OpenShift Enterprise 3.0 Using Images

OpenShift Enterprise 3 Guide to Using Images

Red Hat OpenShift Documentation Team

OpenShift Enterprise 3 Guide to Using Images

Legal Notice

Abstract

Use these topics to find out what different S2I (Source-to-Image), database and Docker images are available for OpenShift Enterprise 3.0 users.

Table of Contents

CHAPTER 1. OVERVIEW

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

Red Hat's official container images are provided in the Red Hat Registry at registry.access.redhat.com. OpenShift'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 users.

2.2. NODE.JS

2.2.1. Overview

OpenShift 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 or by Docker.

2.2.2. Versions

Currently, OpenShift 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 via:

```
$ 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 Docker registry. Additionally, you can create animage stream that points to the image, either in your Docker registry or at the external location. Your OpenShift resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift images.

2.2.4. Configuration

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

2.3. RUBY

2.3. RUBY

2.3.1. Overview

OpenShift 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 or by Docker.

2.3.2. Versions

Currently, OpenShift provides version 2.0 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 via:

```
$ 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 Docker registry. Additionally, you can create animage stream that points to the image, either in your Docker registry or at the external location. Your OpenShift resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift 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 BuildConfig Source Strategy definition.

Table 2.1. Ruby Environment Variables

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	The presence of this variable 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.

2.4. PERL

2.4.1. Overview

OpenShift 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 or by Docker.

2.4.2. Versions

Currently, OpenShift 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 via:

```
$ 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 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 resources can then reference the ImageStream. You can find example image stream definitions for all the provided OpenShift 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 **`.sti/environment`** file inside your source code repository, or define them in the environment section of the BuildConfig Source Strategy 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.

2.5. PHP

2.5.1. Overview

OpenShift 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 or by Docker.

2.5.2. Versions

Currently, OpenShift provides version 5.5 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 via:

```
$ 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 Docker registry. Additionally, you can create animage stream that points to the image, either in your Docker registry or at the external location. Your OpenShift resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift 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 **`.sti/environment`** file inside your source code repository, or define them in the environment section of the BuildConfig Source Strategy 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

Variable name	Description	Default
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.	
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

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 php.ini file.
PHP_INI_SCAN_DIR	Path to scan for additional ini 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. Logs

This image logs primarily to standard out and as such the logs can be viewed via the oc logs command. Access logs are stored in***/tmp/access_log*** which can be viewed using oc exec to access the container.

2.6. PYTHON

2.6.1. Overview

OpenShift 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 or by Docker.

2.6.2. Versions

Currently, OpenShift provides version 3.3 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 via:

```
$ 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 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 resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift 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 BuildConfig Source Strategy 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 **`$(MODULE_NAME):$(VARIABLE_NAME)`**, 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.

CHAPTER 3. DATABASE IMAGES

3.1. OVERVIEW

This topic group includes information on the different database images available for OpenShift 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 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 provides version 5.5 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 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 resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift 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 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.

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 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_LEN	The minimum length of the word to be included in a FULLTEXT index.	4

Variable Name	Description	Default
MYSQL_FT_MAX_WORD_LEN	The maximum length of the word to be included in a FULLTEXT index.	20
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()             |
+------------+--------------------+
| sampledb   | 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()          |
+------------+-----------------+
| sampledb   | root@172.17.42.1 |
+------------+-----------------+
```

3.2.5. 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 MySQL templates should have been registered in the default **openshift** project by your cluster administrator during the First Steps setup process. 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 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 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 **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 3.3. 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 entrypoint 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.

3.3. POSTGRESQL

3.3.1. Overview

OpenShift 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 supports version 9.2 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
```

To use these images, you can either access them directly from these registries or push them into your OpenShift 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 resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift 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/rhscl/postgresql-94-rhel7
```

3.3.4.2. Running PostgreSQL Commands in Containers

OpenShift 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 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
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 First Steps setup process. 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 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 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 provides version 2.4 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
```

To use these images, you can either access them directly from these registries or push them into your OpenShift 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 resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift 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/rhscl/mongodb-26-rhel7
```

3.4.4.2. Running MongoDB Commands in Containers

OpenShift 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.
MONGODB_PASSWORD	Password for the user account.
MONGODB_DATABASE	Database name.

Variable Name	Description
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 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: 2.4.9
connecting to: sampledb
2.4.9
```

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 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 First Steps setup process. 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 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, a *service pod* is defined in the template. This pod starts MongoDB with the **initiate** argument, which instructs the container entrypoint to behave slightly differently than a regular, stand-alone MongoDB database.

Example 3.5. Service Pod Object Definition in the Example Template

