OpenShift Dedicated 3

Using Images

OpenShift Dedicated 3 Guide to Using Images

OpenShift Dedicated 3 Using Images

OpenShift Dedicated 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 Dedicated 3 users.

Table of Contents

CHAPTER 1. OVERVIEW

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

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 Dedicated users.

2.2. .NET CORE

2.2.1. Benefits of Using .NET Core

.NET Core is a general purpose development platform featuring automatic memory management and modern programming languages. It allows users to build high-quality applications efficiently. .NET Core is available on Red Hat Enterprise Linux (RHEL 7) and OpenShift Dedicated via certified containers. .NET Core offers:

- The ability to follow a microservices-based approach, where some components are built with .NET and others with Java, but all can run on a common, supported platform in Red Hat Enterprise Linux and OpenShift Dedicated.

- The capacity to more easily develop new .NET Core workloads on Windows; customers are able to deploy and run on either Red Hat Enterprise Linux or Windows Server.

- A heterogeneous data center, where the underlying infrastructure is capable of running .NET applications without having to rely solely on Windows Server.

- Access to many of the popular development frameworks such as .NET, Java, Ruby, and Python from within OpenShift Dedicated.

2.2.2. Supported Versions

- .NET Core version 2.2

- .NET Core version 2.1

- .NET Core version 1.1

- .NET Core version 1.0

- Supported on Red Hat Enterprise Linux (RHEL) 7

For release details related to .NET Core version 2.2, see Release Notes for Containers.

For release details related to .NET Core version 2.1, see Release Notes for Containers.

Versions 1.1 and 1.0 (**rh-dotnetcore11** and **rh-dotnetcore10**) ship with the *project.json* build system (**1.0.0-preview2** SDK). See the Known Issues chapter in the version 1.1 Release Notes for details on installing this SDK on a non-RHEL system.

2.2.3. Images

The RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/dotnet/dotnet-22-rhel7
```

```
$ docker pull registry.redhat.io/dotnet/dotnet-21-rhel7
$ docker pull registry.redhat.io/dotnet/dotnetcore-11-rhel7
$ docker pull registry.redhat.io/dotnet/dotnetcore-10-rhel7
```

To use these images, you can either access them directly from the image registry or push them into your OpenShift Dedicated container image registry. Additionally, you can create an image stream that points to the image, either in your container image registry or at the external location. Your OpenShift Dedicated resources can then reference the image stream definition.

2.2.4. Build Process

S2I produces ready-to-run images by injecting source code into a container and letting the container prepare that source code for execution. It performs the following steps:

1. Starts a container from the builder image.

2. Downloads the application source.

3. Streams the scripts and application sources into the builder image container.

4. Runs the *assemble* script (from the builder image).

5. Saves the final image.

See S2I Build Process for a detailed overview of the build process.

2.2.5. Environment Variables

The .NET Core images support several environment variables, which you can set to control the build behavior of your .NET Core application.

> **NOTE**
>
> You must set environment variables that control build behavior in the S2I build configuration or in the *.s2i/environment* file to make them available to the build steps.

Table 2.1. NET Core Environment Variables

Variable Name	Description	Default
DOTNET_STARTUP_PROJECT	Selects projects to run. This must be a project file (for example, *csproj* or *fsproj*) or a folder containing a single project file.	**.**
DOTNET_SDK_VERSION	Selects the default SDK version when building. If there is a *global.json* file in the source repository, that takes precedence. When set to **latest**, the latest SDK in the image is used.	Lowest SDK version available in the image.

Variable Name	Description	Default
DOTNET_ASSEMBLY_ NAME	Selects the assembly to run. This must *not* include the **.dll** extension. Set this to the output assembly name specified in *csproj* (**PropertyGroup**/**AssemblyName**).	The name of the *csproj* file.
DOTNET_RESTORE_S OURCES	Specifies the space-separated list of **NuGet** package sources used during the restore operation. This overrides all of the sources specified in the *NuGet.config* file.	
DOTNET_TOOLS	Specifies a list of .NET tools to install before building the application. To install a specific version, add **@<version>** to the end of the package name.	
DOTNET_NPM_TOOLS	Specifies a list of NPM packages to install before building the application.	
DOTNET_TEST_PROJ ECTS	Specifies the list of test projects to test. This must be project files or folders containing a single project file. **dotnet test** is invoked for each item.	
DOTNET_CONFIGURA TION	Runs the application in **Debug** or **Release** mode. This value should be either **Release** or **Debug**.	**Release**
DOTNET_VERBOSITY	Specifies the verbosity of the dotnet build commands. When set, the environment variables are printed at the start of the build. This variable can be set to one of the msbuild verbosity values (**q[uiet]**, **m[inimal]**, **n[ormal]**, **d[etailed]**, and **diag[nostic]**).	
HTTP_PROXY, **HTTPS_PROXY**	Configures the HTTP/HTTPS proxy used when building and running the application.	
NPM_MIRROR	Uses a custom NPM registry mirror to download packages during the build process.	

Variable Name	Description	Default
ASPNETCORE_URLS	This variable is set to **http://*:8080** to configure ASP.NET Core to use the port exposed by the image. Changing this is *not* recommended.	**http://*:8080**
DOTNET_RM_SRC	When set to **true**, the source code is not included in the image.	
DOTNET_SSL_DIRS	Used to specify a list of folders and files with additional SSL certificates to trust. The certificates are trusted by each process that runs during the build and all processes that run in the image after the build, including the application that was built. The items can be absolute paths starting with **/** or paths in the source repository (for example, certificates).	
DOTNET_RESTORE_DISABLE_PARALLEL	When set to **true**, disables restoring multiple projects in parallel. This reduces restore timeout errors when the build container is running with low CPU limits.	**false**

2.2.6. Quickly Deploying Applications from .NET Core Source

> **IMPORTANT**
>
> The .NET image stream must first be installed. If you ran a standard installation, the image stream will be present.

An image can be used to build an application by running **oc new-app** against a sample repository:

```
$ oc new-app registry.redhat.io/dotnet/dotnet-22-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-2.2
--context-dir=app
$ oc new-app registry.redhat.io/dotnet/dotnet-21-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-2.1
--context-dir=app
$ oc new-app registry.redhat.io/dotnet/dotnetcore-11-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-1.1
--context-dir=app
$ oc new-app registry.redhat.io/dotnet/dotnetcore-10-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-1.0
--context-dir=app
```

2.2.7. .NET Core Templates

OpenShift Dedicated includes templates for the .NET Core images to help easily deploy a sample application.

The .NET Core sample application running on **dotnet/dotnet-22-rhel7** can be deployed with:

```
$ oc new-app --template dotnet-example -p
DOTNET_IMAGE_STREAM_TAG=dotnet:2.2 -p SOURCE_REPOSITORY_REF=dotnetcore-2.2
```

The .NET Core sample application running on **dotnet/dotnetcore-10-rhel7** can be deployed with:

```
$ oc new-app --template dotnet-example
```

The .NET Core MusicStore application using PostgreSQL as database can be deployed with:

```
$ oc new-app --template=dotnet-pgsql-persistent
```

2.3. NODE.JS

2.3.1. Overview

OpenShift Dedicated 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 Dedicated or by a container runtime.

2.3.2. Versions

Currently, OpenShift Dedicated provides versions 0.10, 4, and 6 of Node.js.

2.3.3. Images

These images come in two flavors, depending on your needs:

- RHEL 7

- CentOS 7

RHEL 7 Based Images

The RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/openshift3/nodejs-010-rhel7
$ docker pull registry.redhat.io/rhscl/nodejs-4-rhel7
```

CentOS 7 Based Image

This image is available on Docker Hub:

```
$ 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 Dedicated container image registry. Additionally, you can create an image stream that

points to the image, either in your container image registry or at the external location. Your OpenShift Dedicated resources can then reference the ImageStream. You can find example image stream definitions for all the provided OpenShift Dedicated images.

2.3.4. Build Process

S2I produces ready-to-run images by injecting source code into a container and letting the container prepare that source code for execution. It performs the following steps:

1. Starts a container from the builder image.

2. Downloads the application source.

3. Streams the scripts and application sources into the builder image container.

4. Runs the *assemble* script (from the builder image).

5. Saves the final image.

See S2I Build Process for a detailed overview of the build process.

2.3.5. Configuration

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

To set these environment variables as part of your image, you can place them into a *.s2i/environment file* inside your source code repository, or define them in the environment section of the build configuration's **sourceStrategy** definition.

You can also set environment variables to be used with an existing image when creating new applications, or by updating environment variables for existing objects such as deployment configurations.

> **NOTE**
>
> Environment variables that control build behavior must be set as part of the s2i build configuration or in the *.s2i/environment* file to make them available to the build steps.

Table 2.2. Development Mode Environment Variables

Variable name	Description
DEV_MODE	When set to **true**, enables hot deploy and opens the debug port. Additionally, indicates to tooling that the image is in development mode. Default is **false**.
DEBUG_PORT	The debug port. Only valid if **DEV_MODE** is set to true. Default is 5858.
NPM_MIRROR	The custom NPM registry mirror URL. All NPM packages will be downloaded from the mirror link during the build process.

2.3.6. 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 **DEV_MODE=true** environment variable.

You can set new environment variables when creating new applications, or updating environment variables for existing objects.

> ⚠️ **WARNING**
>
> Only use the **DEV_MODE=true** environment variable while developing or debugging. Using this in your production environment is not recommended.

To change the source code of a running pod, open a remote shell into the container:

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

Entering into a running container changes your current directory to */opt/app-root/src*, where the source code is located.

2.4. PERL

2.4.1. Overview

OpenShift Dedicated 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 Dedicated or by a container runtime.

2.4.2. Versions

Currently, OpenShift Dedicated supports versions 5.16, 5.20, and 5.24 of Perl.

2.4.3. Images

Images comes in two flavors, depending on your needs:

- RHEL 7

- CentOS 7

RHEL 7 Based Images

The RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/openshift3/perl-516-rhel7
$ docker pull registry.redhat.io/rhscl/perl-520-rhel7
$ docker pull registry.redhat.io/rhscl/perl-524-rhel7
```

