Red Hat Software Collections 3

Using Red Hat Software Collections Container Images

Basic Usage Instructions for Red Hat Software Collections 3.1 Container images

Last Updated: 2018-09-05

Red Hat Software Collections 3 Using Red Hat Software Collections Container Images

Basic Usage Instructions for Red Hat Software Collections 3.1 Container images

Lenka Špačková
lspackova@redhat.com

Robert Krátký
rkratky@redhat.com

Vladimír Slávik
vslavik@redhat.com

Legal Notice

Abstract

This document provides instructions for obtaining, configuring, and using container images that are shipped with Red Hat Software Collections.

Table of Contents

PREFACE

This version of the document is provided only as a preview. It is under development and is subject to substantial change. Consider the included information incomplete and use it with caution.

As a part of the Red Hat Software Collections offering, Red Hat provides a number of container images, which are based on the corresponding Software Collections. These include application, daemon, and database images. The provided images are detailed in the tables:

- Chapter 3, *Container Images Based on Red Hat Software Collections 3.1*
- Chapter 4, *Container Images Based on Red Hat Software Collections 3.0*

You can use these images in a containerized environment to build, deploy, and run your applications.

For more information on containers and container images, see the Core Concepts of the OpenShift Enterprise 3.9 Architecture, which discusses core concepts and methods related to delivering containerized applications.

For more information on Software Collections, see the Red Hat Software Collections and Red Hat Developer Toolset documentation.

NOTE

Running Red Hat Software Collections container images is supported only on Red Hat Enterprise Linux 7 Server and Red Hat Enterprise Linux Atomic Host. You cannot run the images on Red Hat Enterprise Linux 7 Workstation or Red Hat Enterprise Linux 6 or earlier.

When using SELinux for controlling processes within a container, make sure that any content that is volume mounted into the container is readable, and potentially writable, depending on the use case. For more information, see Using Volumes With the docker Container Can Cause Problems With SELinux.

There are two basic approaches that you can take to use the container images shipped with Red Hat Software Collections: using base images or using Source-to-Image.

CHAPTER 1. USING BASE IMAGES

To use container images provided by Red Hat as base images in your own Dockerfile, add the following line to it:

```
FROM registry.access.redhat.com/rhscl/python-35-rhel7
```

Working with Dockerfiles is covered in the Red Hat Enterprise Linux Atomic Host 7 Getting Started with Containers document. Detailed information on Dockerfiles can be found in the Dockerfile reference document.

CHAPTER 2. USING SOURCE-TO-IMAGE (S2I)

Source-to-Image (S2I) is a framework and a tool that allows you to write images which use the application source code as an input and produce a new image that runs the assembled application as an output. The main advantage of using the S2I tool for building reproducible container images is the ease of use for developers.

To use the S2I tool on your system, subscribe to Red Hat Software Collections and run the following command to install the **source-to-image** package:

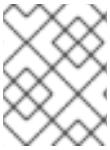
```
# yum install source-to-image
```

Use the RHSM channel: **rhel-server-rhscl-7-rpms**. Note that the **source-to-image** package requires the **docker** package from the Red Hat Enterprise Linux Extras channel.

Alternatively, you can use the **rhel-x86_64-server-7-rhscl-1** RHN channel, but note that the RHN channel is accessible only through Red Hat Satellite instances.

For details about subscribing to Red Hat Software Collections, see Getting Access to Red Hat Software Collections.

More information about the S2I tool is available at GitHub.



NOTE

Similarly to Red Hat Software Collections container images, the S2I tool runs only on Red Hat Enterprise Linux 7 Server, not on Red Hat Enterprise Linux 7 Workstation.

2.1. BUILD PROCESS

The build process consists of the following three fundamental elements, which are combined into a final container image:

- The source code of your application—written in a programming language or framework.
- Builder image—container image provided by Red Hat that supports building images using the S2I tool.
- S2I scripts that are part of the builder image.

During the build process, S2I creates a tar file that contains the source code and scripts, then streams that file into the builder image.

For more information on the Source-to-Image framework, see S2I Requirements.

2.2. EXAMPLE: BUILDING A PYTHON APPLICATION FROM GIT USING S2I

This example shows how to build:

- A new container image from the **python-35-rhel7** builder image that is available in the Red Hat Container Registry, and

- A test application available from a public Git repository in the GitHub sti-python repository, in the **`3.5/test/setup-test-app/`** directory.

  1. Install the S2I tool from the Red Hat Software Collections repository:

     ```
     # yum install source-to-image
     ```

  2. Pull the builder image:

     ```
     # docker pull registry.access.redhat.com/rhscl/python-35-rhel7
     ```

  3. Build the test application from the GitHub sti-python repository, in the **`3.5/test/setup-test-app/`** directory:

     ```
     # s2i build https://github.com/openshift/sti-python.git --
     context-dir=3.5/test/setup-test-app/ rhscl/python-35-rhel7
     python-35-rhel7-app
     ```

     This produces a new application image, **python-35-rhel7-app**.

  4. Run the resulting **python-35-rhel7-app** image:

     ```
     # docker run -d -p 8080:8080 --name example-app python-35-rhel7-
     app
     ```

  5. Fetch a document from **`http://localhost:8080/`**:

     ```
     $ wget http://localhost:8080/
     ```

     The example document is returned.

  6. Stop the container:

     ```
     # docker stop example-app
     ```

CHAPTER 3. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.1

Component	Description
Application Images	
rhscl/php-70-rhel7	PHP 7.0 platform for building and running applications
rhscl/perl-526-rhel7	Perl 5.26 platform for building and running applications
rhscl/ruby-25-rhel7	Ruby 2.5 platform for building and running applications
Daemon Images	
rhscl/httpd-24-rhel7	Apache HTTP 2.4 Server
rhscl/varnish-5-rhel7	Varnish Cache 5.0 HTTP reverse proxy
Database Images	
rhscl/mongodb-36-rhel7	MongoDB 3.6 NoSQL database server
rhscl/postgresql-10-rhel7	PostgreSQL 10 SQL database server
Red Hat Developer Toolset 7.1 Images	
rhscl/devtoolset-7-toolchain-rhel7	Red Hat Developer Toolset toolchain
rhscl/devtoolset-7-perftools-rhel7	Red Hat Developer Toolset perftools

All images are based on components from Red Hat Software Collections. The images are available for Red Hat Enterprise Linux 7 through the Red Hat Container Registry.

For detailed information about components provided by Red Hat Software Collections 3.1, see the Red Hat Software Collections 3.1 Release Notes.