```
{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "mongodb-service",
    "creationTimestamp": null,
    "labels": {
      "name": "mongodb-service"
    }
  },
  "spec": {
    "restartPolicy": "Never",
    "dnsPolicy": "ClusterFirst",
    "containers": [
      {
        "name": "initiate",
        "image": "openshift/mongodb-24-centos7",
        "args": ["initiate"],
        "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}"
            }
          ]
        }
      ]
    }
  }
```

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. The service pod runs MongoDB without the **`--auth`** argument, so it can bootstrap the **PRIMARY** member without providing any authentication.

When the user accounts and databases are created and the data are replicated to other members, the service pod then gives up its **PRIMARY** role and shuts down.

Note

It is important that the **`restartPolicy`** field in the service pod is set to**Never** to prevent the service pod from restarting when the container exits.

As soon as the service pod shuts down, other members start a new election and the new **PRIMARY** member is elected from the running members.

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 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. XPAAS MIDDLEWARE IMAGES

5.1. OVERVIEW

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

5.2. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (JBOSS EAP) XPAAS IMAGE

5.2.1. Overview

Red Hat JBoss Enterprise Application Platform (JBoss EAP) is available as a containerized xPaaS image that is designed for use with OpenShift. Developers can quickly build, scale, and test applications deployed across hybrid environments.

Important

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

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

EAP_HOME in this documentation, as in theJBoss EAP documentation, is used refer to the JBoss EAP installation directory. The location of ***EAP_HOME*** inside a JBoss EAP xPaaS image is ***/opt/eap/***, which the **JBOSS_HOME** environment variable is set to by default.

5.2.2. Comparing the JBoss EAP xPaaS Image to the Regular Release of JBoss EAP

5.2.2.1. Functionality Differences for OpenShift JBoss EAP xPaaS Images

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

- The JBoss EAP Management Console is not available to manage OpenShift JBoss EAP xPaaS images.
- The JBoss EAP Management CLI is only bound locally. This means that you can only access the Management CLI of a container from within the pod.
- Domain mode is not supported. OpenShift controls the creation and distribution of applications in the containers.
- The default root page is disabled. You may want to deploy your own application to the root context (as ***ROOT.war***).

- A-MQ is supported for inter-pod and remote messaging. HornetQ is only supported for intra-Pod messaging and only enabled in the absence of A-MQ.

5.2.2.2. Managing OpenShift JBoss EAP xPaaS Images

A major difference in managing an OpenShift JBoss EAP xPaaS image is that there is no Management Console exposed for the JBoss EAP 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 EAP Management CLI (***EAP_HOME/bin/jboss-cli.sh***) is still accessible from within the container for troubleshooting purposes. First open a remote shell session to the running pod:

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

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.

5.2.2.3. Unsupported Configurations

The following is a list of unsupported configurations specific to the JBoss EAP xPaaS image:

- Using MySQL in a scaled environment with XA distributed transactions is not recommended. For applications that support both scaling and XA distributed transactions, PostgreSQL is recommended instead.

5.2.3. Installing the JBoss EAP xPaaS Image Streams and Application Templates

To use the Red Hat xPaaS middleware images in your OpenShift project, you must first install the image streams and Source-to-Image (S2I) application templates.

5.2.4. Running and Configuring the JBoss EAP xPaaS Image

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

5.2.4.1. Using the JBoss EAP xPaaS Image Source-to-Image (S2I) Process

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

The S2I process for the JBoss EAP 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**).

   The results of a successful Maven build are copied to ***EAP_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.

2. Any JAR, WAR, and EAR in the ***deployments*** 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***.

   

   Note

   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***.

5.2.4.1.1. Using a Different JDK Version in the JBoss EAP xPaaS Image

The JBoss EAP xPaaS image may come with multiple versions of OpenJDK installed, but only one is the default. For example, the JBoss EAP 6.4 xPaaS image comes with OpenJDK 1.7 and 1.8 installed, but OpenJDK 1.8 is the default.

If you want the JBoss EAP 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"
}
```

5.2.4.2. Using a Modified JBoss EAP xPaaS Image

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

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

You can run the JBoss EAP xPaaS image in Docker, make the required configuration changes using the JBoss EAP Management CLI (***EAP_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 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.

5.2.5. Troubleshooting

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

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

Note

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

5.3. RED HAT JBOSS A-MQ XPAAS IMAGE

5.3.1. 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.

5.3.2. 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.

5.3.3. Installing the JBoss A-MQ xPaaS Image Streams and Application Templates

To use the Red Hat xPaaS middleware images in your OpenShift project, you must first install the image streams and Source-to-Image (S2I) application templates.

5.3.4. Configuring the JBoss A-MQ Image

5.3.4.1. 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.

5.3.4.2. 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.

5.3.5. Configuring the JBoss A-MQ Persistent Image

5.3.5.1. 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.

5.3.6. 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.

5.3.7. 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.

5.3.8. 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.

5.4. RED HAT JBOSS WEB SERVER XPAAS IMAGES

5.4.1. 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/***.

5.4.2. 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.

5.4.3. Installing the JBoss Web Server xPaaS Image Streams and Application Templates

To use the Red Hat xPaaS middleware images in your OpenShift project, you must first install the image streams and Source-to-Image (S2I) application templates.

Note

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

5.4.4. 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 ***deployment*** 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***.

5.4.5. 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/***.

CHAPTER 6. REVISION HISTORY: USING IMAGES

6.1. TUE MAY 03 2016

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

6.2. MON FEB 22 2016

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

6.3. MON FEB 01 2016

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

6.4. TUE JUN 23 2015

OpenShift Enterprise 3.0 release.