Red Hat Software Collections 3

Using Red Hat Software Collections Container Images

Basic Usage Instructions for Red Hat Software Collections 3.8 Container images

Last Updated: 2023-12-20

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

Basic Usage Instructions for Red Hat Software Collections 3.8 Container images

Lenka Špačková
lspackova@redhat.com

Olga Tikhomirova

Robert Krátký

Vladimír Slávik

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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. Red Hat Software Collections container images include application, web server, and database images. This document provides instructions for obtaining, configuring, and using container images that are distributed with Red Hat Software Collections.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 4

CHAPTER 1. RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES 5
1.1. RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES AS BUILDER IMAGES 6
1.2. EXTENDING EXISTING CONTAINER IMAGES 6

CHAPTER 2. BUILDING APPLICATION IMAGES USING RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES 7
2.1. BUILDING APPLICATION IMAGES USING RED HAT SOFTWARE COLLECTIONS IMAGES AS BASE IMAGES 7
2.2. BUILDING APPLICATION IMAGES FROM DOCKERFILES USING S2I SCRIPTS 8
2.3. BUILDING APPLICATION IMAGES USING SOURCE-TO-IMAGE IN OPENSHIFT 10
2.4. BUILDING APPLICATION IMAGES USING THE SOURCE-TO-IMAGE UTILITY 11

CHAPTER 3. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.8 13

CHAPTER 4. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.7 15

CHAPTER 5. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.6 17

CHAPTER 6. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.5 18

CHAPTER 7. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.4 19

CHAPTER 8. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.3 20

CHAPTER 9. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.2 21

CHAPTER 10. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.1 22

CHAPTER 11. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.0 24

CHAPTER 12. APPLICATION IMAGES 26
12.1. NODE.JS 26
12.1.1. Description 26
12.1.2. Access 26
12.1.3. Configuration 26
12.2. PHP 26
12.2.1. Description 26
12.2.2. Access 27
12.2.3. Configuration 27
12.2.4. Extending the Image 29
12.3. PERL 30
12.3.1. Description 30
12.3.2. Access 30
12.3.3. Configuration 30
12.4. PYTHON 31
12.4.1. Description 31
12.4.2. Access 31
12.4.3. Configuration 31
12.5. RUBY 33
12.5.1. Description 33
12.5.2. Access 33
12.5.3. Configuration 34

CHAPTER 13. DAEMON IMAGES **35**
13.1. APACHE HTTP SERVER 35
13.1.1. Description 35
13.1.2. Access 35
13.1.3. Configuration and Usage 35
13.2. NGINX 36
13.2.1. Description 36
13.2.2. Access 36
13.2.3. Configuration 36
13.3. VARNISH CACHE 37
13.3.1. Description 37
13.3.2. Access 37
13.3.3. Configuration 37

CHAPTER 14. DATABASE IMAGES **38**
14.1. MARIADB 38
14.1.1. Description 38
14.1.2. Access 38
14.1.3. Configuration and Usage 38
14.1.4. Extending the Image 41
14.2. MYSQL 41
14.2.1. Description 41
14.2.2. Access and Usage 41
14.2.3. Configuration 41
14.3. POSTGRESQL 44
14.3.1. Description 44
14.3.2. Access and Usage 44
14.3.3. Configuration 45
14.3.4. Data Migration 47
14.3.5. Upgrading the Database 48
14.3.6. Extending the Image 49
14.4. REDIS 49
14.4.1. Description 49
14.4.2. Access 50
14.4.3. Configuration and Usage 50

CHAPTER 15. RED HAT DEVELOPER TOOLSET IMAGES **51**
15.1. RUNNING RED HAT DEVELOPER TOOLSET TOOLS FROM PRE-BUILT CONTAINER IMAGES 51
15.2. RED HAT DEVELOPER TOOLSET TOOLCHAIN CONTAINER IMAGE 52
15.2.1. Description 52
15.2.2. Access 53
15.3. RED HAT DEVELOPER TOOLSET PERFORMANCE TOOLS CONTAINER IMAGE 53
15.3.1. Description 53
15.3.2. Access 53
15.3.3. Usage 54

CHAPTER 16. COMPILER TOOLSET IMAGES **55**

CHAPTER 17. REVISION HISTORY **56**

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES

Red Hat Software Collections container images are based on the corresponding collection and the rhel7 or the ubi7 base image. For more information about Universal Base Images, see Universal Base Images (UBI): Images, repositories, packages, and source code.

Red Hat Software Collections container images include application, daemon, and database images. Running Red Hat Software Collections container images is supported on:

- Red Hat Enterprise Linux 7 Server
- Red Hat Enterprise Linux 7 Atomic Host
- Red Hat Enterprise Linux 8

For information about components available as Software Collections for Red Hat Enterprise Linux 7, see the Red Hat Software Collections and Red Hat Developer Toolset documentation.

Red Hat Software Collections container images are detailed in the tables:

- Chapter 3, *Container Images Based on Red Hat Software Collections 3.8*
- Chapter 4, *Container Images Based on Red Hat Software Collections 3.7*
- Chapter 5, *Container Images Based on Red Hat Software Collections 3.6*
- Chapter 6, *Container Images Based on Red Hat Software Collections 3.5*
- Chapter 7, *Container Images Based on Red Hat Software Collections 3.4*
- Chapter 8, *Container Images Based on Red Hat Software Collections 3.3*
- Chapter 9, *Container Images Based on Red Hat Software Collections 3.2*
- Chapter 10, *Container Images Based on Red Hat Software Collections 3.1*
- Chapter 11, *Container Images Based on Red Hat Software Collections 3.0*

You can also search for available container images in the Red Hat Ecosystem Catalog .

IMPORTANT

Only the latest version of each container image provided by Red Hat is supported.