CentOS 7 Based Image

A CentOS image for Perl 5.16 is available on Docker Hub:

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

2.4.4. Build Process

S2I produces ready-to-run images by injecting source code into a container and letting the container prepare that source code for execution. It performs the following steps:

1. Starts a container from the builder image.

2. Downloads the application source.

3. Streams the scripts and application sources into the builder image container.

4. Runs the *assemble* script (from the builder image).

5. Saves the final image.

See S2I Build Process for a detailed overview of the build process.

2.4.5. 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 as part of your image, you can place them into a *.s2i/environment* file inside your source code repository, or define them in the environment section of the build configuration's **sourceStrategy** definition.

You can also set environment variables to be used with an existing image when creating new applications, or by updating environment variables for existing objects such as deployment configurations.

> **NOTE**
>
> Environment variables that control build behavior must be set as part of the s2i build configuration or in the *.s2i/environment* file to make them available to the build steps.

Table 2.3. Perl Environment Variables

Variable name	Description
ENABLE_CPAN_TEST	When set to **true**, this variable installs all the cpan modules and runs their tests. By default, the testing of the modules is turned off.

Variable name	Description
`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.
`HTTPD_START_SERVERS`	The StartServers directive sets the number of child server processes created on startup. Default is 8.
`HTTPD_MAX_REQUEST_WORKERS`	Number of simultaneous requests that will be handled by Apache. The default is 256, but it will be automatically lowered if memory is limited.

2.4.6. 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.

2.4.7. 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 set 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 Dedicated 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 Dedicated or by a container runtime.

2.5.2. Versions

Currently, OpenShift Dedicated provides versions 5.5, 5.6, and 7.0 of PHP.

2.5.3. Images

These images come in two flavors, depending on your needs:

- RHEL 7

- CentOS 7

RHEL 7 Based Images

The RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/openshift3/php-55-rhel7
$ docker pull registry.redhat.io/rhscl/php-56-rhel7
$ docker pull registry.redhat.io/rhscl/php-70-rhel7
```

CentOS 7 Based Images

CentOS images for PHP 5.5 and 5.6 are available on Docker Hub:

```
$ docker pull openshift/php-55-centos7
$ docker pull openshift/php-56-centos7
```

To use these images, you can either access them directly from these image registries or push them into your OpenShift Dedicated container image registry. Additionally, you can create an image stream that points to the image, either in your container image registry or at the external location. Your OpenShift Dedicated resources can then reference the image stream.

You can find example image stream definitions for all the provided OpenShift Dedicated images.

2.5.4. Build Process

S2I produces ready-to-run images by injecting source code into a container and letting the container prepare that source code for execution. It performs the following steps:

1. Starts a container from the builder image.

2. Downloads the application source.

3. Streams the scripts and application sources into the builder image container.

4. Runs the *assemble* script (from the builder image).

5. Saves the final image.

See S2I Build Process for a detailed overview of the build process.

2.5.5. 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 as part of your image, you can place them into a *.s2i/environment file* inside your source code repository, or define them in the environment section of the build configuration's **sourceStrategy** definition.

You can also set environment variables to be used with an existing image when creating new applications, or by updating environment variables for existing objects such as deployment configurations.

> **NOTE**
>
> Environment variables that control build behavior must be set as part of the s2i build configuration or in the *.s2i/environment* file to make them available to the build steps.

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

Table 2.4. 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*
DOCUMENTROOT	Path that defines the document root for your application (for example, */public*).	*/*

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

Table 2.5. Additional PHP settings

Variable Name	Description	Default
OPCACHE_MEMORY_CONSU MPTION	The OPcache shared memory storage size.	**16M**
OPCACHE_REVALIDATE_F REQ	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 2.6. 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

You can use a custom composer repository mirror URL to download packages instead of the default 'packagist.org':

Table 2.7. Composer Environment Variables

Variable Name	Description	**COMPOSER_MIRROR**

2.5.5.1. 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.

2.5.6. 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.

2.5.7. 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 Dedicated 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 Dedicated or by a container runtime.

2.6.2. Versions

Currently, OpenShift Dedicated provides versions 2.7, 3.3, 3.4, and 3.5 of Python.

2.6.3. Images

These images come in two flavors, depending on your needs:

- RHEL 7

- CentOS 7

RHEL 7 Based Images

The RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/rhscl/python-27-rhel7
$ docker pull registry.redhat.io/openshift3/python-33-rhel7
$ docker pull registry.redhat.io/rhscl/python-34-rhel7
$ docker pull registry.redhat.io/rhscl/python-35-rhel7
```

CentOS 7 Based Images

These images are available on Docker Hub:

```
$ docker pull centos/python-27-centos7
$ docker pull openshift/python-33-centos7
$ docker pull centos/python-34-centos7
```

```
$ docker pull centos/python-35-centos7
```

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

2.6.4. Build Process

S2I produces ready-to-run images by injecting source code into a container and letting the container prepare that source code for execution. It performs the following steps:

1. Starts a container from the builder image.

2. Downloads the application source.

3. Streams the scripts and application sources into the builder image container.

4. Runs the *assemble* script (from the builder image).

5. Saves the final image.

See S2I Build Process for a detailed overview of the build process.

2.6.5. 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 as part of your image, you can place them into a *.s2i/environment file* inside your source code repository, or define them in the environment section of the build configuration's **sourceStrategy** definition.

You can also set environment variables to be used with an existing image when creating new applications, or by updating environment variables for existing objects such as deployment configurations.

> **NOTE**
>
> Environment variables that control build behavior must be set as part of the s2i build configuration or in the *.s2i/environment* file to make them available to the build steps.

Table 2.8. 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.

Variable name	Description
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.
PIP_INDEX_URL	Set this variable to use a custom index URL or mirror to download required packages during build process. This only affects packages listed in the *requirements.txt* file.
WEB_CONCURRENCY	Set this to change the default setting for the number of workers. By default, this is set to the number of available cores times 4.

2.6.6. 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 set 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.

2.7. RUBY

2.7.1. Overview

OpenShift Dedicated 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 Dedicated or by a container runtime.

2.7.2. Versions

Currently, OpenShift Dedicated provides versions 2.0, 2.2, and 2.3 of Ruby.

2.7.3. Images

These images come in two flavors, depending on your needs:

- RHEL 7

- CentOS 7

RHEL 7 Based Images

The RHEL 7 images are available through the Red Hat registry:

```
$ docker pull registry.redhat.io/openshift3/ruby-20-rhel7
$ docker pull registry.redhat.io/rhscl/ruby-22-rhel7
$ docker pull registry.redhat.io/rhscl/ruby-23-rhel7
```

CentOS 7 Based Images

These images are available on Docker Hub:

```
$ docker pull openshift/ruby-20-centos7
$ docker pull openshift/ruby-22-centos7
$ docker pull centos/ruby-23-centos7
```

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

2.7.4. Build Process

S2I produces ready-to-run images by injecting source code into a container and letting the container prepare that source code for execution. It performs the following steps:

1. Starts a container from the builder image.

2. Downloads the application source.

3. Streams the scripts and application sources into the builder image container.

4. Runs the *assemble* script (from the builder image).

5. Saves the final image.

See S2I Build Process for a detailed overview of the build process.

2.7.5. 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 as part of your image, you can place them into a *.s2i/environment file* inside your source code repository, or define them in the environment section of the build configuration's **sourceStrategy** definition.

You can also set environment variables to be used with an existing image when creating new applications, or by updating environment variables for existing objects such as deployment configurations.

> **NOTE**
>
> Environment variables that control build behavior must be set as part of the s2i build configuration or in the *.s2i/environment* file to make them available to the build steps.

Table 2.9. 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	When set to **true**, 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.
PUMA_MIN_THREADS, **PUMA_MAX_THREADS**	This variable indicates the minimum and maximum number of threads that will be available in Puma's thread pool.