For more information about the Red Hat Developer Toolset 7.1 components, see the Red Hat Developer Toolset 7.1 User Guide.

For information regarding container images based on Red Hat Software Collections 2, see Using Red Hat Software Collections 2 Container Images.

CHAPTER 4. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.0

Component	Description
Application Images	
rhscl/nodejs-8-rhel7	Node.js 8 platform for building and running applications
rhscl/php-71-rhel7	PHP 7.1 platform for building and running applications
rhscl/python-36-rhel7	Python 3.6 platform for building and running applications
Daemon Images	
rhscl/httpd-24-rhel7	Apache HTTP 2.4 Server
rhscl/nginx-112-rhel7	nginx 1.12 server and a reverse proxy server
Database Images	
rhscl/mariadb-102-rhel7	MariaDB 10.2 SQL database server
rhscl/mongodb-34-rhel7	MongoDB 3.4 NoSQL database server
rhscl/postgresql-96-rhel7	PostgreSQL 9.6 SQL database server

All images are based on components from Red Hat Software Collections. The images are available for Red Hat Enterprise Linux 7 through the Red Hat Container Registry.

For detailed information about components provided by Red Hat Software Collections 3.0, see the Red Hat Software Collections 3.0 Release Notes.

For more information about the Red Hat Developer Toolset 7.0 components, see the Red Hat Developer Toolset 7 User Guide.

For information regarding container images based on Red Hat Software Collections 2, see the Using Red Hat Software Collections 2 Container Images.

CHAPTER 5. EXTENDING EXISTING CONTAINER IMAGES

To extend a functionality of a container image provided by Red Hat:

- Set environment variables or use OpenShift secrets.
- If the desired changes cannot be achieved that way, build a new container on top of the provided container image. Write a custom Dockerfile and use the original container in the FROM clause. See the example described in the Knowledgebase article How to Extend the rhscl/mariadb-101-rhel7 Container Image.

CHAPTER 6. APPLICATION IMAGES

6.1. NODE.JS

6.1.1. Description

The **rhscl/nodejs-8-rhel7** image provides a Node.js 8 platform for building and running applications.

6.1.2. Access

To pull the **rhscl/nodejs-8-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/nodejs-8-rhel7
```

6.1.3. Configuration

To set environment variables, you can place them as a key-value pair into a **`.sti/environment`** file inside your source code repository.

Variable Name	Description
`NODE_ENV`	NodeJS runtime mode (default: "production")
`DEV_MODE`	When set to "true", **nodemon** will be used to automatically reload the server while you work (default: "false"). Setting **`DEV_MODE`** to "true" will change the **`NODE_ENV`** default to "development" (if not explicitly set).
`NPM_RUN`	Select an alternate / custom runtime mode, defined in your **`package.json`** file's **`scripts`** section (default: npm run "start"). These user-defined run-scripts are unavailable while **`DEV_MODE`** is in use.
`HTTP_PROXY`	Use an npm proxy during assembly
`HTTPS_PROXY`	Use an npm proxy during assembly
`NPM_MIRROR`	Use a custom NPM registry mirror to download packages during the build process

6.2. PHP

6.2.1. Description

The **rhscl/php-71-rhel7** image provides a PHP 7.1 platform for building and running applications; the **rhscl/php-70-rhel7** image provides a PHP 7.0 platform. **`Node.js`** with **`npm`** is preinstalled in the PHP images.

6.2.2. Access

To pull the **rhscl/php-71-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/php-71-rhel7
```

To pull the **rhscl/php-70-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/php-70-rhel7
```

6.2.3. Configuration

To set environment variables, place them as a key-value pair into a **.sti/environment** file inside your source code repository.

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

Variable Name	Description	Default
ERROR_REPORTING	Informs PHP of which errors, warnings and notices you would like it to take action for	**E_ALL & ~E_NOTICE**
DISPLAY_ERRORS	Controls whether or not and where PHP will output errors, notices and arnings	**ON**
DISPLAY_STARTUP_ERRORS	Cause display errors which occur during PHP's startup sequence to be handled separately from display errors	**OFF**
TRACK_ERRORS	Store the last error/warning message in **$php_errormsg** (boolean)	**OFF**
HTML_ERRORS	Link errors to documentation related to the error	**ON**
INCLUDE_PATH	Path for PHP source files	**.:/opt/app-root/src:/opt/rh/rh-php71/root/usr/share/pear**
SESSION_NAME	Name of the session	**PHPSESSID**
SESSION_HANDLER	Method for saving sessions	`files
SESSION_PATH	Location for session data files	**/tmp/sessions**

Variable Name	Description	Default
SESSION_COOKIE_DOMAIN	The domain for which the cookie is valid	
SESSION_COOKIE_HTTPONLY	Whether or not to add the httpOnly flag to the cookie	**0**
SESSION_COOKIE_SECURE	Specifies whether cookies should only be sent over secure connections	**Off**
SHORT_OPEN_TAG	Determines whether or not PHP will recognize code between <? and ?> tags	OFF
DOCUMENTROOT	Path that defines the DocumentRoot for your application (ie. /public)	/

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

Variable Name	Description	Default
OPCACHE_MEMORY_CONSUMPTION	The OPcache shared memory storage size	16M
OPCACHE_REVALIDATE_FREQ	How often to check script timestamps for updates, in seconds. 0 will result in OPcache checking for updates on every request.	2

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

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 override the Apache MPM prefork settings to increase the performance for of the PHP application. In case you set the Cgroup limits in Docker, the image will attempt to automatically set the optimal values. You can override this at any time by specifying the values yourself:

Variable Name	Description	Default
HTTPD_START_SERVERS	The StartServers directive sets the number of child server processes created on startup.	8
HTTPD_MAX_REQUEST_WORKERS	The MaxRequestWorkers directive sets the limit on the number of simultaneous requests that will be served. **MaxRequestWorkers** was called **MaxClients** before version httpd 2.3.13.	256 (this is automatically tuned by setting Cgroup limits for the container using this formula: **TOTAL_MEMORY / 15MB**. The 15MB is average size of a single **httpd** process.

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

Variable Name	Description
COMPOSER_MIRROR	Adds a custom composer repository mirror URL to composer configuration. Note: This only affects packages listed in **composer.json**.

In case the DocumentRoot of the application is nested within the source directory **/opt/app-root/src**, users can provide their own **.htaccess** file. This allows the overriding of Apache's behavior and specifies how application requests should be handled. The **.htaccess** file needs to be located at the root of the application source. For details about **.htaccess**, see the Apache HTTP Server Tutorial.

6.3. PERL

6.3.1. Description

The **rhscl/perl-526-rhel7** image provides a Perl 5.26 platform for building and running applications. **Apache httpd 2.4** with **mod_perl** for deploying Perl web applications is preinstalled, as well as **Node.js** with **npm**.

The image also supports deploying Perl Web Server Gateway Interface (PSGI) applications.

6.3.2. Access

To pull the **rhscl/perl-526-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/perl-526-rhel7
```

6.3.3. Configuration

To set environment variables, you can place them as a key-value pair into a **.sti/environment** file inside your source code repository.