NOTE

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 the podman man page.

Additional Resources

- Getting Started with Containers

- Managing Containers
- Core Concepts of the OpenShift Enterprise Architecture
- README files of the Red Hat Software Collections container images in the **/help.1** file inside the image, or in the upstream GitHub repository.

1.1. RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES AS BUILDER IMAGES

You can use Red Hat Software Collections container images as builder images to build, deploy, and run your applications. To support common use cases, the following Source-to-Image (S2I) scripts are included in the builder images:

- The **/usr/libexec/s2i/assemble** script inside the image is run to produce a new image with the application artifacts. The script takes sources of a given application and places them into appropriate directories inside the image. If the application source includes definition of the dependent components (for example, **requirements.txt** that lists components from **PyPi** in case of Python projects), the components are installed into the image.
- The **/usr/libexec/s2i/run** script is set as the default command in the resulting container image (the new image with the application artifacts).

You can run the resulting application images using **podman**. For instructions, see Working with containers. In Red Hat Enterprise Linux 7, you can still use the **docker** command instead of **podman** with the same command-line syntax.

1.2. EXTENDING EXISTING CONTAINER IMAGES

To extend a functionality of a container image provided by Red Hat, you have the following options:

- Set environment variables. See documentation for the respective container image.
- Use OpenShift secrets.
- Build your custom application images. For instructions, see Chapter 2, *Building Application Images Using Red Hat Software Collections Container Images*.
- Use the Source-to-Image build strategy in OpenShift, which enables you to add your own configuration files, for daemon images that support this feature. Follow documentation for the respective container image.
- In case of other daemon or database images, 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 section called *Build an application using a Dockerfile* in the documentation for the respective container image or the example described in the Knowledgebase article How to Extend the rhscl/mariadb-101-rhel7 Container Image.

CHAPTER 2. BUILDING APPLICATION IMAGES USING RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES

You have several options how to build your application images using Red Hat Software Collections container images:

- Use container images provided by Red Hat as base images
- Use a Dockerfile with S2I scripts
- Use **Source-to-Image** in OpenShift
- Use the **source-to-image** utility

2.1. BUILDING APPLICATION IMAGES USING RED HAT SOFTWARE COLLECTIONS IMAGES AS BASE IMAGES

To use container images provided by Red Hat as base images:

1. Create a Dockerfile for your application image and ensure it contains the following line:

```
FROM registry.redhat.io/rhscl_image_name
```

2. Add your application code in the **src**/ directory to the image by putting the following line into the Dockerfile:

```
ADD src /opt/app-root/src
```

3. Build your application image using **podman**:

```
# podman build -t application_image_name .
```

4. Run your application image using **podman**. For example, to launch an interactive shell within your application image, run:

```
# podman run -ti application_image_name /bin/bash -l
```

Example 2.1. A Django application built from a Dockerfile using the rhscl/python-38-rhel7 base image

This example shows a Dockerfile that you can use for creating a simple Django application from the **rhscl/python-38-rhel7** container image.

```
# Set base image
FROM registry.redhat.io/rhscl/python-38-rhel7

# Add application sources
ADD --chown=1001:0 app-src .

# Install the dependencies
RUN pip install -U "pip>=19.3.1" && \
    pip install -r requirements.txt && \
```

```
    python manage.py collectstatic --noinput && \
    python manage.py migrate

# Run the application
CMD python manage.py runserver 0.0.0.0:8080
```

Additional Resources

- Building an image from a Dockerfile
- Dockerfile reference document

2.2. BUILDING APPLICATION IMAGES FROM DOCKERFILES USING S2I SCRIPTS

You can use Red Hat Software Collections container images as builder images and build your application images from Dockerfile using the **assemble** and **run** S2I scripts included in the builder images. For more information about the **assemble** and **run** S2I scripts, see Section 1.1, "Red Hat Software Collections Container Images as Builder Images".

To create an application image from a Dockerfile using S2I scripts, follow these steps:

1. Log in to the container registry:

```
# podman login registry.redhat.io
```

2. Pull a builder image:

```
# podman pull registry.redhat.io/rhscl_image_name
```

3. Prepare an application code.

4. Create a custom Dockerfile for your application image and ensure you:

   a. Define the builder image with this line:

```
FROM registry.redhat.io/rhscl_image_name
```

   b. Put the application source in the **src/** directory into the container and ensure that the default container user has sufficient permissions to access the source:

```
ADD --chown=1001:0 src /tmp/src
```

   c. Install dependencies using the **/usr/libexec/s2i/assemble** script:

```
RUN /usr/libexec/s2i/assemble
```

   d. Set the default command in the resulting image using the **/usr/libexec/s2i/run** script:

```
CMD /usr/libexec/s2i/run
```

5. Build your application image using podman:

```
# podman build -t application_image_name .
```

6. Run your application image using podman. For example, to launch an interactive shell within your application image, run:

```
# podman run -ti application_image_name /bin/bash -l
```

Example 2.2. Creating a Python 3.8 application image from a Dockerfile using S2I scripts

This example shows how to build and run a Python 3.8 application from a Dockerfile with S2I scripts provided by the builder image.

1. Log in to the container registry:

```
# podman login registry.redhat.io
```

2. Pull a builder image:

```
# podman pull registry.redhat.io/rhscl/python-38-rhel7
```

3. Pull an application code available at https://github.com/sclorg/django-ex.git:

```
$ git clone https://github.com/sclorg/django-ex.git app-src
```

Alternatively, use examples available at https://github.com/sclorg/s2i-python-container/tree/master/examples.

