Red Hat Software Collections 3

Using Red Hat Software Collections Container Images

Basic Usage Instructions for Red Hat Software Collections 3.6 Container images
Basic Usage Instructions for Red Hat Software Collections 3.6 Container images

Lenka Špačková
lspackova@redhat.com

Robert Krátký

Vladimír Slávik
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

**CHAPTER 1. RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES** ........................................ 4  
1.1. RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES AS BUILDER IMAGES 5  
1.2. EXTENDING EXISTING CONTAINER IMAGES 5  

**CHAPTER 2. BUILDING APPLICATION IMAGES USING RED HAT SOFTWARE COLLECTIONS CONTAINER IMAGES** ........................................ 6  
2.1. BUILDING APPLICATION IMAGES USING RED HAT SOFTWARE COLLECTIONS IMAGES AS BASE IMAGES 6  
2.2. BUILDING APPLICATION IMAGES FROM DOCKERFILES USING S2I SCRIPTS 7  
2.3. BUILDING APPLICATION IMAGES USING SOURCE-TO-IMAGE IN OPENShift 9  
2.4. BUILDING APPLICATION IMAGES USING THE SOURCE-TO-IMAGE UTILITY 10  

**CHAPTER 3. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.6** ............. 12  

**CHAPTER 4. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.5** ............. 14  

**CHAPTER 5. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.4** ............. 15  

**CHAPTER 6. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.3** ............. 16  

**CHAPTER 7. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.2** ............. 17  

**CHAPTER 8. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.1** ............. 18  

**CHAPTER 9. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.0** ............. 20  

**CHAPTER 10. APPLICATION IMAGES** ............................................................................. 22  
10.1. NODE.JS 22  
10.1.1. Description 22  
10.1.2. Access 22  
10.1.3. Configuration 22  
10.2. PHP 23  
10.2.1. Description 23  
10.2.2. Access 23  
10.2.3. Configuration 23  
10.2.4. Extending the Image 25  
10.3. PERL 26  
10.3.1. Description 26  
10.3.2. Access 26  
10.3.3. Configuration 26  
10.4. PYTHON 27  
10.4.1. Description 27  
10.4.2. Access 27  
10.4.3. Configuration 28  
10.5. RUBY 29  
10.5.1. Description 29  
10.5.2. Access 29  
10.5.3. Configuration 30  

**CHAPTER 11. DAEMON IMAGES** .................................................................................. 31  
11.1. APACHE HTTP SERVER 31  
11.1.1. Description 31  
11.1.2. Access 31  
11.1.3. Configuration and Usage 31
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 3.6
- Chapter 4, Container Images Based on Red Hat Software Collections 3.5
- Chapter 5, Container Images Based on Red Hat Software Collections 3.4
- Chapter 6, Container Images Based on Red Hat Software Collections 3.3
- Chapter 7, Container Images Based on Red Hat Software Collections 3.2
- Chapter 8, Container Images Based on Red Hat Software Collections 3.1
- Chapter 9, 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
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, for example, nginx.

- 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/rhsc1_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/rhsc1/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 && 
   ...
# 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/rhsc1_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/rhsc1_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:

   ```bash
   $ 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:

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

2. List available pods (instances):

   ```bash
   $ oc get pods
   ```

3. Execute a selected pod on localhost:

   ```bash
   $ 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 3.6

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Supported architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/nodejs-14-rhel7</td>
<td>Node.js 14 platform for building and running applications</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/perl-530-rhel7</td>
<td>Perl 5.30 platform for building and running applications</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/php-73-rhel7</td>
<td>PHP 7.3 platform for building and running applications</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/ruby-25-rhel7</td>
<td>Ruby 2.5 platform for building and running applications</td>
<td>x86_64</td>
</tr>
<tr>
<td><strong>Daemon Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/httpd-24-rhel7</td>
<td>Apache HTTP 2.4 Server</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/nginx-118-rhel7</td>
<td>nginx 1.18 server and a reverse proxy server</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td><strong>Red Hat Developer Toolset Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/devtoolset-10-toolchain-rhel7</td>
<td>Red Hat Developer Toolset toolchain</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/devtoolset-10-perftools-rhel7</td>
<td>Red Hat Developer Toolset perftools</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
</tbody>
</table>

**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](https://access.redhat.com/documentation/en-us/red_hat_software_collections/3.6/html/container_images_based_on_red_hat_software_collections#-3_release_notes).