Variable Name	Description	Default
ENABLE_CPAN_TEST	Allows the installation of all specified cpan packages and the running of their tests	**false**
CPAN_MIRROR	Specifies a mirror URL which will used by cpanminus to install dependencies	URL is not specified by default
PERL_APACHE2_RELOAD	Enables automatic reloading of modified Perl modules	**false**
HTTPD_START_SERVERS	The StartServers directive sets the number of child server processes created on startup	**8**
HTTPD_MAX_REQUEST_WORKERS	Number of simultaneous requests that will be handled by Apache	**256** but will be automatically lowered if memory is limited
PSGI_FILE	Specifies relative path to the PSGI application file. Use empty value to disable the PSGI autoconfiguration	Single *.psgi file in the top-level directory, if it exists
PSGI_URI_PATH	Specifies URI path that is handled by the PSGI application	**/**

To install additional Perl modules from the Comprehensive Perl Archive Network (CPAN), create a **cpanfile** in the root directory of your application sources. The file must conform to the **cpanfile** format as defined in Module-CPANFile CPAN distribution. For detailed information about the cpanfile format, refer to the cpanfile documentation.

To modify the **Apache httpd** behavior, drop the **.htaccess** file in the application sources tree where appropriate. For details about **.htaccess**, see the Apache HTTP Server Tutorial.

6.4. PYTHON

6.4.1. Description

The **rhscl/python-36-rhel7** image provides a Python 3.6 platform for building and running applications; **Node.js** with **npm** is preinstalled.

6.4.2. Access

To pull the **rhscl/python-36-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/python-36-rhel7
```

6.4.3. Configuration

To set environment variables, you can place them as a key-value pair into a **.sti/environment** file inside your source code repository.

Variable Name	Description
APP_SCRIPT	Used to run the application from a script file. This should be a path to a script file (defaults to **app.sh** unless set to null) that will be run to start the application.
APP_FILE	Used to run the application from a Python script. This should be a path to a Python file (defaults to **app.py**) that will be passed to the Python interpreter to start the application.
APP_MODULE	Used to run the application with Gunicorn, as documented here. This variable specifies a WSGI callable with the pattern **MODULE_NAME:VARIABLE_NAME**, where **MODULE_NAME** is the full dotted path of a module, and **VARIABLE_NAME** refers to a WSGI callable inside the specified module. Gunicorn will look for a WSGI callable named application if not specified. If **APP_MODULE** is not provided, the run script will look for a **wsgi.py** file in your project and use it if it exists. If using **setup.py** for installing the application, the **MODULE_NAME** part can be read from there. For an example, see **setup-test-app**.
APP_HOME	This variable can be used to specify a sub-directory in which the application to be run is contained. The directory pointed to by this variable needs to contain **wsgi.py** (for Gunicorn) or **manage.py** (for Django). If **APP_HOME** is not provided, the **assemble** and **run** scripts will use the application's root directory.
APP_CONFIG	Path to a valid Python file with a Gunicorn configuration file.
DISABLE_COLLECTSTATIC	Set this variable to a non-empty value to inhibit the execution of **manage.py collectstatic** during the build. This affects only Django projects.
DISABLE_MIGRATE	Set this variable to a non-empty value to inhibit the execution of **manage.py migrate** when the produced image is run. This affects only Django projects.

Variable Name	Description
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 requirements.txt.
UPGRADE_PIP_TO_LATEST	Set this variable to a non-empty value to have the **pip** program be upgraded to the most recent version before any Python packages are installed. If not set, it will use whatever the default version is included by the platform for the Python version being used.
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 2.

6.5. RUBY

6.5.1. Description

The **rhscl/ruby-25-rhel7** image provides a Ruby 2.5 platform for building and running applications; **Node.js** with **npm** is preinstalled.

6.5.2. Access

To pull the **rhscl/ruby-25-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/ruby-25-rhel7
```

6.5.3. Configuration

To set environment variables, you can place them as a key-value pair into a **.sti/environment** file inside your source code repository.

Variable Name	Description
RACK_ENV	This variable specifies the environment where the Ruby application will be deployed (unless overwritten) - **production**, **development**, **test**. Each level has different behaviors in terms of logging verbosity, error pages, Ruby gem installation, and other. Note that application assets will be compiled only if the **RACK_ENV** is set to **production**.
DISABLE_ASSET_COMPILATION	This variable indicates that the asset compilation process will be skipped. Because this only takes place when the application is run in the production environment, it should be used only when assets are already compiled.

Variable Name	Description
PUMA_MIN_THREADS, **PUMA_MAX_THREADS**	These variables indicate the minimum and maximum threads that will be available in Puma's thread pool.
PUMA_WORKERS	This variable indicates the number of worker processes that will be launched. See documentation on Puma's clustered mode.
RUBYGEM_MIRROR	Set this variable to use a custom RubyGems mirror URL to download required gem packages during the build process.

For S2I scripts to work, you need to include the **puma** or **rack** gem in the application's Gemfile.

CHAPTER 7. DAEMON IMAGES

7.1. APACHE HTTP SERVER

7.1.1. Description

The **rhscl/httpd-24-rhel7** image provides an Apache HTTP 2.4 Server. The image can be used as a base image for other applications based on Apache HTTP web server.