4. Create a Dockerfile with this content:

```
FROM registry.redhat.io/rhscl/python-38-rhel7

# Add application sources to a directory that the assemble script expects them
# and set permissions so that the container runs without root access
USER 0
ADD app-src /tmp/src
RUN chown -R 1001:0 /tmp/src
USER 1001

# Install the dependencies
RUN /usr/libexec/s2i/assemble

# Set the default command for the resulting image
CMD /usr/libexec/s2i/run
```

5. Build a new image from a Dockerfile prepared in the previous step:

```
# podman build -t python-app .
```

6. Run the resulting image with your Python application:

```
# podman run -d python-app
```

Additional Resources

- Building an image from a Dockerfile
- Dockerfile reference document
- The *Environment variables for Source-to-Image* section in the respective builder image README file, which is located in the **/help.1** file inside the image, or in the upstream GitHub repository.
- The environment variables are also documented in the detailed description of the image in the Red Hat Ecosystem Catalog .

2.3. BUILDING APPLICATION IMAGES USING SOURCE-TO-IMAGE IN OPENSHIFT

Source-to-Image (S2I) in OpenShift is a framework which enables you to write images that take application source code as an input, use a builder Red Hat Software Collections container image, and produce a new image that runs the assembled application as an output.

To create an application using S2I in OpenShift:

1. Build an application using an image available through OpenShift:

```
$ oc new-app openshift_image_name~path_to_application_source_code
```

   For example, to build a Python 3.8 application using the supported image available through the **python:3.8** imagestream tag in OpenShift, run:

```
$ oc new-app python:3.8~https://github.com/sclorg/django-ex.git
```

2. List available pods (instances):

```
$ oc get pods
```

3. Execute a selected pod on localhost:

```
$ oc exec pod -- curl 127.0.0.1:8080
```

Additional Resources

- OpenShift Container Platform documentation
- S2I Requirements
- source-to-image README file on GitHub
- The Environment variables for Source-to-Image section in the respective builder image README file.

2.4. BUILDING APPLICATION IMAGES USING THE SOURCE-TO-IMAGE UTILITY

The Red Hat Software Collections offering provides the **source-to-image** utility, which you can use without OpenShift on Red Hat Enterprise Linux 7 Server.

NOTE

The **source-to-image** utility is available only for Red Hat Enterprise Linux 7 and works only with images pulled by **docker**. You cannot use **podman** with the **source-to-image** utility.

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

- Source code of your application, written in a programming language or framework.
- A builder image, which is a Red Hat Software Collections container image that supports building images using the **source-to-image** utility.
- S2I scripts that are part of the builder image. For more information about these scripts, see Section 1.1, “Red Hat Software Collections Container Images as Builder Images” .

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

To use the **source-to-image** utility on your system:

1. Subscribe to Red Hat Software Collections. For instructions, see Getting Access to Red Hat Software Collections.
2. Enable the Red Hat Software Collections Server repository, which provides the **source-to-image** package, and the Red Hat Enterprise Linux 7 Server repository, which includes the **docker** package, required by **source-to-image**:

```
# subscription-manager repos --enable rhel-server-rhscl-7-rpms --enable rhel-7-server-extras-rpms
```

3. Install the **source-to-image** package:

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

4. Log in to the container registry:

```
# docker login registry.redhat.io
```

Pull a builder image:

```
# docker pull registry.redhat.io/rhscl_image_name
```

Build an application image from the application source code:

```
# s2i build path_to_application_source_code_repository --context-dir=source_code_context_directory application_image_name
```

5. Run the resulting image using **docker**.

Example 2.3. Building a Python 3.8 application from a Git repository using the source-to-image utility

This example shows how to build a test application available from a public Git repository using the **rhscl/python-38-rhel7** builder image and the **source-to-image** utility.

1. Log in to the container registry:

```
# docker login registry.redhat.io
```

2. Pull the **rhscl/python-38-rhel7** builder image:

```
# docker pull registry.redhat.io/rhscl/python-38-rhel7
```

3. Build the test application from the GitHub s2i-python repository, in the **3.8/test/setup-test-app/** directory:

```
# s2i build https://github.com/sclorg/s2i-python-container.git --context-dir=3.8/test/setup-test-app/ registry.redhat.io/rhscl/python-38-rhel7 python-38-rhel7-app
```

This produces a new application image, **python-38-rhel7-app**.

4. Run the resulting **python-38-rhel7-app** image:

```
# docker run -d -p 8080:8080 --name example-app python-38-rhel7-app
```

5. Fetch the resulting example document from http://localhost:8080/:

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

6. Stop the container:

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

Additional Resources

- S2I Requirements
- source-to-image README file on GitHub
- The *Environment variables for Source-to-Image section* in the respective builder image README file, which is located in the **/help.1** file inside the image, or in the upstream GitHub repository.

CHAPTER 3. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.8

Component	Description	Supported architectures
Daemon Images		
rhscl/nginx-120-rhel7	nginx 1.20 server and a reverse proxy server	x86_64, s390x, ppc64le
Database Images		
rhscl/redis-6-rhel7	Redis 6 key-value store	x86_64, s390x, ppc64le
Red Hat Developer Toolset Images		
rhscl/devtoolset-12-toolchain-rhel7 (available since November 2022)	Red Hat Developer Toolset toolchain	x86_64, s390x, ppc64le
rhscl/devtoolset-12-perftools-rhel7 (available since November 2022)	Red Hat Developer Toolset perftools	x86_64, s390x, ppc64le
rhscl/devtoolset-11-toolchain-rhel7	Red Hat Developer Toolset toolchain (EOL)	x86_64, s390x, ppc64le
rhscl/devtoolset-11-perftools-rhel7	Red Hat Developer Toolset perftools (EOL)	x86_64, s390x, ppc64le

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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 3.8 , see the Red Hat Software Collections 3.8 Release Notes .

For more information about the Red Hat Developer Toolset 11 components, see the Red Hat Developer Toolset 11 User Guide.

