials=/etc/origin/master/openshift-registry.kubeconfig
```

For more information, see [Deploying a Docker Registry](#)

5. **Deploy a Router.** For more information, see [Deploying a Router](#).
6. Ensure that you can run the **oc create** command with [cluster-admin](#) privileges.

6.8.5. Using the SSO Image Streams and Application Templates

The Red Hat xPaaS middleware images were [automatically created during the installation](#) of OpenShift along with the other default image streams and templates.

6.8.6. Preparing and Deploying the SSO xPaaS Application Templates

6.8.6.1. Using the OpenShift CLI

1. Prepare the JBoss EAP and SSO application service accounts and their associated secrets.

```
$ oc create -n <project-name> -f <application-
templates_file_path>/secrets/eap-app-secret.json
```

```
$ oc create -n <project-name> -f <application-
templates_file_path>/secrets/sso-app-secret.json
```

2. Deploy one of the SSO application templates. This example deploys the **sso70-postgresql** template, which deploys an SSO pod backed by a PostgreSQL database on a separate pod.

```
$ oc process -f <application-templates_file_path>/sso/sso70-
postgresql.json | oc create -n <project-name> -f -
```

Or, if the template has been imported into common namespace:

```
$ oc new-app --template=sso70-postgresql -n <project-name>
```

6.8.6.2. Using the OpenShift Web Console

Log in to the OpenShift web console:

1. Click **Add to project** to list all of the default image streams and templates.
2. Use the **Filter by keyword** search bar to limit the list to those that match `sso`. You may need to click **See all** to show the desired application template.
3. Click an application template to list all of the deployment parameters. These parameters can be configured manually, or can be left as default.
4. Click **Create** to deploy the application template.

6.8.6.3. Deployment Process

Once deployed, two pods will be created: one for the SSO web servers and one for the database. After the SSO web server pod has started, the web servers can be accessed at their custom configured hostnames, or at the default hostnames:

- <http://sso-<project-name>.<hostname>/auth>: for the web server, and
- <https://secure-sso-<project-name>.<hostname>/auth>: for the encrypted web server.

The default login username/password credentials are *admin/admin*.

6.8.7. Quickstart Example: Using the SSO xPaaS Image with the SSO-enabled JBoss EAP xPaaS Image

This example uses the OpenShift web console to deploy SSO xPaaS backed by a PostgreSQL database. Once deployed, an SSO realm, role, and user will be created to be used when configuring the SSO-enabled JBoss EAP xPaaS Image deployment. Once successfully deployed, the SSO user can then be used to authenticate and access JBoss EAP.

6.8.7.1. Deploy the SSO Application Template

1. Log in to the OpenShift web console and select the <project-name> project space.
2. Click **Add to project** to list all of the default image streams and templates.
3. Use the **Filter by keyword** search bar to limit the list to those that match *sso*. You may need to click **See all** to show the desired application template.
4. Click the **sso70-postgresql** application template to list all of the deployment parameters. These parameters will be left as default for this example.
5. Click **Create** to deploy the application template and start pod deployment. This may take a couple of minutes.

6.8.7.2. Create SSO Credentials

Log in to the encrypted SSO web server at <https://secure-sso-<project-name>.<hostname>/auth> using the default *admin/admin* user name and password.

- **Create a Realm**

1. Create a new realm by hovering your cursor over the realm namespace (default is **Master**) at the top of the sidebar and click the **Add Realm** button.
2. Enter a realm name and click **Create**.

- **Copy the Public Key** In the newly created realm, click the **Keys** tab and copy the public key that has been generated. This will be needed to deploy the SSO-enabled JBoss EAP image.

- **Create a Role** Create a role in SSO with a name that corresponds to the JEE role defined in the **web.xml** of the example application. This role will be assigned to an SSO *application user* to authenticate access to user applications.

1. Click **Roles** in the **Configure** sidebar to list the roles for this realm. As this is a

new realm, there should only be the default *offline_access* role. Click **Add Role**.