12
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.
# CHAPTER 4. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.5

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Supported architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/python-38-rhel7</td>
<td>Python 3.8 platform for building and running applications</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/ruby-27-rhel7</td>
<td>Ruby 2.7 platform for building and running applications</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td><strong>Daemon Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/varnish-6-rhel7</td>
<td>Varnish Cache 6.0 HTTP reverse proxy</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td><strong>Red Hat Developer Toolset Red Hat Developer Toolset Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/devtoolset-9-toolchain-rhel7</td>
<td>Red Hat Developer Toolset toolchain (EOL)</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/devtoolset-9-perftools-rhel7</td>
<td>Red Hat Developer Toolset perftools (EOL)</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
</tbody>
</table>

**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 5. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.4

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Supported architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/nodejs-12-rhel7</td>
<td>Node.js 12 platform for building and running applications</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daemon Images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/nginx-116-rhel7</td>
<td>nginx 1.16 server and a reverse proxy server</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database Images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/postgresql-12-rhel7</td>
<td>PostgreSQL 12 SQL database server</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
# CHAPTER 6. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.3

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Supported architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/ruby-26-rhel7</td>
<td>Ruby 2.6 platform for building and running applications</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td><strong>Daemon Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Database Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/mariadb-103-rhel7</td>
<td>MariaDB 10.3 SQL database server</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/redis-5-rhel7</td>
<td>Redis 5 key-value store</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td><strong>Red Hat Developer Toolset Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/devtoolset-8-toolchain-rhel7</td>
<td>Red Hat Developer Toolset toolchain (EOL)</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/devtoolset-8-perftools-rhel7</td>
<td>Red Hat Developer Toolset perftools (EOL)</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
</tbody>
</table>

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 7. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.2

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

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 8. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.1

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Supported architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/php-70-rhel7</td>
<td>PHP 7.0 platform for building and running applications (EOL)</td>
<td>x86_64</td>
</tr>
<tr>
<td>rhscl/perl-526-rhel7</td>
<td>Perl 5.26 platform for building and running applications</td>
<td>x86_64</td>
</tr>
<tr>
<td><strong>Daemon Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/varnish-5-rhel7</td>
<td>Varnish Cache 5.0 HTTP reverse proxy</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td><strong>Database Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/mongodb-36-rhel7</td>
<td>MongoDB 3.6 NoSQL database server</td>
<td>x86_64</td>
</tr>
<tr>
<td>rhscl/postgresql-10-rhel7</td>
<td>PostgreSQL 10 SQL database server</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td><strong>Red Hat Developer Toolset 10.0 Images</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhscl/devtoolset-7-toolchain-rhel7</td>
<td>Red Hat Developer Toolset toolchain (EOL)</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
<tr>
<td>rhscl/devtoolset-7-perftools-rhel7</td>
<td>Red Hat Developer Toolset perftools (EOL)</td>
<td>x86_64, s390x, ppc64le</td>
</tr>
</tbody>
</table>

**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 9. CONTAINER IMAGES BASED ON RED HAT SOFTWARE COLLECTIONS 3.0

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

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 10. APPLICATION IMAGES

10.1. NODE.JS

10.1.1. Description

The rhscl/nodejs-14-rhel7 image provides a Node.js 14 platform for building and running applications; the rhscl/nodejs-12-rhel7 image provides a Node.js 12 platform, and the rhscl/nodejs-10-rhel7 a Node.js 10 platform.