For information about the Red Hat Developer Toolset 12 components, see the Red Hat Developer Toolset 12 User Guide.

EOL images are no longer supported.

CHAPTER 4. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.7

Component	Description	Supported architectures
Application Images		
rhscl/ruby-30-rhel7	Ruby 3.0 platform for building and running applications	x86_64, s390x, ppc64le
rhscl/ruby-27-rhel7	Ruby 2.7 platform for building and running applications (EOL)	x86_64, s390x, ppc64le
rhscl/ruby-26-rhel7	Ruby 2.6 platform for building and running applications (EOL)	x86_64, s390x, ppc64le
Database Images		
rhscl/mariadb-105-rhel7	MariaDB 10.5 SQL database server	x86_64, s390x, ppc64le
rhscl/postgresql-13-rhel7	PostgreSQL 13 SQL database server	x86_64, s390x, ppc64le
Red Hat Developer Toolset Images		
rhscl/devtoolset-10-toolchain-rhel7	Red Hat Developer Toolset toolchain (EOL)	x86_64, s390x, ppc64le
rhscl/devtoolset-10-perftools-rhel7	Red Hat Developer Toolset perftools (EOL)	x86_64, s390x, ppc64le

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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 3.7 , see the Red Hat Software Collections 3.7 Release Notes .

For more information about the Red Hat Developer Toolset 10 components, see the Red Hat Developer Toolset 10 User Guide.

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

EOL images are no longer supported.

CHAPTER 5. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.6

Component	Description	Supported architectures
Application Images		
rhscl/nodejs-14-rhel7	Node.js 14 platform for building and running applications	x86_64, s390x, ppc64le
rhscl/perl-530-rhel7	Perl 5.30 platform for building and running applications	x86_64, s390x, ppc64le
rhscl/php-73-rhel7	PHP 7.3 platform for building and running applications	x86_64, s390x, ppc64le
rhscl/ruby-25-rhel7	Ruby 2.5 platform for building and running applications (EOL)	x86_64
Daemon Images		
rhscl/httpd-24-rhel7	Apache HTTP 2.4 Server	x86_64, s390x, ppc64le
rhscl/nginx-118-rhel7	nginx 1.18 server and a reverse proxy server (EOL)	x86_64, s390x, ppc64le

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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 3.6, see the Red Hat Software Collections 3.6 Release Notes .

For more information about the Red Hat Developer Toolset 10 components, see the Red Hat Developer Toolset 10 User Guide.

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

EOL images are no longer supported.

CHAPTER 6. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.5

Component	Description	Supported architectures
Application Images		
rhscl/python-38-rhel7	Python 3.8 platform for building and running applications	x86_64, s390x, ppc64le
Daemon Images		
rhscl/varnish-6-rhel7	Varnish Cache 6.0 HTTP reverse proxy	x86_64, s390x, ppc64le
Red Hat Developer Toolset Red Hat Developer Toolset Images		
rhscl/devtoolset-9-toolchain-rhel7	Red Hat Developer Toolset toolchain (EOL)	x86_64, s390x, ppc64le
rhscl/devtoolset-9-perftools-rhel7	Red Hat Developer Toolset perftools (EOL)	x86_64, s390x, ppc64le

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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 3.5, see the Red Hat Software Collections 3.5 Release Notes.

For more information about the Red Hat Developer Toolset 9.1 components, see the Red Hat Developer Toolset 9 User Guide.

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

EOL images are no longer supported.

CHAPTER 7. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.4

Component	Description	Supported architectures
Application Images		
rhscl/nodejs-12-rhel7	Node.js 12 platform for building and running applications (EOL)	x86_64, s390x, ppc64le
Daemon Images		
rhscl/nginx-116-rhel7	nginx 1.16 server and a reverse proxy server (EOL)	x86_64, s390x, ppc64le
Database Images		
rhscl/postgresql-12-rhel7	PostgreSQL 12 SQL database server	x86_64, s390x, ppc64le

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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 3.4, see the Red Hat Software Collections 3.4 Release Notes .

For more information about the Red Hat Developer Toolset 9.0 components, see the Red Hat Developer Toolset 9 User Guide.

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

EOL images are no longer supported.

CHAPTER 8. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.3

Component	Description	Supported architectures
Database Images		
rhscl/mariadb-103-rhel7	MariaDB 10.3 SQL database server (EOL)	x86_64, s390x, ppc64le
rhscl/redis-5-rhel7	Redis 5 key-value store (EOL)	x86_64, s390x, ppc64le
Red Hat Developer Toolset Images		
rhscl/devtoolset-8-toolchain-rhel7	Red Hat Developer Toolset toolchain (EOL)	x86_64, s390x, ppc64le
rhscl/devtoolset-8-perftools-rhel7	Red Hat Developer Toolset perftools (EOL)	x86_64, s390x, ppc64le

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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 3.3, see the Red Hat Software Collections 3.3 Release Notes.

For more information about the Red Hat Developer Toolset 8.1 components, see the Red Hat Developer Toolset 8 User Guide.

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

EOL images are no longer supported.

CHAPTER 9. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.2

Component	Description	Supported architectures
Application Images		
rhscl/nodejs-10-rhel7	Node.js 10 platform for building and running applications (EOL)	x86_64, s390x, ppc64le
rhscl/php-72-rhel7	PHP 7.2 platform for building and running applications (EOL)	x86_64, s390x, ppc64le
Daemon Images		
rhscl/nginx-114-rhel7	nginx 1.14 server and a reverse proxy server (EOL)	x86_64, s390x, ppc64le
Database Images		
rhscl/mysql-80-rhel7	MySQL 8.0 SQL database server	x86_64, s390x, ppc64le

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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 3.2, see the Red Hat Software Collections 3.2 Release Notes .