Variable name	Description
PUMA_WORKERS	This variable indicates the number of worker processes to be launched in Puma's clustered mode (when Puma runs more than two processes). If not explicitly set, the default behavior sets **PUMA_WORKERS** to a value that is appropriate for the memory available to the container and the number of cores on the host.
RUBYGEM_MIRROR	Set this variable to use a custom RubyGems mirror URL to download required gem packages during the build process. Note: This environment variable is only available for Ruby 2.2+ images.

2.7.6. 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 set env** command:

```
$ oc set 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 set 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.

2.8. CUSTOMIZING S2I IMAGES

2.8.1. Overview

S2I builder images normally include ***assemble*** and ***run*** scripts, but the default behavior of those scripts may not be suitable for all users. This topic covers a few approaches for customizing the behavior of an S2I builder that includes default scripts.

2.8.2. Invoking Scripts Embedded in an Image

Typically, builder images provide their own version of the S2I scripts that cover the most common use-cases. If these scripts do not fulfill your needs, S2I provides a way of overriding them by adding custom ones in the *.s2i/bin* directory. However, by doing this you are completely replacing the standard scripts. In some cases this is acceptable, but in other scenarios you may prefer to execute a few commands before (or after) the scripts while retaining the logic of the script provided in the image. In this case, it is possible to create a wrapper script that executes custom logic and delegates further work to the default script in the image.

To determine the location of the scripts inside of the builder image, look at the value of **io.openshift.s2i.scripts-url** label. Use **docker inspect**:

```
$ docker inspect --format='{{ index .Config.Labels
"io.openshift.s2i.scripts-url" }}' openshift/wildfly-100-centos7
image:///usr/libexec/s2i
```

You inspected the **openshift/wildfly-100-centos7** builder image and found out that the scripts are in the */usr/libexec/s2i* directory.

With this knowledge, invoke any of these scripts from your own by wrapping its invocation.

Example 2.1. *.s2i/bin/assemble* script

```
#!/bin/bash
echo "Before assembling"

/usr/libexec/s2i/assemble
rc=$?
```

```
if [ $rc -eq 0 ]; then
    echo "After successful assembling"
else
    echo "After failed assembling"
fi

exit $rc
```

The example shows a custom *assemble* script that prints the message, executes standard *assemble* script from the image and prints another message depending on the exit code of the *assemble* script.

When wrapping the *run* script, you must use **exec** for invoking it to ensure signals are handled properly. Unfortunately, the use of **exec** also precludes the ability to run additional commands after invoking the default image run script.

Example 2.2. *.s2i/bin/run* **script**

```
#!/bin/bash
echo "Before running application"
exec /usr/libexec/s2i/run
```

CHAPTER 3. DATABASE IMAGES

3.1. OVERVIEW

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

3.2. MYSQL

3.2.1. Overview

OpenShift Dedicated provides a container 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 Dedicated provides versions 5.6 and 5.7 of MySQL.

3.2.3. Images

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

- RHEL 7

- CentOS 7

RHEL 7 Based Images

The RHEL 7 image is available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/rhscl/mysql-56-rhel7
$ docker pull registry.redhat.io/rhscl/mysql-57-rhel7
```

CentOS 7 Based Images

CentOS images for MySQL 5.6 and 5.7 are available on Docker Hub:

```
$ docker pull centos/mysql-56-centos7
$ docker pull centos/mysql-57-centos7
```

To use these images, you can either access them directly from these registries or push them into your OpenShift Dedicated container image registry. Additionally, you can create an ImageStream that points to the image, either in your container image registry or at the external location. Your OpenShift Dedicated resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift Dedicated 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> \
    -e MYSQL_PASSWORD=<password> \
    -e MYSQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/mysql-56-rhel7
```

3.2.4.2. Running MySQL Commands in Containers

OpenShift Dedicated 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: 5.6.37 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.

Variable Name	Description
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 Dedicated 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_MAX_ALLOWED_PACKET	The maximum size of one packet or any generated/intermediate string.	200M
MYSQL_FT_MIN_WORD_LEN	The minimum length of the word to be included in a FULLTEXT index.	4
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

Variable Name	Description	Default
`MYSQL_TABLE_OPEN_CACHE`	The number of open tables for all threads.	400
`MYSQL_KEY_BUFFER_SIZE`	The size of the buffer used for index blocks.	32M (or 10% of available memory)
`MYSQL_SORT_BUFFER_SIZE`	The size of the buffer used for sorting.	256K
`MYSQL_READ_BUFFER_SIZE`	The size of the buffer used for a sequential scan.	8M (or 5% of available memory)
`MYSQL_INNODB_BUFFER_POOL_SIZE`	The size of the buffer pool where InnoDB caches table and index data.	32M (or 50% of available memory)
`MYSQL_INNODB_LOG_FILE_SIZE`	The size of each log file in a log group.	8M (or 15% of available memory)
`MYSQL_INNODB_LOG_BUFFER_SIZE`	The size of the buffer that InnoDB uses to write to the log files on disk.	8M (or 15% of available memory)

Some of the memory-related parameters have two default values. The fixed value is used when a container does not have memory limits assigned. The other value is calculated dynamically during a container's startup based on available memory.

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 set 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 set 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 set 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 Dedicated 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.

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 Dedicated deployment.

You can instantiate 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.

3.2.5.1. 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-56-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 Dedicated 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-56-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.

3.2.5.2. 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: {}
```

3.2.5.3. 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.

3.2.6. Troubleshooting

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

3.2.6.1. 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

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**.

Alternatively, 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 Dedicated provides a container 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 Dedicated supports versions 9.4 and 9.5 of PostgreSQL.

3.3.3. Images

These images come in two flavors, depending on your needs:

- RHEL 7

- CentOS 7

RHEL 7 Based Image

The RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/rhscl/postgresql-94-rhel7
$ docker pull registry.redhat.io/rhscl/postgresql-95-rhel7
```

CentOS 7 Based Image

These images are available on Docker Hub:

```
$ docker pull centos/postgresql-94-centos7
$ docker pull centos/postgresql-95-centos7
```

To use these images, you can either access them directly from these registries or push them into your OpenShift Dedicated container image registry. Additionally, you can create an ImageStream that points to the image, either in your container image registry or at the external location. Your OpenShift Dedicated resources can then reference the ImageStream. You can find example ImageStream definitions for all the provided OpenShift Dedicated 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> \
    -e POSTGRESQL_PASSWORD=<password> \
    -e POSTGRESQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/postgresql-95-rhel7
```

3.3.4.2. Running PostgreSQL Commands in Containers

OpenShift Dedicated 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 (9.5.16)
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 Dedicated 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_CONNE CTIONS	Maximum number of client connections allowed.	100
POSTGRESQL_MAX_PREPA RED_TRANSACTIONS	Maximum number of transactions that can be in the "prepared" state. If using prepared transactions, the value should be at least as large as **POSTGRESQL_MAX_CONNECTIONS**.	0
POSTGRESQL_SHARED_BU FFERS	Amount of memory dedicated to PostgreSQL for caching data.	32M

Variable Name	Description	Default
POSTGRESQL_EFFECTIVE _CACHE_SIZE	Estimated amount of memory available for disk caching by the operating system and within PostgreSQL itself.	128M

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 set 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 set 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 set 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 Dedicated 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.

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 Dedicated deployment.

You can instantiate 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 Dedicated provides a container 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 Dedicated provides versions 2.6, 3.2, and 3.4 of MongoDB.

3.4.3. Images

These images come in two flavors, depending on your needs:

- RHEL 7

- CentOS 7

RHEL 7 Based Images

The RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/rhscl/mongodb-26-rhel7
$ docker pull registry.redhat.io/rhscl/mongodb-32-rhel7
$ docker pull registry.redhat.io/rhscl/mongodb-34-rhel7
```

CentOS 7 Based Images

These images are available on Docker Hub:

```
$ docker pull centos/mongodb-26-centos7
$ docker pull centos/mongodb-32-centos7
$ docker pull centos/mongodb-34-centos7
```

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

3.4.4. Configuration and usage

3.4.4.1. Initializing the database

You can configure MongoDB with an ephemeral volume or a persistent volume. The first time you use the 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 with an ephemeral volume:

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

3.4.4.2. Running MongoDB commands in containers

OpenShift Dedicated 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: 2.6.9
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.

Variable Name	Description
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 Dedicated 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**

> **NOTE**
>
> Text search is enabled by default in MongoDB versions 2.6 and higher, and therefore has no configurable parameter.

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⁄data* - 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 set 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 set 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 set 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: sampledb
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: 2.6.9
connecting to: admin
2.6.9
```

3.4.5. Creating a database service from a template

OpenShift Dedicated 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.

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 Dedicated deployment.