7.1.2. Access

To pull the **rhscl/httpd-24-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/httpd-24-rhel7
```

The **rhscl/httpd-24-rhel7** image supports using the S2I tool.

7.1.3. Configuration

The **Apache HTTP Server** container image supports the following configuration variable, which can be set by using the **`-e`** option with the **`docker run`** command:

Variable Name	Description
`HTTPD_LOG_TO_VOLUME`	By default, **`httpd`** logs into standard output, so the logs are accessible by using the **`docker logs`** command. When **`HTTPD_LOG_TO_VOLUME`** is set, **`httpd`** logs into **`/var/log/httpd24`**, which can be mounted to host system using the Docker volumes.

If you want to run the image and mount the log files into **`/wwwlogs`** on the host as a docker volume, execute the following command:

```
$ docker run -d -u 0 -e HTTPD_LOG_TO_VOLUME=1 --name httpd -v
/wwwlogs:/var/log/httpd24:Z rhscl/httpd-24-rhel7
```

You can also set the following mount points by passing the **`-v /host:/container`** option to the **`docker run`** command:

Volume Mount Point	Description
`/var/www`	Apache HTTP Server data directory
`/var/log/httpd24`	Apache HTTP Server log directory (available only when running as root)

When mouting a directory from the host into the container, ensure that the mounted directory has the appropriate permissions and that the owner and group of the directory matches the user UID or name which is running inside the container.

NOTE

The **rhscl/httpd-24-rhel7** container image now uses **1001** as the default UID to work correctly within the source-to-image strategy in OpenShift. Additionally, the container image listens on port **8080** by default. Previously, the **rhscl/httpd-24-rhel7** container image listened on port **80** by default and ran as UID **0**.

To run the **rhscl/httpd-24-rhel7** container image as UID **0**, specify the **-u 0** option of the **docker run** command:

```
docker run -u 0 rhscl/httpd-24-rhel7
```

7.2. NGINX

7.2.1. Description

The **rhscl/nginx-112-rhel7** image provides an nginx 1.12 server and a reverse proxy server; the image can be used as a base image for other applications based on nginx 1.12 web server.

7.2.2. Access

To pull the **rhscl/nginx-112-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/nginx-112-rhel7
```

7.2.3. Configuration

The **nginx** container images support the following configuration variable, which can be set by using the **-e** option with the **docker run** command:

Variable Name	Description
NGINX_LOG_TO_VOLUME	By default, nginx logs into standard output, so the logs are accessible by using the **docker logs** command. When **NGINX_LOG_TO_VOLUME** is set, nginx logs into **/var/log/nginx112**, which can be mounted to host system using the Docker volumes.

The **rhscl/nginx-112-rhel7** images supports using the S2I tool.

7.3. VARNISH CACHE

7.3.1. Description

The **rhscl/varnish-5-rhel7** image provides Varnish Cache 5.0, an HTTP reverse proxy.

7.3.2. Access

To pull the **rhscl/varnish-5-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/varnish-5-rhel7
```

7.3.3. Configuration

No further configuration is required.

The **rhscl/varnish-5-rhel7** image supports using the S2I tool. Note that the **`default.vcl`** configuration file in the directory accessed by S2I needs to be in the VCL format.

CHAPTER 8. DATABASE IMAGES

8.1. MARIADB

8.1.1. Description

The **rhscl/mariadb-102-rhel7** image provides a MariaDB 10.2 SQL database server.

8.1.2. Access

To pull the **rhscl/mariadb-102-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/mariadb-102-rhel7
```

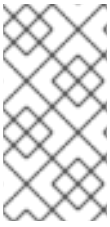
8.1.3. Usage and Configuration

The usage and configuration is the same as for the MySQL image. Note that the name of the daemon is **`mysqld`** and all environment variables have the same names as in MySQL.

8.1.4. Configuration

The image recognizes the following environment variables that you can set during initialization by passing **`-e VAR=VALUE`** to the **`docker run`** command:

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)



NOTE

The **root** user has no password set by default, only allowing local connections. You can set it by setting the **`MYSQL_ROOT_PASSWORD`** environment variable when initializing your container. This will allow you to login to the **root** account remotely. Local connections will still not require a password.

To disable remote root access, simply unset **`MYSQL_ROOT_PASSWORD`** and restart the container.

The following environment variables influence the MySQL configuration file and are all optional:

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 in case the native AIO is broken. See http://help.directadmin.com/item.php?id=529	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)
MYSQL_INNODB_LOG_FILE_SIZE	The size of each log file in a log group	8M (or 15% of available available)
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)

Variable name	Description	Default
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**	statement

You can also set the following mount point by passing the **-v /host:/container:Z** option to the **docker run** command:

Volume Mount Point	Description
/var/lib/mysql/data	MySQL data directory

NOTE

When mounting a directory from the host into the container, ensure that the mounted directory has the appropriate permissions and that the owner and group of the directory matches the user UID or name which is running inside the container, which is 27 by default.

See also How to Extend the rhscl/mariadb-101-rhel7 Container Image, which is applicable also to **rhscl/mariadb-102-rhel7**.

8.2. MONGODB

8.2.1. Description

The **rhscl/mongodb-36-rhel7** image provides a MongoDB 3.6 NoSQL database server; the **rhscl/mongodb-34-rhel7** image provides a MongoDB 3.4 NoSQL database server.

8.2.2. Access and Usage

To pull the **rhscl/mongodb-36-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/mongodb-36-rhel7
```

To pull the **rhscl/mongodb-34-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/mongodb-34-rhel7
```

To set only the mandatory environment variables and store the database in the **/home/user/database** directory on the host file system, execute the following command:

```
# docker run -d -e MONGODB_USER=<user> -e MONGODB_PASSWORD=<password> \
  -e MONGODB_DATABASE=<database> -e
MONGODB_ADMIN_PASSWORD=<admin_password> \
```

```
-v /home/user/database:/var/lib/mongodb/data <image_name>
```

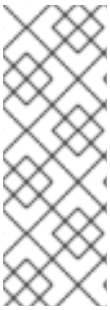
If you are initializing the database and it is the first time you are using the specified shared volume, two databases will be created with two users: the **admin** database with the **admin** user, and the **MONGODB_DATABASE** with the **MONGODB_USER**. After that, the MongoDB daemon will be started. If you are re-attaching the volume to another container, the creation of the database user and **admin** user will be skipped and only the MongoDB daemon will be started.

The **mongod** deamon writes its logs to the standard output. To examine the container image log, use the **docker logs <image_name>** command.

8.2.3. Configuration

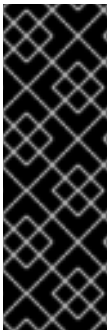
The image recognizes the following environment variables that you can set during initialization by passing **-e VAR=VALUE** to the **docker run** command:

Variable Name	Description
MONGODB_ADMIN_PASSWORD	Password for the **admin** user
MONGODB_USER	User name for the **MONGODB** account to be created
MONGODB_PASSWORD	Password for the user account
MONGODB_DATABASE	Database name



NOTE

The administrator user name is set to **admin** and you have to to specify the password by setting the **MONGODB_ADMIN_PASSWORD** environment variable. This process is done upon database initialization. The **admin** user has the **dbAdminAnyDatabase**, **userAdminAnyDatabase**, **readWriteAnyDatabase**, and **clusterAdmin** roles (see MongoDB reference for details). An unpriviledged user has only the **readWrite** role and needs to have a **MONGODB_USER** and **MONGODB_PASSWORD** environment variables set.