2. Enter the role name and optional description and click **Save**.

- **Create Users and Assign Roles** Create two users. The *realm management user* will be assigned the **realm-management** roles to handle automatic SSO client registration in the SSO server. The *application user* will be assigned the JEE role, created in the previous step, to authenticate access to user applications.

Create the *realm management user*:

1. Click **Users** in the **Manage** sidebar to view the user information for the realm. Click **Add User**.
2. Enter a valid **Username** and any additional optional information for the *realm management user* and click **Save**.
3. Edit the user configuration. Click the **Credentials** tab in the user space and enter a password for the user. After the password has been confirmed you can click the **Reset Password** button to set the user password. A pop-up window will prompt for additional confirmation.
4. Click **Role Mappings** to list the realm and client role configuration. In the **Client Roles** drop-down menu, select **realm-management** and add all of the available roles to the user. This provides the user SSO server rights that can be used by the JBoss EAP image to create clients.

Create the *application user*:

1. Click **Users** in the **Manage** sidebar to view the user information for the realm. Click **Add User**.
2. Enter a valid **Username** and any additional optional information for the *application user* and click **Save**.
3. Edit the user configuration. Click the **Credentials** tab in the user space and enter a password for the user. After the password has been confirmed you can click the **Reset Password** button to set the user password. A pop-up window will prompt for additional confirmation.
4. Click **Role Mappings** to list the realm and client role configuration. In **Available Roles**, add the JEE role created earlier.

6.8.7.3. Deploy the SSO-enabled JBoss EAP Image

1. Return to the OpenShift web console and click **Add to project** to list all of the default image streams and templates.
2. Use the **Filter by keyword** search bar to limit the list to those that match *ssso*. You may need to click **See all** to show the desired application template.
3. Click the **eap64-ssso-s2i** image to list all of the deployment parameters. Edit the configuration of the following SSO parameters:
 - **SSO_URI**: The SSO web server authentication address: [https://secure-ssso-
<project-name>.<hostname>/auth](https://secure-ssso-
<project-name>.<hostname>/auth)

- **SSO_REALM**: The SSO realm created for this procedure.
- **SSO_USERNAME**: The name of the *realm management user*.
- **SSO_PASSWORD**: The password of the user.
- **SSO_PUBLIC_KEY**: The public key generated by the realm. It is located in the **Keys** tab of the **Realm Settings** in the SSO console.
- **SSO_BEARER_ONLY**: If set to **true**, the OpenID Connect client will be registered as bearer-only.
- **SSO_ENABLE_CORS**: If set to **true**, the Keycloak adapter enables Cross-Origin Resource Sharing (CORS).

4. Click **Create** to deploy the JBoss EAP image.

It may take several minutes for the JBoss EAP image to deploy. When it does, it can be accessed at:

- `http://<application-name>-<project-name>.<hostname>/<app-context>`: for the web server, and
- `https://secure-<application-name>-<project-name>.<hostname>/<app-context>`: for the encrypted web server, where `<app-context>` is one of `app-jee`, `app-profile-jee`, `app-profile-jee-saml`, or `service` depending on the example application.

6.8.7.3.1. Alternate Deployments

You can also create the client registration in the **Clients** frame of the **Configure** sidebar. Once a client has been registered, click the **Installation** tab and download the configuration **.xml**:

- For OpenID Connect application sources, save the **Keycloak OIDC JBoss Subsystem XML** to `<file_path>/configuration/secure-deployments`.
- For SAML application sources, save the **Keycloak SAML Wildfly/JBoss Subsystem** to `<file_path>/configuration/secure-saml-deployments`.

You can also edit the ***standalone-openshift.xml*** of the JBoss EAP image, which will deploy the manual configuration instead of the default. For more information, see [Using a Modified JBoss EAP xPaaS Image](#).

6.8.7.4. Log in to the JBoss EAP Server Using SSO

1. Access the JBoss EAP application server and click **Login**. You will be redirected to the SSO login.
2. Log in using the SSO user created in the example. You will be authenticated against the SSO server and returned to the JBoss EAP application server.