10.1.2. Access

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

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

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

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

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

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

10.1.3. Configuration

To set environment variables, you can place them as a key–value pair into a .s2i/environment file inside your source code repository.

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

10.2. PHP

10.2.1. 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.

10.2.2. Access

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

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

10.2.3. 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:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_REPORTING</td>
<td>Informs PHP of which errors, warnings and notices you would like it to take action for</td>
<td>E_ALL &amp; ~E_NOTICE</td>
</tr>
<tr>
<td>DISPLAY_ERRORS</td>
<td>Controls whether or not and where PHP will output errors, notices and warnings</td>
<td>ON</td>
</tr>
<tr>
<td>DISPLAY_STARTUP_ERRORS</td>
<td>Cause display errors which occur during PHP’s startup sequence to be handled separately from display errors</td>
<td>OFF</td>
</tr>
<tr>
<td>TRACK_ERRORS</td>
<td>Store the last error/warning message in $php_errormsg (boolean)</td>
<td>OFF</td>
</tr>
<tr>
<td>HTML_ERRORS</td>
<td>Link errors to documentation related to the error</td>
<td>ON</td>
</tr>
<tr>
<td>INCLUDE_PATH</td>
<td>Path for PHP source files</td>
<td>..:/opt/app-root/src:/opt/rh/rh-php73/root/usr/share/pear</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>PHP_MEMORY_LIMIT</strong></td>
<td>Memory limit</td>
<td>128M</td>
</tr>
<tr>
<td><strong>SESSION_NAME</strong></td>
<td>Name of the session</td>
<td>PHPSESSID</td>
</tr>
<tr>
<td><strong>SESSION_HANDLER</strong></td>
<td>Method for saving sessions</td>
<td>files</td>
</tr>
<tr>
<td><strong>SESSION_PATH</strong></td>
<td>Location for session data files</td>
<td>/tmp/sessions</td>
</tr>
<tr>
<td><strong>SESSION_COOKIE_DOMAIN</strong></td>
<td>The domain for which the cookie is valid</td>
<td></td>
</tr>
<tr>
<td><strong>SESSION_COOKIE_HTTPONLY</strong></td>
<td>Whether or not to add the httpOnly flag to the cookie</td>
<td>0</td>
</tr>
<tr>
<td><strong>SESSION_COOKIE_SECURE</strong></td>
<td>Specifies whether cookies should only be sent over secure connections</td>
<td>OFF</td>
</tr>
<tr>
<td><strong>SHORT_OPEN_TAG</strong></td>
<td>Determines whether or not PHP will recognize code between &lt;? and ?&gt; tags</td>
<td>OFF</td>
</tr>
<tr>
<td><strong>DOCUMENTROOT</strong></td>
<td>Path that defines the DocumentRoot for your application (ie. /public)</td>
<td>/</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OPCACHE_MEMORY_CONSUMPtion</strong></td>
<td>The OPcache shared memory storage size in megabytes</td>
<td>128</td>
</tr>
<tr>
<td><strong>OPCACHE_REVALIDATE_FREQ</strong></td>
<td>How often to check script timestamps for updates, in seconds. 0 will result in OPcache checking for updates on every request.</td>
<td>2</td>
</tr>
<tr>
<td><strong>OPCACHE_MAX_FILES</strong></td>
<td>The maximum number of keys (scripts) in the OPcache hash table. Only numbers between 200 and 1000000 are allowed.</td>
<td>4000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets the path to the <strong>php.ini</strong> file</td>
</tr>
<tr>
<td>Path to scan for additional ini configuration files</td>
</tr>
</tbody>
</table>