You can instantiate 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. MongoDB replication

Red Hat provides a proof-of-concept template for MongoDB replication (clustering) using StatefulSet. 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/sclorg/mongodb-
container/master/examples/petset/mongodb-petset-persistent.yaml
```

> **IMPORTANT**
>
> The example template uses persistent storage. You must have persistent volumes available in your cluster to use this template.

As OpenShift Dedicated automatically restarts unhealthy pods (containers), it will restart replica set members if one or more of these members crashes or fails.

While a replica set member is down or being restarted, it may be one of these scenarios:

1. PRIMARY member is down:
   In this case, the other two members elect a new PRIMARY. Until then, reads are not affected, but the writes fail. After a successful election, writes and reads process normally.

2. One of the SECONDARY member is down:
   Reads and writes are unaffected. Depending on the **oplogSize** configuration and the write rate, the third member might fail to join back the replica set, requiring manual intervention to re-sync its copy of the database.

3. Any two members are down:
   When a three-member replica set member cannot reach any other member, it will step down from the PRIMARY role if it had it. In this case, reads might be served by a SECONDARY member, and writes fail. As soon as one more member is back up, an election picks a new PRIMARY member and reads and writes process normally.

4. All members are down:
   In this extreme case, both reads and writes fail. After two or more members are back up, an election reestablishes the replica set to have a PRIMARY and a SECONDARY member, after which reads and writes process normally.

This is the recommended replication strategy for MongoDB.

NOTE

For production environments, you must maintain as much separation between members as possible. It is recommended to use one or more of the node selection features to schedule StatefulSet pods onto different nodes, and to provide them storage backed by independent volumes.

3.4.6.1. Limitations

- Only MongoDB 3.2 is supported.

- You have to manually update replica set configuration in case of scaling down.

- Changing a user and administrator password is a manual process. It requires:

  - updating values of environment variables in the StatefulSet configuration,

  - changing password in the database, and

  - restarting all pods one after another.

3.4.6.2. Using the example template

Assuming you already have three pre-created persistent volumes or configured persistent volume provisioning.

1. Create a new poject where you want to create a MongoDB cluster:

   ```
   $ oc new-project mongodb-cluster-example
   ```

2. Create a new application using the example template:

```
$ oc new-app https://raw.githubusercontent.com/sclorg/mongodb-
container/master/examples/petset/mongodb-petset-persistent.yaml
```

This command created a a MongoDB cluster with three replica set members.

3. Check the status of the new MongoDB pods:

```
$ oc get pods
NAME          READY      STATUS     RESTARTS    AGE
mongodb-0   1/1        Running    0           50s
mongodb-1   1/1        Running    0           50s
mongodb-2   1/1        Running    0           49s
```

After creating a cluster from the example template, you have a replica set with three members. Once the pods are running you can perform various actions on these pods such as:

- Checking logs for one of the pods:

  ```
  $ oc logs mongodb-0
  ```

- Log in to the pod:

  ```
  $ oc rsh mongodb-0
  sh-4.2$
  ```

- Log into a MongoDB instance:

  ```
  sh-4.2$ mongo $MONGODB_DATABASE -u $MONGODB_USER -p$MONGODB_PASSWORD
  MongoDB shell version: 3.2.6
  connecting to: sampledb
  rs0:PRIMARY>
  ```

3.4.6.3. Scale up

MongoDB recommends an odd number of members in a replica set. If there are sufficient available persistent volumes, or a dynamic storage provisioner is present, scaling up is done by using the **oc scale** command:

```
$ oc scale --replicas=5 statefulsets/mongodb

$ oc get pods
NAME          READY      STATUS     RESTARTS    AGE
mongodb-0   1/1        Running    0           9m
mongodb-1   1/1        Running    0           8m
mongodb-2   1/1        Running    0           8m
mongodb-3   1/1        Running    0           1m
mongodb-4   1/1        Running    0           57s
```

This creates new pods which connect to the replica set and updates its configuration.

NOTE

Scaling up an existing database requires manual intervention if the database size is greater than the **oplogSize** configuration. For such cases, a manual initial sync of the new members is required. For more information, see Check the Size of the Oplog and the MongoDB Replication documentation.

3.4.6.4. Scale down

To scale down a replica set it is possible to go from five to three members, or from three to only one member.

Although scaling up may be done without manual intervention when the preconditions are met (storage availability, size of existing database and **oplogSize**), scaling down always require manual intervention.

To scale down:

1. Set the new number of replicas by using the **oc scale** command:

   ```
   $ oc scale --replicas=3 statefulsets/mongodb
   ```

   If the new number of replicas still constitutes a majority of the previous number, the replica set may elect a new PRIMARY in case one of the pods that was deleted had the PRIMARY member role. For example, when scaling down from five members to three members.

   Alternatively, scaling down to a lower number temporarily renders the replica set to have only SECONDARY members and be in read-only mode. For example, when scaling down from five members to only one member.

2. Update the replica set configuration to remove members that no longer exist.
   This may be improved in the future, a possible implementation being setting a **PreStop** pod hook that inspects the number of replicas (exposed via the downward API) and determines that the pod is being removed from the StatefulSet, and not being restarted for some other reason.

3. Purge the volume used by the decommissioned pods.

3.5. MARIADB

3.5.1. Overview

OpenShift Dedicated provides a container image for running MariaDB. This image can provide database services based on username, password, and database name settings provided in a configuration file.

3.5.2. Versions

Currently, OpenShift Dedicated provides versions 10.0 and 10.1 of MariaDB.

3.5.3. Images

These images come in two flavors, depending on your needs:

- RHEL 7

- CentOS 7

RHEL 7 Based Images

The RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/rhscl/mariadb-100-rhel7
$ docker pull registry.redhat.io/rhscl/mariadb-101-rhel7
```

CentOS 7 Based Images

These images are available on Docker Hub:

```
$ docker pull openshift/mariadb-100-centos7
$ docker pull centos/mariadb-101-centos7
```

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

3.5.4. Configuration and Usage

3.5.4.1. Initializing the Database

The first time you use the shared volume, the database is created along with the database administrator user and the MariaDB root user (if you specify the **MYSQL_ROOT_PASSWORD** environment variable). Afterwards, the MariaDB 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 MariaDB daemon starts.

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

```
$ oc new-app \
    -e MYSQL_USER=<username> \
    -e MYSQL_PASSWORD=<password> \
    -e MYSQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/mariadb-101-rhel7
```

3.5.4.2. Running MariaDB Commands in Containers

OpenShift Dedicated uses Software Collections (SCLs) to install and launch MariaDB. If you want to execute a MariaDB command inside of a running container (for debugging), you must invoke it using bash.

To do so, first identify the name of the running MariaDB 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 **mysql** commands from the bash shell to start a MariaDB interactive session and perform normal MariaDB 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.

3.5.4.3. Environment Variables

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

Table 3.7. MariaDB Environment Variables

Variable Name	Description
MYSQL_USER	User name for MySQL account to be created.
MYSQL_PASSWORD	Password for the user account.
MYSQL_DATABASE	Database name.
MYSQL_ROOT_PASSWORD	Password for the root user (optional).

> **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 Dedicated will continuously try to restart it.

MariaDB settings can be configured with the following environment variables:

Table 3.8. Additional MariaDB Settings

Variable Name	Description	Default
MYSQL_LOWER_CASE_TABLE_NAMES	Sets how the table names are stored and compared.	0

Variable Name	Description	Default
MYSQL_MAX_CONNECTIONS	The maximum permitted number of simultaneous client connections.	151
MYSQL_MAX_ALLOWED_PACKET	The maximum size of one packet or any generated/intermediate string.	200M
MYSQL_FT_MIN_WORD_LEN	The minimum length of the word to be included in a FULLTEXT index.	4
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
MYSQL_TABLE_OPEN_CACHE	The number of open tables for all threads.	400
MYSQL_KEY_BUFFER_SIZE	The size of the buffer used for index blocks.	32M (or 10% of available memory)
MYSQL_SORT_BUFFER_SIZE	The size of the buffer used for sorting.	256K
MYSQL_READ_BUFFER_SIZE	The size of the buffer used for a sequential scan.	8M (or 5% of available memory)
MYSQL_INNODB_BUFFER_POOL_SIZE	The size of the buffer pool where InnoDB caches table and index data.	32M (or 50% of available memory)

Variable Name	Description	Default
`MYSQL_INNODB_LOG_FIL E_SIZE`	The size of each log file in a log group.	8M (or 15% of availab le memor y)
`MYSQL_INNODB_LOG_BUF FER_SIZE`	The size of the buffer that InnoDB uses to write to the log files on disk.	8M (or 15% of availab le memor y)
`MYSQL_DEFAULTS_FILE`	Point to an alternative configuration file.	/etc/my .cnf
`MYSQL_BINLOG_FORMAT`	Set sets the binlog format, supported values are `row` and `statement`.	statem ent

3.5.4.4. Volume Mount Points

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

- */var/lib/mysql/data* - The MySQL data directory is where MariaDB stores database files.

> **NOTE**
>
> When mounting a directory from the host into the container, ensure that the mounted directory has the appropriate permissions. Also verify that the owner and group of the directory match the user name running inside the container.