IMPORTANT

Since passwords are part of the image configuration, the only supported method to change passwords for an unpriviledged user and the **admin** user is by changing the environment variables **MONGODB_PASSWORD** and **MONGODB_ADMIN_PASSWORD**, respectively. Changing database passwords directly in MongoDB will cause a mismatch between the values stored in the variables and the actual passwords. Whenever a database container image starts, it will reset the passwords to the values stored in the environment variables.

The following environment variable influences the MongoDB configuration file and is optional:

Variable Name	Description	Default
MONGODB_QUIET	Runs MongoDB in a quiet mode that attempts to limit the amount of output	**true**

You can also set the following mount point by passing the **-v /host:/container** option to the **docker run** command:

Volume Mount Point	Description
/var/lib/mongodb/data	MongoDB data directory

NOTE

When mounting a directory from the host into the container, ensure that the mounted directory has the appropriate permissions and that the owner and group of the directory matches the user UID or name which is running inside the container.

8.2.4. Extending the Image

The MongoDB image can be extended using source-to-image.

For example, to build a customized MongoDB database image **my-mongodb-rhel7** with configuration in the **~/image-configuration/** directory, run:

```
$ s2i build ~/image-configuration/ rhscl/mongodb-36-rhel7 my-mongodb-rhel7
```

Change the image name to **rhscl/mongodb-34-rhel7** when using this image.

The directory passed to the S2I build should contain one or more of the following directories:

mongodb-cfg/	When running the **run-mongod** or **run-mongod-replication** commands, the contained **mongod.conf** file is used for **mongod** configuration. The **envsubst** command is run on the **mongod.conf** file to still allow customization of the image using environment variables. A custom configuration file does not affect the name of a replica set; it has to be set in the **MONGODB_REPLICA_NAME** environment variable. It is impossible to configure SSL using a custom configuration file.
mongodb-pre-init/	The contained shell scripts (***.sh**) are sourced before the **mongod** server is started.

mongodb-init/	The contained shell scripts (***.sh**) are sourced when the **mongod** server is started for the first time, that is, when the data directory is empty. The **run-mongod** command does not have an enabled authentication in this phase. The **run-mongod-replication** command does have an enabled authentication in this phase.
mongodb-start/	Similar to **mongodb-init/**, except these scripts are always sourced (after the **mongodb-init/** scripts, if they exist). These scripts are skipped if **run-mongod-replication** is run with an already initialized data directory.

The following variables can be used in the scripts provided to S2I:

mongo_common_args	Contains arguments for the **mongod** server. Note that changing this can break existing customization scripts, that is, the default scripts.
MEMBER_ID	Contains an ID of the container. It is defined only in scripts for replication (the **run-mongod-replication** command) and has a different value for each container image in a replica set cluster. Customization scripts are run by all container images in a replica set; **MEMBER_ID** can be used to write scripts which are run only by some containers.

During the S2I build, all provided files are copied into the **/opt/app-root/src/** directory in the new image. If some configuration files are present in the destination directory, files with the same name are overwritten. Also, only one file with the same name can be used for customization. User-provided files are preferred over the default files in the **/usr/share/container-scripts/mongodb/** directory, so it is possible to overwrite them.

The same configuration directory structure can be used to customize the image every time the image is started using the **docker run** command. The directory has to be mounted into **/opt/app-root/src/** in the image by using the **-v ./image-configuration/:/opt/app-root/src/** option. This overwrites the customization built into the image.

8.3. POSTGRESQL

8.3.1. Description

The **rhscl/postgresql-10-rhel7** image provides a PostgreSQL 10 SQL database server.

The **rhscl/postgresql-96-rhel7** image provides a PostgreSQL 9.6 SQL database server.

8.3.2. Access and Usage

To pull the **rhscl/postgresql-10-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/postgresql-10-rhel7
```

To pull the **rhscl/postgresql-96-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/postgresql-96-rhel7
```

To set only the mandatory environment variables and not store the database in a host directory, execute the following command:

```
# docker run -d --name postgresql_database -e POSTGRESQL_USER=<user> \
  -e POSTGRESQL_PASSWORD=<pass> -e POSTGRESQL_DATABASE=<db> \
  -p 5432:5432 <image_name>
```

This will create a container named **postgresql_database** running PostgreSQL with database db and user with credentials **user:pass**. Port **5432** will be exposed and mapped to the host. If you want your database to be persistent across container executions, also add a **-v /host/db/path:/var/lib/pgsql/data** argument. This will be the PostgreSQL database cluster directory.

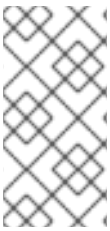
If the database cluster directory is not initialized, the entrypoint script will first run **initdb** and set up necessary database users and passwords. After the database is initialized, or if it was already present, **postgres** is executed and will run as **PID 1**. You can stop the detached container by running the **docker stop postgresql_database** command.

The the **postgres** daemon first writes its logs to the standard output. To examine the container image log, use the **docker logs <image_name>** command. Then the log output is redirected to the logging collector process and appears in the **pg_log/** directory.

8.3.3. Configuration

The image recognizes the following environment variables that you can set during initialization by passing **-e VAR=VALUE** to the **docker run** command:

Variable Name	Description
POSTGRESQL_USER	User name for PostgreSQL account to be created
POSTGRESQL_PASSWORD	Password for the user account
POSTGRESQL_DATABASE	Database name
POSTGRESQL_ADMIN_PASSWORD	Password for the postgres admin account (optional)



NOTE

The **postgres** administrator account has no password set by default, only allowing local connections. You can set it by setting the **POSTGRESQL_ADMIN_PASSWORD** environment variable when initializing your container. This will allow you to login to the **postgres** account remotely. Local connections will still not require a password.

IMPORTANT

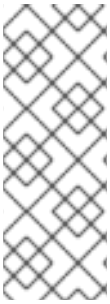
Since passwords are part of the image configuration, the only supported method to change passwords for the database user and postgres admin user is by changing the environment variables **POSTGRESQL_PASSWORD** and **POSTGRESQL_ADMIN_PASSWORD**, respectively. Changing database passwords through SQL statements 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 image starts, it will reset the passwords to the values stored in the environment variables.