For more information about the Red Hat Developer Toolset 8.0 components, see the Red Hat Developer Toolset 8 User Guide.

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

EOL images are no longer supported.

CHAPTER 10. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.1

Component	Description	Supported architectures
Application Images		
rhscl/php-70-rhel7	PHP 7.0 platform for building and running applications (EOL)	x86_64
rhscl/perl-526-rhel7	Perl 5.26 platform for building and running applications (EOL)	x86_64
Daemon Images		
rhscl/varnish-5-rhel7	Varnish Cache 5.0 HTTP reverse proxy (EOL)	x86_64, s390x, ppc64le
Database Images		
rhscl/mongodb-36-rhel7	MongoDB 3.6 NoSQL database server (EOL)	x86_64
rhscl/postgresql-10-rhel7	PostgreSQL 10 SQL database server	x86_64, s390x, ppc64le
Red Hat Developer Toolset Images		
rhscl/devtoolset-7-toolchain-rhel7	Red Hat Developer Toolset toolchain (EOL)	x86_64, s390x, ppc64le
rhscl/devtoolset-7-perftools-rhel7	Red Hat Developer Toolset perftools (EOL)	x86_64, s390x, ppc64le

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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 User Guide.

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

EOL images are no longer supported.

CHAPTER 11. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.0

Component	Description	Supported architectures
Application Images		
rhscl/nodejs-8-rhel7	Node.js 8 platform for building and running applications (EOL)	x86_64, s390x, ppc64le
rhscl/php-71-rhel7	PHP 7.1 platform for building and running applications (EOL)	x86_64
rhscl/python-36-rhel7	Python 3.6 platform for building and running applications (EOL)	x86_64, s390x, ppc64le
Daemon Images		
rhscl/nginx-112-rhel7	nginx 1.12 server and a reverse proxy server (EOL)	x86_64, s390x, ppc64le
Database Images		
rhscl/mariadb-102-rhel7	MariaDB 10.2 SQL database server (EOL)	x86_64
rhscl/mongodb-34-rhel7	MongoDB 3.4 NoSQL database server (EOL)	x86_64
rhscl/postgresql-96-rhel7	PostgreSQL 9.6 SQL database server (EOL)	x86_64

Legend:

- x86_64 - AMD64 and Intel 64 architectures
- s390x - 64-bit IBM Z
- ppc64le - IBM POWER, little endian

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.

EOL images are no longer supported.

CHAPTER 12. APPLICATION IMAGES

12.1. NODE.JS

12.1.1. Description

The **rhscl/nodejs-14-rhel7** image provides a Node.js 14 platform for building and running applications.

12.1.2. Access

To pull the **rhscl/nodejs-14-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/nodejs-14-rhel7
```

12.1.3. Configuration

To set environment variables, you can place them as a key-value pair into a **.s2i/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

12.2. PHP

12.2.1. Description

The **rhscl/php-73-rhel7** image provides a PHP 7.3 platform for building and running applications. **Node.js** with **npm** is preinstalled in the PHP images.

12.2.2. Access

To pull the **rhscl/php-73-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/php-73-rhel7
```

12.2.3. Configuration

To set environment variables, place them as a key-value pair into a **.s2i/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-php73/root/usr/share/pear**
PHP_MEMORY_LIMIT	Memory limit	128M
SESSION_NAME	Name of the session	**PHPSESSID**
SESSION_HANDLER	Method for saving sessions	**files**
SESSION_PATH	Location for session data files	**/tmp/sessions**
SESSION_COOKIE_DOMAIN	The domain for which the cookie is valid	

Variable Name	Description	Default
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)	/

Replace the version of the **rh-php7*** Software Collection when appropriate.

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 in megabytes	128
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
OPCACHE_MAX_FILES	The maximum number of keys (scripts) in the OPcache hash table. Only numbers between 200 and 1000000 are allowed.	4000

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, 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.	256 (this is automatically tuned by setting Cgroup limits for the container using this formula: **TOTAL_MEMORY** / **15MB**. 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**.
COMPOSER_INSTALLER	Overrides the default URL for downloading Composer of https://getcomposer.org/installer. Useful in disconnected environments.
COMPOSER_ARGS	Adds extra arguments to the **composer install** command line (for example, **--no-dev**).

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.

12.2.4. Extending the Image

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

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

```
$ s2i build ~/image-configuration/ rhscl/php-73-rhel7 my-php-rhel7
```

Make sure to change the source image version accordingly.

The structure of the application can be similar to this example:

Directory name	Description
./httpd-cfg	Can contain additional Apache configuration files (***.conf**)
./httpd-ssl	Can contain own SSL certificate (in the **certs/** subdirectory) and key (in the **private/** subdirectory)
./php-pre-start	Can contain shell scripts (***.sh**) that are sourced before **httpd** is started
./php-post-assemble	Can contain shell scripts (***.sh**) that are sourced at the end of **assemble** script
./	Application source code

12.3. PERL

12.3.1. Description

The **rhscl/perl-530-rhel7** image provides a Perl 5.30 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**.

These images also support deploying Perl Web Server Gateway Interface (PSGI) applications.

12.3.2. Access

To pull the **rhscl/perl-530-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/perl-530-rhel7
```

12.3.3. Configuration

To set environment variables, you can place them as a key-value pair into a **.s2i/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

Variable Name	Description	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 a relative path to the PSGI application file. Use an empty value to disable the PSGI auto-configuration	Single *.psgi file in the top-level directory, if it exists
PSGI_URI_PATH	Specifies a 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.

12.4. PYTHON

12.4.1. Description

The **rhscl/python-38-rhel7** image provides a Python 3.8 platform for building and running applications. **Node.js** with **npm** is preinstalled.

12.4.2. Access

To pull the **rhscl/python-38-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/python-38-rhel7
```

12.4.3. Configuration

To set environment variables, you can place them as a key-value pair into a **.s2i/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_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.
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_SETUP_PY_PROCESSING	Set this variable to a non-empty value to skip processing of the **setup.py** script if you use **-e .** in **requirements.txt** to trigger its processing or you don't want your application to be installed into a site-packages directory.