You can override the Apache MPM prefork settings to increase the performance 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:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTPD_START_SERVERS</strong></td>
<td>The StartServers directive sets the number of child server processes created on startup.</td>
<td>8</td>
</tr>
<tr>
<td><strong>HTTPD_MAX_REQUEST_WORKERS</strong></td>
<td>The MaxRequestWorkers directive sets the limit on the number of simultaneous requests that will be served.</td>
<td>256 (this is automatically tuned by setting Cgroup limits for the container using this formula: [\text{TOTAL_MEMORY} / 15\text{MB}]. 15MB is average size of a single httpd process.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMPOSER_MIRROR</strong></td>
<td>Adds a custom composer repository mirror URL to composer configuration. Note: This only affects packages listed in composer.json.</td>
</tr>
<tr>
<td><strong>COMPOSER_ARGS</strong></td>
<td>Adds extra arguments to the composer install command line (for example, --no-dev).</td>
</tr>
</tbody>
</table>

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.

### 10.2.4. 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:

```bash
$ 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:

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

10.3. PERL

10.3.1. Description

The `rhscl/perl-530-rhel7` image provides a Perl 5.30 platform for building and running applications; the `rhscl/perl-526-rhel7` image provides a Perl 5.26 platform. 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.

10.3.2. Access

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

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

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

```bash
# podman pull registry.redhat.io/rhscl/perl-526-rhel7
```

10.3.3. Configuration

To set environment variables, you can place them as a key–value pair into a `.s2i/environment` file inside your source code repository.
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_CPAN_TEST</td>
<td>Allows the installation of all specified cpan packages and the running of their tests</td>
<td>false</td>
</tr>
<tr>
<td>CPAN_MIRROR</td>
<td>Specifies a mirror URL which will be used by cpanminus to install dependencies</td>
<td>URL is not specified by default</td>
</tr>
<tr>
<td>PERL_APACHE2_RELOAD</td>
<td>Enables automatic reloading of modified Perl modules</td>
<td>false</td>
</tr>
<tr>
<td>HTTPD_START_SERVERS</td>
<td>The StartServers directive sets the number of child server processes created on startup</td>
<td>8</td>
</tr>
<tr>
<td>HTTPD_MAX_REQUEST_WORKERS</td>
<td>Number of simultaneous requests that will be handled by Apache</td>
<td>256 but will be automatically lowered if memory is limited</td>
</tr>
<tr>
<td>PSGI_FILE</td>
<td>Specifies a relative path to the PSGI application file. Use an empty value to disable the PSGI auto-configuration</td>
<td>Single *.psgi file in the top-level directory, if it exists</td>
</tr>
<tr>
<td>PSGI_URI_PATH</td>
<td>Specifies a URI path that is handled by the PSGI application</td>
<td>/</td>
</tr>
</tbody>
</table>

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](#).

### 10.4. PYTHON

#### 10.4.1. Description

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

#### 10.4.2. Access

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