The following options are related to migration:

Variable Name	Description	Default
POSTGRESQL_MIGRATION_REMOTE_HOST	Hostname/IP to migrate from	
POSTGRESQL_MIGRATION_ADMIN_PASSWORD	Password for the remote **postgres** admin user	
POSTGRESQL_MIGRATION_IGNORE_ERRORS	Optional: Ignore sql import errors	no

The following environment variables influence the PostgreSQL configuration file and are all optional:

Variable Name	Description	Default
POSTGRESQL_MAX_CONNECTIONS	The maximum number of client connections allowed. This also sets the maximum number of prepared transactions.	100
POSTGRESQL_MAX_PREPARED_TRANSACTIONS	Sets the maximum number of transactions that can be in the "prepared" state. If you are using prepared transactions, you will probably want this to be at least as large as max_connections	0
POSTGRESQL_SHARED_BUFFERS	Sets how much memory is dedicated to PostgreSQL to use for caching data	32M
POSTGRESQL_EFFECTIVE_CACHE_SIZE	Set to an estimate of how much memory is available for disk caching by the operating system and within the database itself	128M

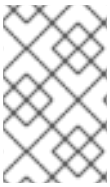


NOTE

When the PostgreSQL image is run with the **`--memory`** parameter set and if there are no values provided for **`POSTGRESQL_SHARED_BUFFERS`** and **`POSTGRESQL_EFFECTIVE_CACHE_SIZE`**, these values are automatically calculated based on the value provided in the **`--memory`** parameter. The values are calculated based on the upstream formulas and are set to 1/4 and 1/2 of the given memory, respectively.

You can also set the following mount point by passing the **`-v /host:/container`** option to the **`docker run`** command:

Volume Mount Point	Description
`/var/lib/pgsql/data`	PostgreSQL database cluster directory



NOTE

When mounting a directory from the host into the container, ensure that the mounted directory has the appropriate permissions and that the owner and group of the directory matches the user UID or name which is running inside the container.

Unless you use the **`-u`** option with the **`docker run`** command, processes in containers are usually run under UID **`26`**. To change the data directory permissions, use the following command:

```
$ setfacl -m u:26:-wx /your/data/dir
$ docker run <...> -v /your/data/dir:/var/lib/pgsql/data:Z <...>
```

8.3.4. Data Migration

PostgreSQL container images support migration of data from a remote PostgreSQL server. Use the following command and change the image name and add optional configuration variables when necessary:

```
$ docker run -d --name postgresql_database \
    -e POSTGRESQL_MIGRATION_REMOTE_HOST=172.17.0.2 \
    -e POSTGRESQL_MIGRATION_ADMIN_PASSWORD=remoteAdminP@ssword \
    [ OPTIONAL_CONFIGURATION_VARIABLES ]
    rhscl/postgresql-96-rhel7
```

The migration is done the dump and restore way (running **`pg_dumpall`** against a remote cluster and importing the dump locally by **`psql`**). Because the process is streamed (unix pipeline), there are no intermediate dump files created during this process to not waste additional storage space.

If some SQL commands fail during applying, the default behavior of the migration script is to fail as well to ensure the "all or nothing" result of a scripted, unattended migration. In most common cases, successful migration is expected (but not guaranteed), given you migrate from a previous version of PostgreSQL server container, which is created using the same principles - for example, migration from **`rhscl/postgresql-96-rhel7`** to **`rhscl/postgresql-10-rhel7`**. Migration from a different kind of PostgreSQL container image will likely fail.

If this "all or nothing" principle is inadequate for you, there is an optional **POSTGRESQL_MIGRATION_IGNORE_ERRORS** option which peforms "best effort" migration. However, some data might be lost and it is up to the user to review the standard error output and fix issues manually in the post-migration time.

NOTE

The container image provides migration help for users' convenience, but fully automatic migration is not guaranteed. Thus, before you start proceeding with the database migration, you will need to perform manual steps to get all your data migrated.

You might not use variables such as **POSTGRESQL_USER** in the migration scenario. All data (including information about databases, roles, or passwords) are copied from the old cluster. Ensure that you use the same optional configuration variables as you used for initialization of the old PostgreSQL container image. If some non-default configuration is done on a remote cluster, you might need to copy the configuration files manually, too.

WARNING

The IP communication between the old and the new PostgreSQL clusters is not encrypted by default, it is up to the user to configure SSL on a remote cluster or ensure security using different means.

8.3.5. Upgrading the Database

WARNING

Before you decide to perform the data directory upgrade, make sure you have backed up all your data. Note that you may need to manually roll back if the upgrade fails.

The PostreSQL image supports automatic upgrade of a data directory created by the PostgreSQL server version provided by the previous rhscl image, for example, the **rhscl/postgresql-10-rhel7** image supports upgrading from **rhscl/postgresql-96-rhel7**. The upgrade process is designed so that you should be able to just switch from image A to image B, and set the **$POSTGRESQL_UPGRADE** variable appropriately to explicitly request the database data transformation.

The upgrade process is internally implemented using the **pg_upgrade** binary, and for that purpose the container needs to contain two versions of PostgreSQL server (see the **pg_upgrade** man page for more information).

For the **pg_upgrade** process and the new server version, it is necessary to initialize a new data directory. This data directory is created automatically by the container tooling in the **/var/lib/pgsql/data/** directory, which is usually an external bind-mountpoint. The **pg_upgrade**

execution is then similar to the dump and restore approach. It starts both the old and the new PostgreSQL servers (within the container) and "dumps" the old data directory and, at the same time, it "restores" it into new data directory. This operation requires a lot of data files copying. Set the **$POSTGRESQL_UPGRADE** variable accordingly based on what type of upgrade you choose:

copy	The data files are copied from the old data directory to the new directory. This option has a low risk of data loss in case of an upgrade failure.
hardlink	Data files are hard-linked from the old to the new data directory, which brings performance optimization. However, the old directory becomes unusable, even in case of a failure.

NOTE

Make sure you have enough space for the copied data. Upgrade failure because of insufficient space might lead to a data loss.

8.3.6. Extending the Image

The PostgreSQL image can be extended using using source-to-image.