3.5.4.5. Changing Passwords

Passwords are part of the image configuration, therefore the only supported method to change passwords for the database user (`MYSQL_USER`) and **admin** 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 set env pod <pod_name> --list
```

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 set 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 set 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 MariaDB 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 |
+------------+------------------+
```

3.5.5. Creating a Database Service from a Template

OpenShift Dedicated 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 MariaDB templates should have been registered in the default **openshift** project by your cluster administrator during the initial cluster setup.

There are two templates available:

- **mariadb-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.

- **mariadb-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 Dedicated deployment.

You can instantiate 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 through the service that was defined.

3.5.6. Troubleshooting

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

3.5.6.1. 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

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

Resolution

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**.

Alternatively, 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.

CHAPTER 4. OTHER IMAGES

4.1. OVERVIEW

This topic group includes information on other container images available for OpenShift Dedicated users.

4.2. JENKINS

4.2.1. Overview

OpenShift Dedicated provides a container 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 Dedicated, tests the output of that build, and then on successful build, retags the output to indicate the build is ready for production. For more details, see the README.

OpenShift Dedicated follows the LTS release of Jenkins. OpenShift Dedicated provides an image containing Jenkins 2.x. A separate image with Jenkins 1.x was previously made available but is now no longer maintained.

4.2.2. Images

The OpenShift Dedicated Jenkins image comes in two flavors:

RHEL 7 Based Image

The RHEL 7 image is available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/openshift3/jenkins-2-rhel7
```

CentOS 7 Based Image

This image is available on Docker Hub:

```
$ docker pull openshift/jenkins-2-centos7
```

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

4.2.3. Configuration and Customization

4.2.3.1. Authentication

You can manage Jenkins authentication in two ways:

- OpenShift Dedicated OAuth authentication provided by the OpenShift Login plug-in.

- Standard authentication provided by Jenkins

4.2.3.1.1. OpenShift Dedicated OAuth authentication

OAuth authentication is activated by configuring the **Configure Global Security** panel in the Jenkins UI, or by setting the **OPENSHIFT_ENABLE_OAUTH** environment variable on the Jenkins **Deployment Config** to anything other than **false**. This activates the OpenShift Login plug-in, which retrieves the configuration information from pod data or by interacting with the OpenShift Dedicated API server.

Valid credentials are controlled by the OpenShift Dedicated identity provider. For example, if **Allow All** is the default identity provider, you can provide any non-empty string for both the user name and password.

Jenkins supports both browser and non-browser access.

Valid users are automatically added to the Jenkins authorization matrix at log in, where OpenShift Dedicated **Roles** dictate the specific Jenkins permissions the user will have.

Users with the **admin** role will have the traditional Jenkins administrative user permissions. Users with the **edit** or **view** role will have progressively less permissions. See the Jenkins image source repository README for the specifics on the OpenShift roles to Jenkins permissions mappings.

> **NOTE**
>
> The **admin** user that is pre-populated in the OpenShift Dedicated Jenkins image with administrative privileges will not be given those privileges when OpenShift Dedicated OAuth is used, unless the OpenShift Dedicated cluster administrator explicitly defines that user in the OpenShift Dedicated identity provider and assigns the **admin** role to the user.

Jenkins' users permissions can be changed after the users are initially established. The OpenShift Login plug-in polls the OpenShift Dedicated API server for permissions and updates the permissions stored in Jenkins for each user with the permissions retrieved from OpenShift Dedicated. If the Jenkins UI is used to update permissions for a Jenkins user, the permission changes are overwritten the next time the plug-in polls OpenShift Dedicated.

You can control how often the polling occurs with the **OPENSHIFT_PERMISSIONS_POLL_INTERVAL** environment variable. The default polling interval is five minutes.

The easiest way to create a new Jenkins service using OAuth authentication is to use a template as described below.

4.2.3.1.2. Jenkins Standard Authentication

Jenkins authentication is used by default if the image is run directly, without using a template.

The first time Jenkins starts, the configuration is created along with the administrator user and password. The default user credentials are **admin** and **password**. Configure the default password by setting the **JENKINS_PASSWORD** environment variable when using (and only when using) standard Jenkins authentication.

To create a new Jenkins application using standard Jenkins authentication:

```
$ oc new-app -e \
    JENKINS_PASSWORD=<password> \
    openshift/jenkins-2-centos7
```

4.2.3.2. Environment Variables

The Jenkins server can be configured with the following environment variables:

- **OPENSHIFT_ENABLE_OAUTH** (default: **false**)
  Determines whether the OpenShift Login plug-in manages authentication when logging into Jenkins. To enable, set to **true**.

- **JENKINS_PASSWORD** (default: **password**)
  The password for the **admin** user when using standard Jenkins authentication. Not applicable when **OPENSHIFT_ENABLE_OAUTH** is set to **true**.

- **OPENSHIFT_JENKINS_JVM_ARCH**
  Set to **x86_64** or **i386** to override the JVM used to host Jenkins. For memory efficiency, by default the Jenkins image dynamically uses a 32-bit JVM if running in a container with a memory limit under 2GiB.

- **JAVA_MAX_HEAP_PARAM**
  CONTAINER_HEAP_PERCENT (default: **0.5**, or 50%)
  JENKINS_MAX_HEAP_UPPER_BOUND_MB
  These values control the maximum heap size of the Jenkins JVM. If **JAVA_MAX_HEAP_PARAM** is set (example setting: **-Xmx512m**), its value takes precedence. Otherwise, the maximum heap size is dynamically calculated as **CONTAINER_HEAP_PERCENT**% (example setting: **0.5**, or 50%) of the container memory limit, optionally capped at **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB (example setting: **512**).

  By default, the maximum heap size of the Jenkins JVM is set to 50% of the container memory limit with no cap.

- **JAVA_INITIAL_HEAP_PARAM**
  CONTAINER_INITIAL_PERCENT
  These values control the initial heap size of the Jenkins JVM. If **JAVA_INITIAL_HEAP_PARAM** is set (example setting: **-Xms32m**), its value takes precedence. Otherwise, the initial heap size may be dynamically calculated as **CONTAINER_INITIAL_PERCENT**% (example setting: **0.1**, or 10%) of the dynamically calculated maximum heap size.

  By default, the initial heap sizing is left to the JVM.

- **CONTAINER_CORE_LIMIT**
  If set, specifies an integer number of cores used for sizing numbers of internal JVM threads. Example setting: **2**.

- **JAVA_TOOL_OPTIONS** (default: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**)
  Specifies options to be heeded by all JVMs running in this container. It is not recommended to override this.

- **JAVA_GC_OPTS** (default: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**)

Specifies Jenkins JVM garbage collection parameters. It is not recommended to override this.

- **JENKINS_JAVA_OVERRIDES**

  Specifies additional options for the Jenkins JVM. These options are appended to all other options, including the Java options above, and may be used to override any of them if necessary. Separate each additional option with a space; if any option contains space characters, escape them with a backslash. Example settings: **-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**.

- **JENKINS_OPTS**

  Specifies arguments to Jenkins.

- **INSTALL_PLUGINS**

  Specifies additional Jenkins plug-ins to install when the container is first run or when **OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS** is set to **true** (see below). Plug-ins are specified as a comma-delimited list of name:version pairs. Example setting: **git:3.7.0,subversion:2.10.2**.

- **OPENSHIFT_PERMISSIONS_POLL_INTERVAL** (default: **300000** - 5 minutes)

  Specifies in milliseconds how often the OpenShift Login plug-in polls OpenShift Dedicated for the permissions associated with each user defined in Jenkins.

- **OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG** (default: **false**)

  When running this image with an OpenShift Dedicated persistent volume for the Jenkins config directory, the transfer of configuration from the image to the persistent volume is only done the first startup of the image as the persistent volume is assigned by the persistent volume claim creation. If you create a custom image that extends this image and updates configuration in the custom image after the initial startup, by default it will not be copied over, unless you set this environment variable to **true**.

- **OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS** (default: **false**)

  When running this image with an OpenShift Dedicated persistent volume for the Jenkins config directory, the transfer of plugins from the image to the persistent volume is only done the first startup of the image as the persistent volume is assigned by the persistent volume claim creation. If you create a custom image that extends this image and updates plugins in the custom image after the initial startup, by default they will not be copied over, unless you set this environment variable to **true**.

- **ENABLE_FATAL_ERROR_LOG_FILE** (default: **false**)

  When running this image with an OpenShift Dedicated persistent claim for the Jenkins config directory, this environment variable allows the fatal error log file to persist when a fatal error occurs. The fatal error file is saved at **/var/lib/jenkins/logs**.

- **NODEJS_SLAVE_IMAGE**

  Setting this value overrides the image used for the default NodeJS agent pod configuration. The default NodeJS agent pod uses **docker.io/openshift/jenkins-agent-nodejs-8-centos7** or **registry.redhat.io/openshift3/jenkins-agent-nodejs-8-rhel7** depending whether you are running the CentOS or RHEL version of the Jenkins image. This variable must be set before Jenkins starts the first time for it to have an effect.

- **MAVEN_SLAVE_IMAGE**

  Setting this value overrides the image used for the default maven agent pod configuration. The default maven agent pod uses **docker.io/openshift/jenkins-agent-maven-35-centos7** or **registry.redhat.io/openshift3/jenkins-agent-maven-35-rhel7**

depending whether you are running the CentOS or RHEL version of the Jenkins image. This variable must be set before Jenkins starts the first time for it to have an effect.

4.2.3.3. Cross Project Access

If you are going to run Jenkins somewhere other than as a deployment within your same project, you will need to provide an access token to Jenkins to access your project.

1. Identify the secret for the service account that has appropriate permissions to access the project Jenkins needs to access:

```
$ oc describe serviceaccount jenkins
Name:        default
Labels:      <none>
Secrets:     {  jenkins-token-uyswp    }
             {  jenkins-dockercfg-xcr3d    }
Tokens:      jenkins-token-izv1u
             jenkins-token-uyswp
```

In this case the secret is named **jenkins-token-uyswp**

2. Retrieve the token from the secret:

```
$ oc describe secret <secret name from above> # for example,
jenkins-token-uyswp
Name:        jenkins-token-uyswp
Labels:      <none>
Annotations:    kubernetes.io/service-
account.name=jenkins,kubernetes.io/service-account.uid=32f5b661-
2a8f-11e5-9528-3c970e3bf0b7
Type:   kubernetes.io/service-account-token
Data
====
ca.crt: 1066 bytes
token:  eyJhbGc..<content cut>....wRA
```

The token field contains the token value Jenkins needs to access the project.

4.2.3.4. 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.

4.2.3.5. Customizing the Jenkins Image through Source-To-Image

To customize the official OpenShift Dedicated 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 plug-ins 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 plug-ins you want to copy into Jenkins.

plugins.txt

This file lists the plug-ins you want to install:

```
pluginId:pluginVersion
```

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 Dedicated:

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source:                              1
    git:
      uri: https://github.com/custom/repository
    type: Git
  strategy:                            2
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jenkins:latest
        namespace: openshift
    type: Source
  output:                              3
    to:
      kind: ImageStreamTag
      name: custom-jenkins:latest
```

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

2 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.

4.2.3.6. Configuring the Jenkins Kubernetes Plug-in

The OpenShift Dedicated Jenkins image includes the pre-installed Kubernetes plug-in that allows Jenkins agents to be dynamically provisioned on multiple container hosts using Kubernetes and OpenShift Dedicated.

To use the Kubernetes plug-in, OpenShift Dedicated provides five images suitable for use as Jenkins agents: the *Base*, *Maven*, and *Node.js* images. See Jenkins Agents for more information.

> **NOTE**
>
> the jenkins-slave-maven-* and jenkins-slave-nodejs-* images are being marked as deprecated during the v3.10 release cycle. The images will still exist in the interim so users can migrate their applications to the newer jenkins-agent-maven-* and jenkins-agent-nodejs-* images.

Both the Maven and Node.js agent images are automatically configured as Kubernetes Pod Template images within the OpenShift Dedicated Jenkins image's configuration for the Kubernetes plug-in. That configuration includes labels for each of the images that can be applied to any of your Jenkins jobs under their "Restrict where this project can be run" setting. If the label is applied, execution of the given job will be done under an OpenShift Dedicated pod running the respective agent image.

The Jenkins image also provides auto-discovery and auto-configuration of additional agent images for the Kubernetes plug-in. With the OpenShift Sync plug-in, the Jenkins image on Jenkins start-up searches within the project that it is running, or the projects specifically listed in the plug-in's configuration for the following:

- Image streams that have the label **role** set to **jenkins-slave**.

- Image stream tags that have the annotation **role** set to **jenkins-slave**.

- ConfigMaps that have the label **role** set to **jenkins-slave**.

When it finds an image stream with the appropriate label, or image stream tag with the appropriate annotation, it generates the corresponding Kubernetes plug-in configuration so you can assign your Jenkins jobs to run in a pod running the container image provided by the image stream.

The name and image references of the image stream or image stream tag are mapped to the name and image fields in the Kubernetes plug-in pod template. You can control the label field of the Kubernetes plug-in pod template by setting an annotation on the image stream or image stream tag object with the key **slave-label**. Otherwise, the name is used as the label.

When it finds a ConfigMap with the appropriate label, it assumes that any values in the key-value data payload of the ConfigMap contains XML consistent with the config format for Jenkins and the Kubernetes plug-in pod templates. A key differentiator to note when using ConfigMaps, instead of image streams or image stream tags, is that you can control all the various fields of the Kubernetes plug-in pod template.

The following is an example ConfigMap:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-slave
data:
  template1: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```

```
      <inheritFrom></inheritFrom>
      <name>template1</name>
      <instanceCap>2147483647</instanceCap>
      <idleMinutes>0</idleMinutes>
      <label>template1</label>
      <serviceAccount>jenkins</serviceAccount>
      <nodeSelector></nodeSelector>
      <volumes/>
      <containers>
        <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <name>jnlp</name>
          <image>openshift/jenkins-agent-maven-35-centos7:v3.10</image>
          <privileged>false</privileged>
          <alwaysPullImage>true</alwaysPullImage>
          <workingDir>/tmp</workingDir>
          <command></command>
          <args>${computer.jnlpmac} ${computer.name}</args>
          <ttyEnabled>false</ttyEnabled>
          <resourceRequestCpu></resourceRequestCpu>
          <resourceRequestMemory></resourceRequestMemory>
          <resourceLimitCpu></resourceLimitCpu>
          <resourceLimitMemory></resourceLimitMemory>
          <envVars/>
        </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
      </containers>
      <envVars/>
      <annotations/>
      <imagePullSecrets/>
      <nodeProperties/>
    </org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```

After startup, the OpenShift Sync plug-in monitors the API server of OpenShift Dedicated for updates to **ImageStreams**, **ImageStreamTags**, and **ConfigMaps** and adjusts the configuration of the Kubernetes plug-in.

In particular, the following rules will apply:

- Removal of the label or annotation from the **ConfigMap**, **ImageStream**, or **ImageStreamTag** will result in the deletion of any existing **PodTemplate** from the configuration of the Kubernetes plug-in.

- Similarly, if those objects are removed, the corresponding configuration is removed from the Kubernetes plug-in.

- Conversely, either the creation of appropriately labeled or annotated **ConfigMap**, **ImageStream**, or **ImageStreamTag** objects, or the adding of labels after their initial creation, leads to the creation of a **PodTemplate** in the Kubernetes-plugin configuration.

- In the case of the **PodTemplate** via **ConfigMap** form, changes to the **ConfigMap** data for the **PodTemplate** will be applied to the **PodTemplate** settings in the Kubernetes plug-in configuration, and will override any changes made to the **PodTemplate** through the Jenkins UI in the interim between changes to the **ConfigMap**.

To use a container image as a Jenkins agent, the image must run the slave agent as an entrypoint. For more details about this, refer to the official Jenkins documentation.

4.2.3.6.1. Permission Considerations

In the previous ConfigMap example, the **<serviceAccount>** element of the Pod Template XML is the OpenShift Dedicated Service Account used for the resulting Pod. The service account credentials mounted into the Pod, with permissions associated with the service account, control which operations against the OpenShift Dedicated master are allowed from the Pod.

Consider the following with service accounts used for the Pod, launched by the Kubernetes Plug-in running in the OpenShift Dedicated Jenkins image:

- If you use the example template for Jenkins provided by OpenShift Dedicated, the **jenkins** service account is defined with the **edit** role for the project Jenkins is running in, and the master Jenkins Pod has that service account mounted.

- The two default Maven and NodeJS Pod Templates injected into the Jenkins configuration are also set to use the same service account as the master.

- Any Pod Templates auto-discovered by the OpenShift Sync plug-in as a result of Image streams or Image stream tags having the required label or annotations have their service account set to the master's service account.

- For the other ways you can provide a Pod Template definition into Jenkins and the Kubernetes plug-in, you have to explicitly specify the service account to use.

- Those other ways include the Jenkins console, the **podTemplate** pipeline DSL provided by the Kubernetes plug-in, or labeling a ConfigMap whose data is the XML configuration for a Pod Template.

- If you do not specify a value for the service account, the **default** service account is used.

- You need to ensure that whatever service account is used has the necessary permissions, roles, and so on defined within OpenShift Dedicated to manipulate whatever projects you choose to manipulate from the within the Pod

4.2.4. Usage

4.2.4.1. Creating a Jenkins Service from a Template

Templates provide parameter fields to define all the environment variables (password) with predefined defaults. OpenShift Dedicated 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.

a deployment configuration and a service.

> **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-persistent** uses a persistent volume store. Data survives a pod restart.

must instantiate the template to be able to use Jenkins:

4.2.4.2. Using the Jenkins Kubernetes Plug-in

Creating a New Jenkins Service

In the below sample, the openshift-jee-sample BuildConfig causes a Jenkins maven agent Pod to be dynamically provisioned. The Pod clones some Java source, builds a WAR file, then causes a second BuildConfig (openshift-jee-sample-docker) to run to layer the newly created WAR file into a container image.

A fuller sample which achieves a similar goal is available here.

> **Example 4.1. Example BuildConfig using the Jenkins Kubernetes Plug-in**
>
> ```
> kind: List
> apiVersion: v1
> items:
> - kind: ImageStream
>   apiVersion: v1
>   metadata:
>     name: openshift-jee-sample
> - kind: BuildConfig
>   apiVersion: v1
>   metadata:
>     name: openshift-jee-sample-docker
>   spec:
>     strategy:
>       type: Docker
>     source:
>       type: Docker
>       dockerfile: |-
>         FROM openshift/wildfly-101-centos7:latest
>         COPY ROOT.war /wildfly/standalone/deployments/ROOT.war
>         CMD $STI_SCRIPTS_PATH/run
>       binary:
>         asFile: ROOT.war
>     output:
>       to:
>         kind: ImageStreamTag
>         name: openshift-jee-sample:latest
> - kind: BuildConfig
>   apiVersion: v1
>   metadata:
>     name: openshift-jee-sample
>   spec:
>     strategy:
>       type: JenkinsPipeline
>       jenkinsPipelineStrategy:
>         jenkinsfile: |-
>           node("maven") {
>             sh "git clone https://github.com/openshift/openshift-jee-
> sample.git ."
>             sh "mvn -B -Popenshift package"
>             sh "oc start-build -F openshift-jee-sample-docker --from-
> file=target/ROOT.war"
>           }
>     triggers:
>     - type: ConfigChange
> ```

It is also possible to override the specification of the dynamically created Jenkins agent Pod. The following is a modification to the above example which overrides the container memory and specifies an environment variable:

Example 4.2. Example BuildConfig using the Jenkins Kubernetes Plug-in, specifying memory limit and environment variable