Variable Name	Description
ENABLE_PIPENV	Set this variable to use Pipenv, the higher-level Python packaging tool, to manage dependencies of the application. This should be used only if your project contains an appropriately formatted **Pipfile** and **Pipfile.lock**.
ENABLE_INIT_WRAPPER	Set this variable to a non-empty value to make use of an init wrapper. This is useful for servers that are not capable of reaping zombie processes, such as Django development server or Tornado. This option can be used together with the **APP_SCRIPT** or **APP_FILE** variables. It never applies to Gunicorn used through **APP_MODULE** as Gunicorn reaps zombie processes correctly.
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.

12.5. RUBY

12.5.1. Description

The **rhscl/ruby-30-rhel7** image provides a Ruby 3.0 platform for building and running applications and the **rhscl/ruby-27-rhel7** image provides a Ruby 2.7 platform.

Node.js with **npm** is preinstalled.

12.5.2. Access

To pull the **rhscl/ruby-30-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/ruby-30-rhel7
```

To pull the **rhscl/ruby-27-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/ruby-27-rhel7
```

12.5.3. Configuration

To set environment variables, you can place them as a key-value pair into a **.s2i/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 set to **true** 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.
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 13. DAEMON IMAGES

13.1. APACHE HTTP SERVER

13.1.1. 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.

13.1.2. Access

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

```
# podman pull registry.redhat.io/rhscl/httpd-24-rhel7
```

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

13.1.3. Configuration and Usage

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

Variable Name	Description
HTTPD_LOG_TO_VOLUME	By default, **httpd** logs into standard output, so the logs are accessible by using the **podman logs** command. When **HTTPD_LOG_TO_VOLUME** is set, **httpd** logs into **/var/log/httpd24**, which can be mounted to host system using the container volumes. This option is allowed allowed when the container is run as UID **0**.
HTTPD_MPM	This variable can be set to change the default Multi-Processing Module (MPM) from the package default MPM.

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

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

To run an image using the **event** MPM (rather than the default **prefork**), execute the following command:

```
$ podman run -d -e HTTPD_MPM=event --name httpd rhscl/httpd-24-rhel7
```

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

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

13.2. NGINX

13.2.1. Description

The **rhscl/nginx-120-rhel7** image provides an nginx 1.20 server and a reverse proxy server; the image can be used as a base image for other applications based on the nginx 1.20 web server, the **rhscl/nginx-118-rhel7** image provides nginx 1.18.

13.2.2. Access

To pull the **rhscl/nginx-120-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/nginx-120-rhel7
```

To pull the **rhscl/nginx-118-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/nginx-118-rhel7
```

13.2.3. Configuration

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

Variable Name	Description
NGINX_LOG_TO_VOLUME	By default, nginx logs into standard output, so the logs are accessible by using the **podman logs** command. When **NGINX_LOG_TO_VOLUME** is set, nginx logs into **/var/opt/rh/rh-nginx120/log/nginx/** or **/var/opt/rh/rh-nginx120/log/nginx/**, which can be mounted to host system using the container volumes.

The **rhscl/nginx-120-rhel7** and **rhscl/nginx-118-rhel7** images support using the S2I tool.

13.3. VARNISH CACHE

13.3.1. Description

The **rhscl/varnish-6-rhel7** image provides Varnish Cache 6.0, an HTTP reverse proxy.

13.3.2. Access

To pull the **rhscl/varnish-6-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/varnish-6-rhel7
```

13.3.3. Configuration

No further configuration is required.

The Red Hat Software Collections Varnish Cache images support 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 14. DATABASE IMAGES

14.1. MARIADB

14.1.1. Description

The **rhscl/mariadb-105-rhel7** image provides a MariaDB 10.5 SQL database server.

14.1.2. Access

To pull the **rhscl/mariadb-105-rhel7** image, run the following command as **root**:

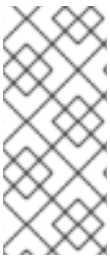
```
# podman pull registry.redhat.io/rhscl/mariadb-105-rhel7
```

14.1.3. Configuration and Usage

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.

The image recognizes the following environment variables that you can set during initialization by passing the **-e VAR=VALUE** option to the **podman 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)
MYSQL_CHARSET	Default character set (optional)
MYSQL_COLLATION	Default collation (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.

IMPORTANT

Because passwords are part of the image configuration, the only supported method to change passwords for an unpriviledged user (**MYSQL_USER**) and the **root** user is by changing the environment variables **MYSQL_PASSWORD** and **MYSQL_ROOT_PASSWORD**, respectively. Changing database passwords through SQL statements or any other way will cause a mismatch between the values stored in the variables and the actual passwords. Whenever a database container starts, it will reset the passwords to the values stored in the environment variables.

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)

Variable name	Description	Default
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)
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**

When the MariaDB image is run with the **--memory** parameter set, values of the following parameters will be automatically calculated based on the available memory unless the parameters are explicitly specified:

Variable name	Default memory percentage
MYSQL_KEY_BUFFER_SIZE	10%
MYSQL_READ_BUFFER_SIZE	5%
MYSQL_INNODB_BUFFER_POOL_SIZE	50%
MYSQL_INNODB_LOG_FILE_SIZE	15%
MYSQL_INNODB_LOG_BUFFER_SIZE	15%

You can also set the following mount point by passing the **-v /host:/container** option to the **podman 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.

14.1.4. Extending the Image

See How to Extend the rhscl/mariadb-101-rhel7 Container Image , which is applicable also to **rhscl/mariadb-105-rhel7**.

14.2. MYSQL

14.2.1. Description

The **rhscl/mysql-80-rhel7** image provides a MySQL 8.0 SQL database server.

14.2.2. Access and Usage

To pull the **rhscl/mysql-80-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/mysql-80-rhel7
```

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

```
# podman run -d --name mysql_database -e MYSQL_USER=<user> -e
MYSQL_PASSWORD=<pass> \
  -e MYSQL_DATABASE=<db> -p 3306:3306 rhscl/mysql-80-rhel7
```

This will create a container named **mysql_database** running MySQL with database db and user with credentials **user:pass**. Port **3306** 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/mysql/data** argument. The directory **/host/db/path** will be the MySQL data directory.