```bash
# podman pull registry.redhat.io/rhscl/python-38-rhel7
```
To set environment variables, you can place them as a key–value pair into a `.s2i/environment` file inside your source code repository.

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

**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.

### 10.5. RUBY

10.5.1. Description

The rhscl/ruby-27-rhel7 image provides a Ruby 2.7 platform for building and running applications; the rhscl/ruby-26-rhel7 image provides a Ruby 2.6 platform, and the rhscl/ruby-25-rhel7 image provides a Ruby 2.5 platform.

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

10.5.2. Access

To pull the rhscl/ruby-27-rhel7 image, run the following command as root:
# podman pull registry.redhat.io/rhscl/ruby-27-rhel7

To pull the rhscl/ruby-26-rhel7 image, run the following command as root:

# podman pull registry.redhat.io/rhscl/ruby-26-rhel7

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

# podman pull registry.redhat.io/rhscl/ruby-25-rhel7

### 10.5.3. Configuration

To set environment variables, you can place them as a key-value pair into a `.s2i/environment` file inside your source code repository.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RACK_ENV</strong></td>
<td>This variable specifies the environment where the Ruby application will be deployed (unless overwritten) - <code>production, development, test</code>. 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 <code>RACK_ENV</code> is set to <code>production</code>.</td>
</tr>
<tr>
<td><strong>DISABLE_ASSET_COMPILATION</strong></td>
<td>This variable set to <code>true</code> indicates that the asset compilation process will be skipped. Because this only takes place when the application is run in the <code>production</code> environment, it should be used only when assets are already compiled.</td>
</tr>
<tr>
<td><strong>PUMA_MIN_THREADS, PUMA_MAX_THREADS</strong></td>
<td>These variables indicate the minimum and maximum threads that will be available in Puma's thread pool.</td>
</tr>
<tr>
<td><strong>PUMA_WORKERS</strong></td>
<td>This variable indicates the number of worker processes that will be launched. See documentation on Puma’s clustered mode.</td>
</tr>
<tr>
<td><strong>RUBYGEM_MIRROR</strong></td>
<td>Set this variable to use a custom RubyGems mirror URL to download required gem packages during the build process.</td>
</tr>
</tbody>
</table>

For S2I scripts to work, you need to include the `puma` or `rack` gem in the application's Gemfile.
CHAPTER 11. DAEMON IMAGES

11.1. APACHE HTTP SERVER

11.1.1. 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.

11.1.2. 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.

11.1.3. 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:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPD_LOG_TO_VOLUME</td>
<td>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 <code>/var/log/httpd24</code>, which can be mounted to host system using the container volumes. This option is allowed only when the container is run as UID 0.</td>
</tr>
<tr>
<td>HTTPD_MPM</td>
<td>This variable can be set to change the default Multi-Processing Module (MPM) from the package default MPM.</td>
</tr>
</tbody>
</table>

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 Table

<table>
<thead>
<tr>
<th>Volume Mount Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/var/www</td>
<td>Apache HTTP Server data directory</td>
</tr>
<tr>
<td>/var/log/httpd24</td>
<td>Apache HTTP Server log directory (available only when running as root)</td>
</tr>
</tbody>
</table>

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.

**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
```

### 11.2. NGINX

#### 11.2.1. Description

The `rhscl/nginx-118-rhel7` image provides an nginx 1.18 server and a reverse proxy server; the image can be used as a base image for other applications based on the nginx 1.18 web server. The `rhscl/nginx-116-rhel7` image provides an nginx 1.16 server and a reverse proxy server.

#### 11.2.2. Access

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

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

To pull the `rhscl/nginx-116-rhel7` image, run the following command as `root`:

```
# podman pull registry.redhat.io/rhscl/nginx-116-rhel7
```

#### 11.2.3. Configuration

The nginx container images support the following configuration variable, which can be set by using the `-e` option with the `podman run` command:
### NGINX LOG TO VOLUME

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGINX_LOG_TO_VOLUME</td>
<td>By default, nginx logs into standard output, so the logs are accessible by using the <code>podman logs</code> command. When <code>NGINX_LOG_TO_VOLUME</code> is set, nginx logs into <code>/var/opt/rh/rh-nginx118/log/nginx/</code> or <code>/var/opt/rh/rh-nginx116/log/nginx/</code>, which can be mounted to host system using the container volumes.</td>
</tr>
</tbody>
</table>

The `rhscl/nginx-118-rhel7` and `rhscl/nginx-116-rhel7` images support using the S2I tool.

### 11.3. VARNISH CACHE

#### 11.3.1. Description

The `rhscl/varnish-6-rhel7` image provides Varnish Cache 6.0, an HTTP reverse proxy. The `rhscl/varnish-5-rhel7` image provides Varnish Cache 5.0.

#### 11.3.2. Access

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

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

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

```
# podman pull registry.redhat.io/rhscl/varnish-5-rhel7
```

#### 11.3.3. 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 12. DATABASE IMAGES

12.1. MARIADB

12.1.1. Description

The `rhscl/mariadb-103-rhel7` image provides a MariaDB 10.3 SQL database server.