For example, to build a customized **new-postgresql** image with configuration in the **~/image-configuration/** directory, use the following command:

```
$ s2i build ~/image-configuration/ postgresql new-postgresql
```

The directory passed to the S2I build should contain one or more of the following directories:

postgresql-pre-start/	Source all ***.sh** files from this directory during an early start of the container. There is no PostgreSQL daemon running in the background.
postgresql-cfg/	Contained configuration files (***.conf**) will be included at the end of the image's **postgresql.conf** file.
postgresql-init/	Contained shell scripts (***.sh**) are sourced when the database is freshly initialized (after successful **initdb** run, which made the data directory non-empty). At the time of sourcing these scripts, the local PostgreSQL server is running. For re-deployments scenarios with persistent data directory, the scripts are not sourced (no-op).
postgresql-start/	Similar to **postgresql-init/**, except these scripts are always sourced (after the **postgresql-init/** scripts, if they exist).

During the S2I build, all provided files are copied into the **/opt/app-root/src/** directory in the new

image. Only one file with the same name can be used for customization, and user-provided files are preferred over default files in the **/usr/share/container-scripts/** directory, so it is possible to overwrite them.

CHAPTER 9. RED HAT DEVELOPER TOOLSET IMAGES

Red Hat Developer Toolset is a Red Hat offering for developers on the Red Hat Enterprise Linux platform. It provides a complete set of development and performance analysis tools that can be installed and used on multiple versions of Red Hat Enterprise Linux. Executables built with the Red Hat Developer Toolset toolchain can then also be deployed and run on multiple versions of Red Hat Enterprise Linux. For detailed compatibility information, see Red Hat Developer Toolset user Guide, section 1.3 Compatibility.

9.1. RUNNING RED HAT DEVELOPER TOOLSET TOOLS FROM PRE-BUILT CONTAINER IMAGES

To display general usage information for pre-built Red Hat Developer Toolset docker-formatted container images that you have already pulled to your local machine, run the following command as **root**:

```
# docker run image_name usage
```

To launch an interactive shell within a pre-built docker-formatted container image, run the following command as **root**:

```
# docker run -ti image_name /bin/bash -l
```

In both of the above commands, substitute the *image_name* parameter with the name of the container image you pulled to your local system and now want to use.

For example, to launch an interactive shell within the container image with selected toolchain components, run the following command as **root**:

```
# docker run -ti rhscl/devtoolset-7-toolchain-rhel7 /bin/bash -l
```

Example 9.1. Using GCC in the Pre-Built Red Hat Developer Toolset Toolchain Image

This example illustrates how to obtain and launch the pre-built docker-formatted container image with selected toolchain components of the Red Hat Developer Toolset and how to run the **gcc** compiler within that image.

1. Make sure you have a **Docker** environment set up properly on your system by following instructions at Getting Docker in RHEL 7.
2. Pull the pre-built toolchain Red Hat Developer Toolset container image from the official Red Hat Container Registry:

   ```
   # docker pull rhscl/devtoolset-7-toolchain-rhel7
   ```

3. To launch the container image with an interactive shell, issue the following command:

   ```
   # docker run -ti rhscl/devtoolset-7-toolchain-rhel7 /bin/bash -l
   ```

4. To launch the container as a regular (non-root) user, use the **sudo** command. To map a directory from the host system to the container file system, include the **-v** (or **--volume**) option in the **docker** command:

```
$ sudo docker run -v ~/Source:/src -ti rhscl/devtoolset-7-
toolchain-rhel7 /bin/bash -l
```

In the above command, the host's **~/Source/** directory is mounted as the **/src/** directory within the container.

5. Once you are in the container's interactive shell, you can run Red Hat Developer Toolset tools as expected. For example, to verify the version of the **gcc** compiler, run:

```
bash-4.2$ gcc -v
[...]
gcc version 6.3.1 20170216 (Red Hat 6.3.1-3) (GCC)
```

9.2. USING CONTAINER IMAGES BUILT FROM DOCKERFILES

Dockerfiles are available for selected Red Hat Developer Toolset components. Dockerfiles are text files that contain instructions for automated building of docker-formatted container images.

Red Hat Developer Toolset 7.0 for Red Hat Enterprise Linux 7 is distributed with the following Dockerfiles:

- devtoolset-7-toolchain
- devtoolset-7-perftools

9.2.1. Obtaining Dockerfiles

The Red Hat Developer Toolset Dockerfiles are provided by the package **devtoolset-7-dockerfiles**. The package contains individual Dockerfiles for building docker-formatted container images with individual components and a meta-Dockerfile for building a docker-formatted container image with all the components offered. To be able to use the Dockerfiles, install this package by executing:

```
# yum install devtoolset-7-dockerfiles
```

Use the RHSM channel **rhel-server-rhscl-7-rpms**. In order to enable it, follow the instructions at Getting Access to Red Hat Developer Toolset.

9.2.2. Building Container Images

Change to the directory where the Dockerfile is installed and run the following command as **root**:

```
# docker build -t image_name
```

Replace *image_name* with the desired name for the new image.

Example 9.2. Building a Container Image with a Red Hat Developer Toolset Component

To build a docker-formatted container image for deploying the **perftools** tools in a container, follow the instructions below:

1. Make sure you have a **Docker** environment set up properly on your system by following instructions at Getting Docker in RHEL 7.

2. Install the package containing the Red Hat Developer Toolset Dockerfiles:

```
# yum install devtoolset-7-dockerfiles
```

3. Determine where the Dockerfile for the required component is located:

```
# rpm -ql devtoolset-7-dockerfiles | grep "perftools-
docker/Dockerfile"
/opt/rh/devtoolset-7/root/usr/share/devtoolset-7-
dockerfiles/rhel7/devtoolset-7-perftools-docker/Dockerfile
```

4. Change to the directory where the required Dockerfile is installed:

```
# cd /opt/rh/devtoolset-7/root/usr/share/devtoolset-7-
dockerfiles/rhel7/devtoolset-7-perftools-docker/
```

5. Build the container image:

```
# docker build -t devtoolset-7-my-perftools .
```

Replace *devtoolset-7-my-perftools* with the name you wish to assign to your resulting container image.

9.2.3. Running Red Hat Developer Toolset Tools from Custom-Built Container Images

To display general usage information for images built from Red Hat Developer Toolset Dockerfiles (see Section 9.2.2, “Building Container Images”), run the following command as **root**:

```
docker run image_name container-usage
```

To launch an interactive shell within a docker-formatted container image you built, run the following command as **root**:

```
docker run -ti image_name /bin/bash -l
```

In both of the above commands, substitute the *image_name* parameter with the name of the container image you chose when building it.