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

14.2.3. Configuration

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

Variable Name	Description
MYSQL_USER	User name for MySQL account to be created
MYSQL_PASSWORD	Password for the user account

Variable Name	Description
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.

IMPORTANT

Because passwords are part of the image configuration, the only supported method to change passwords for an unpriviledged user (**MYSQL_USER**) and the **root** user is by changing the environment variables **MYSQL_PASSWORD** and **MYSQL_ROOT_PASSWORD**, respectively. Changing database passwords through SQL statements or any other way will cause a mismatch between the values stored in the variables and the actual passwords. Whenever a database container starts, it will reset the passwords to the values stored in the environment variables.

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

Variable name	Description	Default
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 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)
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**
MYSQL_LOG_QUERIES_ENABLED	To enable query logging, set this variable to **1**	**0**

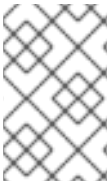
When the MySQL image is run with the **--memory** parameter set, values of the following parameters will be automatically calculated based on the available memory unless the parameters are explicitly specified:

Variable name	Default memory percentage
MYSQL_KEY_BUFFER_SIZE	10%

Variable name	Default memory percentage
MYSQL_READ_BUFFER_SIZE	5%
MYSQL_INNODB_BUFFER_POOL_SIZE	50%
MYSQL_INNODB_LOG_FILE_SIZE	15%
MYSQL_INNODB_LOG_BUFFER_SIZE	15%

You can also set the following mount point by passing the **-v /host:/container** option to the **podman 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.

14.3. POSTGRESQL

14.3.1. Description

The **rhscl/postgresql-13-rhel7** image provides a PostgreSQL 13 SQL database server; the **rhscl/postgresql-12-rhel7** image provides a PostgreSQL 12 server, and the **rhscl/postgresql-10-rhel7** image provides a PostgreSQL 10 server.

14.3.2. Access and Usage

To pull the **rhscl/postgresql-13-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/postgresql-13-rhel7
```

To pull the **rhscl/postgresql-12-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/postgresql-12-rhel7
```

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

```
# podman pull registry.redhat.io/rhscl/postgresql-10-rhel7
```

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

```
# podman 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 **podman stop postgresql_database** command.

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

14.3.3. Configuration

The image recognizes the following environment variables that you can set during initialization by passing **-e VAR=VALUE** to the **podman 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 **podman 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 **podman 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
$ podman run <...> -v /your/data/dir:/var/lib/pgsql/data:Z <...>
```

14.3.4. 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:

```
$ podman 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-12-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-10-rhel7** to **rhscl/postgresql-12-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.

14.3.5. 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.

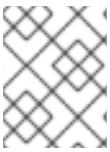
The PostreSQL image supports automatic upgrade of a data directory created by the PostgreSQL server version provided by the previous rhscl image, for example, the **rhscl/postgresql-13-rhel7** image supports upgrading from **rhscl/postgresql-12-rhel7**. The upgrade process is designed so that you should 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.

14.3.6. 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.

14.4. REDIS

14.4.1. Description

The **rhscl/redis-6-rhel7** image provides Redis 6, an advanced key-value store.

14.4.2. Access

To pull the **rhscl/redis-6-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/redis-6-rhel7
```

14.4.3. Configuration and Usage

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

```
# podman run -d --name redis_database -p 6379:6379 rhscl/redis-6-rhel7
```

This command creates a container named **redis_database**. Port **6379** is exposed and mapped to the host.

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

Variable Name	Description
REDIS_PASSWORD	Password for the server access

To set a password, run:

```
# podman run -d --name redis_database -e REDIS_PASSWORD=strongpassword rhscl/redis-6-rhel7
```

IMPORTANT

Use a very strong password because Redis is fast and thus can become a target of a brute-force attack.

To make your database persistent across container executions, add the **-v /host/db/path:/var/lib/redis/data:Z** option to the **podman run** command.

Volume Mount Point	Description
/var/lib/redis/data	Redis 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 that is running inside the container.

To examine the container image log, use the **podman logs <image_name>** command.

CHAPTER 15. 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 12 User Guide .

IMPORTANT

Only container images providing the latest version of Red Hat Developer Toolset are supported.