```
kind: BuildConfig
apiVersion: v1
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod",        1
                    cloud: "openshift",     2
                    inheritFrom: "maven",   3
                    containers: [
            containerTemplate(name: "jnlp",   4
                              image: "openshift/jenkins-agent-maven-
35-centos7:v3.10",    5
                              resourceRequestMemory: "512Mi",   6
                              resourceLimitMemory: "512Mi",     7
                              envVars: [
            envVar(key: "CONTAINER_HEAP_PERCENT", value: "0.25")   8
            ])
        ]) {
        node("mypod") {    9
          sh "git clone https://github.com/openshift/openshift-jee-
sample.git ."
          sh "mvn -B -Popenshift package"
          sh "oc start-build -F openshift-jee-sample-docker --from-
file=target/ROOT.war"
        }
      }
  triggers:
  - type: ConfigChange
```

1 A new Pod template called "mypod" is defined on-the-fly. The new Pod template name is referenced in the node stanza below.

2 The "cloud" value must be set to "openshift".

3 The new Pod template can inherit its configuration from an existing Pod template. In this case, we inherit from the "maven" Pod template which is pre-defined by OpenShift Dedicated.

4 We are overriding values in the pre-existing Container, therefore we must specify it by name. All Jenkins agent images shipped with OpenShift Dedicated use the Container name "jnlp".

5 The Container image must be re-specified. This is a known issue.

6 A memory request of 512Mi is specified.

7 A memory limit of 512Mi is specified.

8 An environment variable CONTAINER_HEAP_PERCENT, with value "0.25", is specified.

9 The node stanza references the name of the Pod template newly defined above.

By default the pod is deleted when the build completes. This behavior can be modified via the plug-in or within a pipeline Jenkinsfile - see Agent Pod Retention for further details.

For more information on Kubernetes plug-in configuration, see the Kubernetes plug-in documentation.

4.2.4.3. Memory Requirements

When deployed by the provided Jenkins Ephemeral or Jenkins Persistent templates, the default memory limit is 512MiB.

See Sizing OpenJDK on OpenShift Dedicated for background information on tuning the JVM used by Jenkins.

For memory efficiency, by default the Jenkins image dynamically uses a 32-bit JVM if running in a container with a memory limit under 2GiB. This behavior can be overridden by the **OPENSHIFT_JENKINS_JVM_ARCH** environment variable.

By default the Jenkins JVM uses 50% of the container memory limit for its heap. This value can be modified by the **CONTAINER_HEAP_PERCENT** environment variable. It can also be capped at an upper limit or overridden entirely. See Environment Variables for more details.

Consider that by default all other processes executed in the Jenkins container, such as shell scripts or **oc** commands run locally from pipelines, are not likely to be able to use more than the remaining 256MiB memory combined without provoking an OOM kill. It is therefore highly recommended that pipelines run external commands in a agent container wherever possible.

It is recommended to specify memory request and limit values on agent containers created by the Jenkins Kubernetes Plug-in. As admin, defaults can be set on a per-agent image basis through the Jenkins configuration. The memory request and limit can also be overridden on a per-container basis as documented above.

You can increase the amount of memory available to Jenkins by overriding the **MEMORY_LIMIT** paramenter when instantiating the Jenkins Ephemeral or Jenkins Persistent template.

4.2.5. Jenkins Plug-ins

The following plug-ins are provided to integrate Jenkins with OpenShift Dedicated. They are available by default in the Jenkins image.

4.2.5.1. OpenShift Dedicated Client Plug-in

The OpenShift Dedicated Client Plug-in aims to provide a readable, concise, comprehensive, and fluent Jenkins Pipeline syntax for rich interactions with OpenShift Dedicated. The plug-in leverages the **oc** binary, which must be available on the nodes executing the script.

This plug-in is fully supported and is included in the Jenkins image. It provides:

- A Fluent-style syntax for use in Jenkins Pipelines.

- Use of and exposure to any option available with **oc**.

- Integration with Jenkins credentials and clusters.

- Continued support for classic Jenkins Freestyle jobs.

See the OpenShift Pipeline Builds tutorial and the plug-in's README for more information.

4.2.5.2. OpenShift Dedicated Pipeline Plug-in

The OpenShift Dedicated Pipeline Plug-in is a prior integration between Jenkins and OpenShift Dedicated which provides less functionality than the OpenShift Dedicated Client Plug-in. It has been deprecated but continues to work with OpenShift Dedicated versions up to v3.11. For later verions of OpenShift Dedicated, either use the **oc** binary directly from your Jenkins Pipelines, or use the OpenShift Dedicated Client Plug-in.

See the plug-in's README for more information.

4.2.5.3. OpenShift Dedicated Sync Plug-in

To facilitate OpenShift Dedicated Pipeline build strategy for integration between Jenkins and OpenShift Dedicated, the OpenShift Sync Plug-in monitors the API server of OpenShift Dedicated for updates to **BuildConfigs** and **Builds** that employ the Pipeline strategy and either creates Jenkins Pipeline projects (when a **BuildConfig** is created) or starts jobs in the resulting projects (when a **Build** is started).

As noted in Configuring the Jenkins Kubernetes Plug-in, this plug-in can create **PodTemplate** configurations for the Kubernetes plug-in based on specifically cited **ImageStream**, **ImageStreamTag**, or **ConfigMap** objects defined in OpenShift Dedicated.

This plug-in can now take **Secret** objects with a label key of **credential.sync.jenkins.openshift.io** and label value of **true** and construct Jenkins credentials which are placed in the default global domain within the Jenkins credentials hierarchy. The ID of the credential will be composed of the namespace the **Secret** is defined in, a hyphen (**-**), followed by the name of the **Secret**.

Similar to the handling of **ConfigMaps** for **PodTemplates**, the **Secret** object defined in OpenShift Dedicated is considered the master configuration. Any subsequent updates to the object in OpenShift Dedicated will be applied to the Jenkins credential (overwriting any changes to the credential made in the interim).

Removal of the **credential.sync.jenkins.openshift.io** property, setting of that property to something other than **true**, or deletion of the **Secret** in OpenShift Dedicated will result in deletion of the associated credential in Jenkins.

The type of secret will be mapped to the jenkins credential type as follows:

- With Opaque type **Secret** objects the plug-in looks for **username** and **password** in the **data** section and constructs a Jenkins UsernamePasswordCredentials credential. Remember, in OpenShift Dedicated the **password** field can be either an actual password or the user's unique token. If those are not present, it will look for the **ssh-privatekey** field and create a Jenkins BasicSSHUserPrivateKey credential.

- With **kubernetes.io/basic-auth** type `Secret` objects the plug-in creates a Jenkins UsernamePasswordCredentials credential.

- With **kubernetes.io/ssh-auth** type **Secret** objects the plug-in creates a Jenkins BasicSSHUserPrivateKey credential.

4.2.5.4. Kubernetes Plug-in

The Kubernetes plug-in is used to run Jenkins agents as pods on your cluster. The auto-configuration of the Kubernetes plug-in is described in Using the Jenkins Kubernetes Plug-in.

4.3. JENKINS AGENTS

4.3.1. Overview

OpenShift Dedicated provides three images suitable for use as Jenkins agents: the *Base*, *Maven*, and *Node.js* images.

The first is a base image for Jenkins agents:

- It pulls in both the required tools (headless Java, the Jenkins JNLP client) and the useful ones (including git, tar, zip, and nss among others).

- It establishes the JNLP agent as the entrypoint.

- It includes the **oc** client tooling for invoking command line operations from within Jenkins jobs.

- It provides Dockerfiles for both CentOS and RHEL images.

Two more images that extend the base image are also provided:

- Maven v3.5 image

- Node.js v8 image

The Maven and Node.js Jenkins agent images provide Dockerfiles for both CentOS and RHEL that you can reference when building new agent images. Also note the **contrib** and **contrib/bin** subdirectories. They allow for the insertion of configuration files and executable scripts for your image.

> **IMPORTANT**
>
> Use and extend an appropriate agent image version for the version of OpenShift Dedicated that you are using. If the **oc** client version embedded in the agent image is not compatible with the OpenShift Dedicated version, unexpected behavior may result. See the versioning policy for more information.

4.3.2. Images

The OpenShift Dedicated Jenkins agent images come in two flavors:

RHEL 7 Based Images

RHEL 7 images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/openshift3/jenkins-slave-base-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-slave-maven-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-slave-nodejs-rhel7
```

```
$ docker pull registry.redhat.io/openshift3/jenkins-agent-maven-35-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-agent-nodejs-8-rhel7
```