Example 9.3. Using elfutils in a Custom-Built Red Hat Developer Toolset Image

This example illustrates how to launch a custom-built docker-formatted container image with the **elfutils** component and how to run the **eu-size** tool within that image.

1. To launch the container image with an interactive shell, issue the following command:

```
# docker run -ti devtoolset-7-my-perftools /bin/bash -l
```

2. To launch the container as a regular (non-root) user, use the **sudo** command. To map a directory from the host system to the container file system, include the **-v** (or **--volume**) option in the **docker** command:

```
$ sudo docker run -v ~/Source:/src -ti devtoolset-7-my-perftools
/bin/bash -l
```

In the above command, the host's **~/Source/** directory is mounted as the **/src/** directory within the container.

3. Once you are in the container's interactive shell, you can run Red Hat Developer Toolset tools as expected. For example, to verify the version of the **eu-size** tool, run:

```
bash-4.2$ eu-size -V
size (elfutils) 0.168
[...]
```

9.3. ADDITIONAL RESOURCES

For more information about Red Hat Developer Toolset, see the following online resources:

- Red Hat Developer Toolset 7.1 User Guide
- Red Hat Developer Toolset 7.1 Release Notes

9.4. RED HAT DEVELOPER TOOLSET TOOLCHAIN

9.4.1. Description

The Red Hat Developer Toolset Toolchain images provide the GNU Compiler Collection (GCC) and GNU Debugger (GDB).

The **rhscl/devtoolset-7-toolchain-rhel7** image provides content corresponding to the following packages:

Component	Version	Package
gcc	7.3.1	devtoolset-7-gcc
g++		devtoolset-7-gcc-c++
gfortran		devtoolset-7-gcc-fortran
gdb	8.0.1	devtoolset-7-gdb

Additionally, the **devtoolset-7-binutils** package is included as a dependency.

9.4.2. Access

To pull the **rhscl/devtoolset-7-toolchain-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/devtoolset-7-toolchain-
rhel7
```

-

9.5. RED HAT DEVELOPER TOOLSET PERFORMANCE TOOLS

9.5.1. Description

The Red Hat Developer Toolset Performance Tools images provide a number of profiling and performance measurement tools.

The **rhscl/devtoolset-7-perftools-rhel7** image provides the following components:

Component	Version	Package
dwz	0.12	devtoolset-7-dwz
Dyninst	9.3.2	devtoolset-7-dyninst
elfutils	0.170	devtoolset-7-elfutils
ltrace	0.7.91	devtoolset-7-ltrace
make	4.2.1	devtoolset-7-make
memstomp	0.1.5	devtoolset-7-memstomp
OProfile	1.2.0	devtoolset-7-oprofile
strace	4.17	devtoolset-7-strace
SystemTap	3.1	devtoolset-7-systemtap
Valgrind	3.13.0	devtoolset-7-valgrind

Additionally, the **devtoolset-7-gcc** and **devtoolset-7-binutils** packages are included as a dependency.

9.5.2. Access

To pull the **rhscl/devtoolset-7-perftools-rhel7** image, run the following command as **root**:

```
# docker pull registry.access.redhat.com/rhscl/devtoolset-7-perftools-
rhel7
```

9.5.3. Usage

Using the SystemTap Tool from Container Images

When using the **SystemTap** tool from a container image, additional configuration is required, and the container needs to be run with special command-line options.

The following three conditions need to be met:

1. The image needs to be run with super-user privileges. To do this, run the image using the following command:

```
~]$ docker run --ti --privileged --ipc=host --net=host --pid=host
devtoolset-6-my-perftools /bin/bash -l
```

   To use the pre-built **perftools** image, substitute the image name for **devtoolset-6-perftools-rhel7** in the above command.

2. The following kernel packages need to be installed in the container:

   - **kernel**
   - **kernel-devel**
   - **kernel-debuginfo**
     The version and release numbers of the above packages must match the version and release numbers of the kernel running on the host system. Run the following command to determine the version and release numbers of the hosts system's kernel:

```
~]$ uname -r
3.10.0-514.10.2.el7.x86_64
```

     Note that the **kernel-debuginfo** package is only available from the *Debug* channel. Enable the **`rhel-7-server-debug-rpms`** repository as described in TODO WHERE. For more information on how to get access to debuginfo packages, see https://access.redhat.com/site/solutions/9907.

     To install the required packages with the correct version, use the **yum** package manager and the output of the **`uname`** command. For example, to install the correct version of the **kernel** package, run the following command as **`root`**:

```
~]# yum install -y kernel-$(uname -r)
```

3. Save the container to a reusable image by executing the **`docker commit`** command. To save a custom-built **SystemTap** container:

```
~]$ docker commit devtoolset-7-systemtap-$(uname -r)
```

9.5.4. Known Issues

- **SystemTap** in **`devtoolset-6-perftools`** can not be used
  The **SystemTap** tool requires the **GCC** compiler to be present as **`/usr/bin/gcc`**, and support retpolines. However, this is not the case in the **`devtoolset-6-perftools`** container. As a consequence, **SystemTap** in this container fails to run any script, and produces the following error message:

```
*** CONFIG_RETPOLINE=y, but not supported by the compiler.
```

  As a result, **SystemTap** from **`devtoolset-6-perftools`** can not be used.

  To work around this issue, install **GCC** in the container before running **SystemTap** :

```
# yum install gcc -y
# stap ...
```

(BZ#1614809)

CHAPTER 10. COMPILER TOOLSET IMAGES

The following Red Hat Developer Tools container images are available as a Technology Preview:

- devtools/llvm-toolset-7-rhel7
- devtools/rust-toolset-7-rhel7
- devtools/go-toolset-7-rhel7

CHAPTER 11. REVISION HISTORY

Version	Date	Change	Author
0.0-8	Aug 29 2018	Added a known issue related to SystemTap in devtoolset-6-perftools.	Lenka Špačková
0.0-7	May 10 2018	Extended MongoDB image documentation.	Lenka Špačková
0.0-6	May 03 2018	Release of Using Red Hat Software Collections 3.1 Container Images.	Lenka Špačková
0.0-5	Apr 04 2018	Release of Using Red Hat Software Collections 3.1 Beta Container Images.	Lenka Špačková
0.0-3	Nov 29 2017	Added the Extending Existing Container Images section.	Lenka Špačková
0.0-2	Oct 24 2017	Release of Using Red Hat Software Collections 3.0 Container Images.	Lenka Špačková
0.0-1	Oct 03 2017	Release of Using Red Hat Software Collections 3.0 Beta Container Images.	Lenka Špačková