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

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

```
# podman run image_name usage
```

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

```
# podman 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**:

```
# podman run -ti rhscl/devtoolset-12-toolchain-rhel7 /bin/bash -l
```

Example 15.1. Using GCC in the Pre-Built Red Hat Developer Toolset Toolchain Image

This example illustrates how to obtain and launch the pre-built 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 container environment set up properly on your system by following instructions at Using podman to work with containers in the *Managing Containers* document.
2. Pull the pre-built toolchain Red Hat Developer Toolset container image from the official Red Hat Container Registry:

   ```
   # podman pull rhscl/devtoolset-12-toolchain-rhel7
   ```

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

   ```
   # podman run -ti rhscl/devtoolset-12-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 **podman** command:

```
$ sudo podman run -v ~/Source:/src -ti rhscl/devtoolset-12-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 12.2.1 20221121 (Red Hat 12.2.1-4) (GCC)
```

Additional Resources

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

- Red Hat Developer Toolset 12 User Guide
- Red Hat Developer Toolset 12.1 Release Notes
- Red Hat Developer Toolset 12.0 Release Notes

15.2. RED HAT DEVELOPER TOOLSET TOOLCHAIN CONTAINER IMAGE

15.2.1. Description

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

The **rhscl/devtoolset-12-toolchain-rhel7** image contains content corresponding to the following packages:

Component	Version	Package
gcc	12.2.1	devtoolset-12-gcc
g++		devtoolset-12-gcc-c++
gfortran		devtoolset-12-gcc-gfortran
gdb	11.2	devtoolset-12-gdb

Additionally, the **devtoolset-12-binutils** package is included as a dependency.

15.2.2. Access

To pull the **rhscl/devtoolset-12-toolchain-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/devtoolset-12-toolchain-rhel7
```

15.3. RED HAT DEVELOPER TOOLSET PERFORMANCE TOOLS CONTAINER IMAGE

15.3.1. Description

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

The **rhscl/devtoolset-12-perftools-rhel7** image includes the following components:

Component	Version	Package
dwz	0.14	devtoolset-12-dwz
Dyninst	12.1.0	devtoolset-12-dyninst
elfutils	0.187	devtoolset-12-elfutils
ltrace	0.7.91	devtoolset-12-ltrace
make	4.3	devtoolset-12-make
memstomp	0.1.5	devtoolset-12-memstomp
OProfile	1.4.0	devtoolset-12-oprofile
strace	5.18	devtoolset-12-strace
SystemTap	4.7	devtoolset-12-systemtap
Valgrind	3.19.0	devtoolset-12-valgrind

Additionally, the **devtoolset-12-gcc** and **devtoolset-12-binutils** packages are included as a dependency.

15.3.2. Access

To pull the **rhscl/devtoolset-12-perftools-rhel7** image, run the following command as **root**:

```
# podman pull registry.redhat.io/rhscl/devtoolset-12-perftools-rhel7
```

15.3.3. 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:

```
~]$ podman run --ti --privileged --ipc=host --net=host --pid=host devtoolset-12-my-perftools
/bin/bash -l
```

   To use the pre-built **perftools** image, substitute the image name for **devtoolset-12-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
3.10.0-1160.90.1.el7.x86_64
```

     Note that the **kernel-debuginfo** package is only available from the *Debug* repository. Enable the **rhel-7-server-debug-rpms** repository. For more information on how to get access to debuginfo packages, see How can I download or install debuginfo packages for RHEL systems?.

     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 **podman commit** command. To save a custom-built **SystemTap** container:

```
~]$ podman commit devtoolset-12-systemtap-$(uname -r)
```

CHAPTER 16. COMPILER TOOLSET IMAGES

Red Hat Developer Tools container images are available for the AMD64 and Intel 64, 64-bit IBM Z, and IBM POWER, little endian architectures for the following compiler toolsets:

- Clang and LLVM Toolset
- Rust Toolset
- Go Toolset

For details, see the Red Hat Developer Tools documentation .

CHAPTER 17. REVISION HISTORY

Version	Date	Change	Author
0.2-7	Dec 20 2023	Removed an outdated link.	Lenka Špačková
0.2-6	Jul 03 2023	The **rhscl/mariadb-103-rhel7** container image is EOL.	Lenka Špačková
0.2-5	May 23 2023	Update with the release of Red Hat Developer Toolset 12.1.	Lenka Špačková
0.2-4	Nov 22 2022	Update with the release of Red Hat Developer Toolset 12.0.	Lenka Špačková
0.2-3	Nov 15 2021	Release of Using Red Hat Software Collections 3.8 Container Images.	Lenka Špačková
0.2-2	Oct 11 2021	Release of Using Red Hat Software Collections 3.8 Beta Container Images.	Lenka Špačková
0.2-1	Jun 03 2021	Release of Using Red Hat Software Collections 3.7 Container Images.	Lenka Špačková
0.2-0	May 03 2021	Release of Using Red Hat Software Collections 3.7 Beta Container Images.	Lenka Špačková
0.1-9	Apr 06 2021	Improved supported architectures.	Lenka Špačková
0.1-8	Jan 13 2021	Improved introductory chapters and extended information about building application images.	Lenka Špačková
0.1-7	Dec 01 2020	Release of Using Red Hat Software Collections 3.6 Container Images.	Lenka Špačková
0.1-6	Oct 29 2020	Release of Using Red Hat Software Collections 3.6 Beta Container Images.	Lenka Špačková
0.1-5	May 26 2020	Release of Using Red Hat Software Collections 3.5 Container Images.	Lenka Špačková
0.1-4	Apr 21 2020	Release of Using Red Hat Software Collections 3.5 Beta Container Images.	Lenka Špačková
0.1-3	Dec 10 2019	Release of Using Red Hat Software Collections 3.4 Container Images.	Lenka Špačková
0.1-2	Nov 07 2019	Release of Using Red Hat Software Collections 3.4 Beta Container Images.	Lenka Špačková

Version	Date	Change	Author
0.1-1	Jun 11 2019	Release of Using Red Hat Software Collections 3.3 Container Images.	Lenka Špačková
0.1-0	Apr 16 2019	Release of Using Red Hat Software Collections 3.3 Beta Container Images.	Lenka Špačková
0.0-9	Nov 13 2018	Release of Using Red Hat Software Collections 3.2 Container Images.	Lenka Špačková
0.0-8	Oct 23 2018	Release of Using Red Hat Software Collections 3.2 Beta Container Images.	Lenka Špačková
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