CentOS 7 Based Images

These images are available on Docker Hub:

```
$ docker pull openshift/jenkins-slave-base-centos7
$ docker pull openshift/jenkins-slave-maven-centos7
$ docker pull openshift/jenkins-slave-nodejs-centos7
$ docker pull openshift/jenkins-agent-maven-35-centos7
$ docker pull openshift/jenkins-agent-nodejs-8-centos7
```

To use these images, you can either access them directly from these registries or push them into your OpenShift Dedicated container image registry.

4.3.3. Configuration and Customization

4.3.3.1. Environment Variables

Each Jenkins agent container can be configured with the following environment variables:

- **OPENSHIFT_JENKINS_JVM_ARCH**
  Set to **x86_64** or **i386** to override the JVM used to host the Jenkins agent. For memory efficiency, by default the Jenkins agent images dynamically use a 32-bit JVM if running in a container with a memory limit under 2GiB.

- **JAVA_MAX_HEAP_PARAM**
  CONTAINER_HEAP_PERCENT (default: **0.5**, i.e. 50%)
  JNLP_MAX_HEAP_UPPER_BOUND_MB
  These values control the maximum heap size of the Jenkins agent JVM. If **JAVA_MAX_HEAP_PARAM** is set (example setting: **-Xmx512m**), its value takes precedence. Otherwise, the maximum heap size is dynamically calculated as **CONTAINER_HEAP_PERCENT**% (example setting: **0.5**, i.e. 50%) of the container memory limit, optionally capped at **JNLP_MAX_HEAP_UPPER_BOUND_MB** MiB (example setting: **512**).

  By default, the maximum heap size of the Jenkins agent JVM is set to 50% of the container memory limit with no cap.

- **JAVA_INITIAL_HEAP_PARAM**
  CONTAINER_INITIAL_PERCENT
  These values control the initial heap size of the Jenkins agent JVM. If **JAVA_INITIAL_HEAP_PARAM** is set (example setting: **-Xms32m**), its value takes precedence. Otherwise, the initial heap size may be dynamically calculated as **CONTAINER_INITIAL_PERCENT**% (example setting: **0.1**, i.e. 10%) of the dynamically calculated maximum heap size.

  By default, the initial heap sizing is left to the JVM.

- **CONTAINER_CORE_LIMIT**
  If set, specifies an integer number of cores used for sizing numbers of internal JVM threads. Example setting: **2**.

- **JAVA_TOOL_OPTIONS** (default: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**)
  Specifies options to be heeded by all JVMs running in this container. It is not recommended to override this.

- **JAVA_GC_OPTS** (default: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**)
  Specifies Jenkins agent JVM garbage collection parameters. It is not recommended to override this.

- **JNLP_JAVA_OVERRIDES**
  Specifies additional options for the Jenkins agent JVM. These options are appended to all other options, including the Java options above, and may be used to override any of them if necessary. Separate each additional option with a space; if any option contains space characters, escape them with a backslash. Example settings: **-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**.

4.3.4. Usage

4.3.4.1. Memory Requirements

A JVM is used in all Jenkins agents to host the Jenkins JNLP agent, as well as to run any Java applications (e.g. **javac**, Maven or Gradle). See Sizing OpenJDK on OpenShift Dedicated for background information on tuning the JVM used by Jenkins agents.

For memory efficiency, by default the Jenkins image dynamically uses a 32-bit JVM if running in a container with a memory limit under 2GiB. This behavior can be overridden by the **OPENSHIFT_JENKINS_JVM_ARCH** environment variable. The JVM choice applies by default both for the Jenkins JNLP agent as well as for any other Java processes within the agent container.

By default the Jenkins JNLP agent JVM uses 50% of the container memory limit for its heap. This value can be modified by the **CONTAINER_HEAP_PERCENT** environment variable. It can also be capped at an upper limit or overridden entirely. See Environment Variables for more details.

Consider that by default any/all other processes executed in the Jenkins agent container, e.g. shell scripts or **oc** commands run from pipelines, may not be able to use more than the remaining 50% memory limit without provoking an OOM kill.

By default, each further JVM process run in a Jenkins agent container will use up to 25% of the container memory limit for their heap. It may be necessary to tune this for many build workloads. See Sizing OpenJDK on OpenShift Dedicated for more information.

See the Jenkins documentation for information on specifying the memory request and limit of a Jenkins agent container.

4.3.4.1.1. Gradle builds

Hosting Gradle builds in the a Jenkins agent on OpenShift presents additional complications, not least because in addition to the Jenkins JNLP agent and Gradle JVMs, Gradle spawns a third JVM to run tests, if these are specified.

See Sizing OpenJDK on OpenShift Dedicated for background information on tuning JVMs on OpenShift.

The following settings are suggested as a starting point for running Gradle builds in a memory constrained Jenkins agent on OpenShift. Settings may be relaxed subsequently as required.

- Ensure the long-lived gradle daemon is disabled by adding **org.gradle.daemon=false** to the gradle.properties file.

- Disable parallel build execution by ensuring **org.gradle.parallel=true** is not set in the gradle.properties file and that **--parallel** is not set as a command line argument.

- Set **java { options.fork = false }** in the build.gradle file to prevent Java compilations running out-of-process.

- Disable multiple additional test processes by ensuring **test { maxParallelForks = 1 }** is set in the build.gradle file.

- Override the gradle JVM memory parameters according to Sizing OpenJDK on OpenShift Dedicated by the GRADLE_OPTS, JAVA_OPTS or JAVA_TOOL_OPTIONS environment variables.

- Set the maximum heap size and JVM arguments for any Gradle test JVM by the maxHeapSize and jvmArgs settings in build.gradle, or though the **-Dorg.gradle.jvmargs** command line argument.

4.3.5. Agent Pod Retention

Jenkins agent pods (also known as slave pods) are deleted by default after the build completes or is aborted. This behavior can be changed by the Kubernetes plug-in *Pod Retention* setting. Pod retention can be set for all Jenkins builds, with overrides for each pod template. The following behaviors are supported:

- *Always* keeps the build pod regardless of build result.

- *Default* uses the plug-in value (pod template only).

- *Never* always deletes the pod.

- *On Failure* keeps the pod if it fails during the build.

You can override pod retention in the pipeline Jenkinsfile:

```
podTemplate(label: "mypod",
  cloud: "openshift",
  inheritFrom: "maven",
  podRetention: onFailure(),    1
  containers: [
    ...
  ]) {
  node("mypod") {
    ...
  }
}
```

1 Allowed values for **podRetention** are **never()**, **onFailure()**, **always()**, and **default()**.

> **WARNING**
>
> Pods that are kept may continue to run and count against resource quotas.

4.4. OTHER CONTAINER IMAGES

If you want to use container images not found in the Red Hat Container Catalog, you can use other arbitrary container images in your OpenShift Dedicated instance, for example those found on the Docker Hub.

For OpenShift Dedicated-specific guidelines on running containers using an arbitrarily assigned user ID, see Support Arbitrary User IDs in the Creating Images guide.

CHAPTER 5. XPAAS MIDDLEWARE IMAGES

5.1. OVERVIEW

Red Hat offers a containerized xPaaS image for a host of middleware products that are designed for use with OpenShift Dedicated. The documentation for these images is in the Red Hat Customer Portal.