12.1.2. Access

To pull the `rhscl/mariadb-103-rhel7` image, run the following command as `root`:

```
# podman pull registry.redhat.io/rhscl/mariadb-103-rhel7
```

12.1.3. 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:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYSQL_USER</td>
<td>User name for MySQL account to be created</td>
</tr>
<tr>
<td>MYSQL_PASSWORD</td>
<td>Password for the user account</td>
</tr>
<tr>
<td>MYSQL_DATABASE</td>
<td>Database name</td>
</tr>
<tr>
<td>MYSQL_ROOT_PASSWORD</td>
<td>Password for the root user (optional)</td>
</tr>
</tbody>
</table>

**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 unprivileged 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:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYSQL_LOWER_CASE_TABLE_NAMES</td>
<td>Sets how the table names are stored and compared</td>
<td>0</td>
</tr>
<tr>
<td>MYSQL_MAX_CONNECTIONS</td>
<td>The maximum permitted number of simultaneous client connections</td>
<td>151</td>
</tr>
<tr>
<td>MYSQL_MAX_ALLOWED_PACKET</td>
<td>The maximum size of one packet or any generated/intermediate string</td>
<td>200M</td>
</tr>
<tr>
<td>MYSQL_FT_MIN_WORD_LENGTH</td>
<td>The minimum length of the word to be included in a FULLTEXT index</td>
<td>4</td>
</tr>
<tr>
<td>MYSQL_FT_MAX_WORD_LENGTH</td>
<td>The maximum length of the word to be included in a FULLTEXT index</td>
<td>20</td>
</tr>
<tr>
<td>MYSQL_AIO</td>
<td>Controls the <code>innodb_use_native_aio</code> setting value in case the native AIO is broken. See <a href="http://help.directadmin.com/item.php?id=529">http://help.directadmin.com/item.php?id=529</a></td>
<td>1</td>
</tr>
<tr>
<td>MYSQL_TABLE_OPEN_CACHE</td>
<td>The number of open tables for all threads</td>
<td>400</td>
</tr>
<tr>
<td>MYSQL_KEY_BUFFER_SIZE</td>
<td>The size of the buffer used for index blocks</td>
<td>32M (or 10% of available memory)</td>
</tr>
<tr>
<td>MYSQL_SORT_BUFFER_SIZE</td>
<td>The size of the buffer used for sorting</td>
<td>256K</td>
</tr>
<tr>
<td>MYSQL_READ_BUFFER_SIZE</td>
<td>The size of the buffer used for a sequential scan</td>
<td>8M (or 5% of available memory)</td>
</tr>
<tr>
<td>MYSQL_INNODB_BUFFER_POOL_SIZE</td>
<td>The size of the buffer pool where InnoDB caches table and index data</td>
<td>32M (or 50% of available memory)</td>
</tr>
<tr>
<td>MYSQL_INNODB_LOG_FILE_SIZE</td>
<td>The size of each log file in a log group</td>
<td>8M (or 15% of available memory)</td>
</tr>
</tbody>
</table>
### MySQL InnoDB Log Buffer Size

The size of the buffer that InnoDB uses to write to the log files on disk is 8M (or 15% of available memory).

### MySQL Defaults File

Point to an alternative configuration file: `/etc/my.cnf`.

### MySQL Binlog Format

Set the binlog format; supported values are `row` and `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:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Default memory percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MYSQL_KEY_BUFFER_SIZE</code></td>
<td>10%</td>
</tr>
<tr>
<td><code>MYSQL_READ_BUFFER_SIZE</code></td>
<td>5%</td>
</tr>
<tr>
<td><code>MYSQL_INNODB_BUFFER_POOL_SIZE</code></td>
<td>50%</td>
</tr>
<tr>
<td><code>MYSQL_INNODB_LOG_FILE_SIZE</code></td>
<td>15%</td>
</tr>
<tr>
<td><code>MYSQL_INNODB_LOG_BUFFER_SIZE</code></td>
<td>15%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Volume Mount Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/var/lib/mysql/data</code></td>
<td>MySQL data directory</td>
</tr>
</tbody>
</table>

### 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.

### 12.1.4 Extending the Image
See How to Extend the rhscl/mariadb-101-rhel7 Container Image, which is applicable also to rhscl/mariadb-103-rhel7.