6.8.8. Known Issues

- There is a known issue with the EAP6 Adapter `HttpServletRequest.logout()` in which the adapter does not log out from the application, which can create a login loop. The workaround is to call `HttpSession.invalidate();` after `request.logout()` to clear the

Keycloak token from the session. For more information, see [KEYCLOAK-2665](#).

- The SSO logs throw a duplication error if the SSO pod is restarted while backed by a database pod. This error can be safely ignored.
- Setting *adminUrl* to a "https://..." address in an OpenID Connect client will cause **javax.net.ssl.SSLHandshakeException** exceptions on the SSO server if the default secrets (**sso-app-secret** and **eap-app-secret**) are used. The application server must use either CA-signed certificates or configure the SSO trust store to trust the self-signed certificates.
- If the client route uses a different domain suffix to the SSO service, the client registration script will erroneously configure the client on the SSO side, causing bad redirection.
- The SSO-enabled JBoss EAP image does not properly set the **adminUrl** property during automatic client registration. As a workaround, log in to the SSO console after the application has started and manually modify the client registration **adminUrl** property to **http://<application-name>-<project-name>.<hostname>/<app-context>**.

CHAPTER 7. REVISION HISTORY: USING IMAGES

7.1. TUE SEP 13 2016

Affected Topic	Description of Change
Other Images → Jenkins	Updated the Jenkins slave template URL within the Using the Jenkins Kubernetes Plug-in to Run Jobs section.
Database Images → MongoDB	Updated some of the language surrounding the Deployment Configuration Object Definition in the Example Template .

7.2. WED JUL 27 2016

Affected Topic	Description of Change
Source-to-Image (S2I) → Python	Updated Python version numbers supported by OpenShift Enterprise.
Source-to-Image (S2I) → PHP	Updated PHP version numbers supported by OpenShift Enterprise.
Database Images → MySQL	Updated MySQL version numbers supported by OpenShift Enterprise.

7.3. WED JUL 20 2016

Affected Topic	Description of Change
Database Images → MySQL	Added MYSQL_SERVICE_HOST and MYSQL_SERVICE_PORT to the Environment Variables section.

7.4. MON JUN 13 2016

Affected Topic	Description of Change
Other Images → Jenkins	Removed specific reference to Jenkins version and added a link to the LTS Changelog.
Red Hat JBoss Web Server xPaaS Images	Updated the binary deployment directory for tomcat STI image to be deployments, not deployment.

7.5. MON MAY 30 2016

Affected Topic	Description of Change
Other Images → Jenkins	Updated the example in the Using Jenkins as a Source-To-Image builder section to use https for GitHub access.

7.6. TUE MAY 03 2016

Affected Topic	Description of Change
Database Images → MySQL	Updated to use the current MySQL script names.

7.7. MON APR 04 2016

Affected Topic	Description of Change
Red Hat JBoss Fuse Integration Services	Added a KCS article link, changed the names of FIS application development workflows, and updated a command in the simple workflow example.

7.8. TUE MAR 29 2016

Affected Topic	Description of Change
Red Hat JBoss Fuse Integration Services	Updated path to image stream JSON file in the OpenShift Source-to-Image (S2I) Workflow section.

7.9. TUE MAR 22 2016

Affected Topic	Description of Change
EAP xPaaS Image 7.0 Beta	Updated the EAP xPaaS image for 7.0 Beta release.
SSO xPaaS Image TP	New topic for the Red Hat SSO xPaaS Technology Preview image.

7.10. MON MAR 21 2016

Affected Topic	Description of Change
MySQL Image	Added a Troubleshooting section.

7.11. MON FEB 22 2016

Affected Topic	Description of Change
Database Images	Updated the commands for creating new databases for MySQL , PostgreSQL , and MongoDB .

7.12. MON FEB 01 2016

Affected Topic	Description of Change
Overview	Added details on which images are supported and their location.

7.13. TUE JAN 26 2016

Affected Topic	Description of Change
xPaaS Middleware Images → Red Hat JBoss Fuse Integration Services	Fixed the KUBERNETES_MASTER value.

7.14. THU NOV 19 2015

OpenShift Enterprise 3.1 release.