12.2. MYSQL

12.2.1. Description

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

12.2.2. 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.

12.2.3. Configuration

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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYSQL_USER</td>
<td>User name for MySQL account to be created</td>
</tr>
<tr>
<td>MYSQL_PASSWORD</td>
<td>Password for the user account</td>
</tr>
<tr>
<td>MYSQL_DATABASE</td>
<td>Database name</td>
</tr>
<tr>
<td>MYSQL_ROOT_PASSWORD</td>
<td>Password for the root user (optional)</td>
</tr>
</tbody>
</table>
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 unprivileged 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:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYSQL_LOWER_CASE_TABLE_NAMES</td>
<td>Sets how the table names are stored and compared</td>
<td>0</td>
</tr>
<tr>
<td>MYSQL_MAX_CONNECTIONS</td>
<td>The maximum permitted number of simultaneous client connections</td>
<td>151</td>
</tr>
<tr>
<td>MYSQL_MAX_ALLOWED_PACKET</td>
<td>The maximum size of one packet or any generated/intermediate string</td>
<td>200M</td>
</tr>
<tr>
<td>MYSQL_FT_MIN_WORD_LENGTH</td>
<td>The minimum length of the word to be included in a FULLTEXT index</td>
<td>4</td>
</tr>
<tr>
<td>MYSQL_FT_MAX_WORD_LENGTH</td>
<td>The maximum length of the word to be included in a FULLTEXT index</td>
<td>20</td>
</tr>
<tr>
<td>MYSQL_AIO</td>
<td>Controls the innodb_use_native_aio setting value in case the native AIO is broken. See <a href="http://help.directadmin.com/item.php?id=529">http://help.directadmin.com/item.php?id=529</a></td>
<td>1</td>
</tr>
<tr>
<td>MYSQL_TABLE_OPEN_CACHE</td>
<td>The number of open tables for all threads</td>
<td>400</td>
</tr>
<tr>
<td>Variable name</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>MYSQL_KEY_BUFFER_SIZE</td>
<td>The size of the buffer used for index blocks</td>
<td>32M (or 10% of available memory)</td>
</tr>
<tr>
<td>MYSQL_SORT_BUFFER_SIZE</td>
<td>The size of the buffer used for sorting</td>
<td>256K</td>
</tr>
<tr>
<td>MYSQL_READ_BUFFER_SIZE</td>
<td>The size of the buffer used for a sequential scan</td>
<td>8M (or 5% of available memory)</td>
</tr>
<tr>
<td>MYSQL_INNODB_BUFFER_POOL_SIZE</td>
<td>The size of the buffer pool where InnoDB caches table and index data</td>
<td>32M (or 50% of available memory)</td>
</tr>
<tr>
<td>MYSQL_INNODB_LOG_FILE_SIZE</td>
<td>The size of each log file in a log group</td>
<td>8M (or 15% of available memory)</td>
</tr>
<tr>
<td>MYSQL_INNODB_LOG_BUFFER_SIZE</td>
<td>The size of the buffer that InnoDB uses to write to the log files on disk</td>
<td>8M (or 15% of available memory)</td>
</tr>
<tr>
<td>MYSQL_DEFAULTS_FILE</td>
<td>Point to an alternative configuration file</td>
<td>/etc/my.cnf</td>
</tr>
<tr>
<td>MYSQL_BINLOG_FORMAT</td>
<td>Set sets the binlog format, supported values are row and statement</td>
<td>statement</td>
</tr>
<tr>
<td>MYSQL_LOG_QUERIES_ENABLED</td>
<td>To enable query logging, set this variable to 1</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Default memory percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYSQL_KEY_BUFFER_SIZE</td>
<td>10%</td>
</tr>
<tr>
<td>MYSQL_READ_BUFFER_SIZE</td>
<td>5%</td>
</tr>
<tr>
<td>MYSQL_INNODB_BUFFER_POOL_SIZE</td>
<td>50%</td>
</tr>
<tr>
<td>MYSQL_INNODB_LOG_FILE_SIZE</td>
<td>15%</td>
</tr>
<tr>
<td>MYSQL_INNODB_LOG_BUFFER_SIZE</td>
<td>15%</td>
</tr>
</tbody>
</table>
You can also set the following mount point by passing the `-v /host:/container` option to the `podman run` command:

<table>
<thead>
<tr>
<th>Volume Mount Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/var/lib/mysql/data</td>
<td>MySQL data directory</td>
</tr>
</tbody>
</table>

**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.

### 12.3. MONGODB

#### 12.3.1. Description

The `rhscl/mongodb-36-rhel7` image provides a MongoDB 3.6 NoSQL database server.

#### 12.3.2. Access and Usage

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

```
# podman pull registry.redhat.io/rhscl/mongodb-36-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:

```
# podman 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` daemon writes its logs to the standard output. To examine the container image log, use the `podman logs <image_name>` command.

#### 12.3.3. Configuration

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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONGODB_ADMIN_PASSWORD</td>
<td>Password for the <code>admin</code> user</td>
</tr>
</tbody>
</table>
The administrator user name is set to `admin` and you have 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 unprivileged user has only the `readWrite` role and needs to have a `MONGODB_USER` and `MONGODB_PASSWORD` environment variables set.

**IMPORTANT**

Because passwords are part of the image configuration, the only supported method to change passwords for an unprivileged 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:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MONGODB_QUIET</code></td>
<td>Runs MongoDB in a quiet mode that attempts to limit the amount of output</td>
<td><code>true</code></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Volume Mount Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/var/lib/mongodb/data</code></td>
<td>MongoDB data directory</td>
</tr>
</tbody>
</table>
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.

12.3.4. 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
```

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

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mongodb-cfg/</td>
<td>When running the <strong>run-mongod</strong> or <strong>run-mongod-replication</strong> commands, the contained mongod.conf file is used for <strong>mongod</strong> configuration. The <strong>envsubst</strong> 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.</td>
</tr>
<tr>
<td>mongodb-pre-init/</td>
<td>The contained shell scripts (*.sh) are sourced before the <strong>mongod</strong> server is started.</td>
</tr>
<tr>
<td>mongodb-init/</td>
<td>The contained shell scripts (*.sh) are sourced when the <strong>mongod</strong> server is started for the first time, that is, when the data directory is empty. The <strong>run-mongod</strong> command does not have an enabled authentication in this phase. The <strong>run-mongod-replication</strong> command does have an enabled authentication in this phase.</td>
</tr>
<tr>
<td>mongodb-start/</td>
<td>Similar to <strong>mongodb-init/</strong>, except these scripts are always sourced (after the <strong>mongodb-init/</strong> scripts, if they exist). These scripts are skipped if <strong>run-mongod-replication</strong> is run with an already initialized data directory.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mongo_common_args</td>
<td>Contains arguments for the <strong>mongod</strong> server. Note that changing this can break existing customization scripts, that is, the default scripts.</td>
</tr>
</tbody>
</table>
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 podman 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.

12.4. POSTGRESQL

12.4.1. Description

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

12.4.2. Access and Usage

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
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.

### 12.4.3. Configuration

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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSTGRESQL_USER</td>
<td>User name for PostgreSQL account to be created</td>
</tr>
<tr>
<td>POSTGRESQL_PASSWORD</td>
<td>Password for the user account</td>
</tr>
<tr>
<td>POSTGRESQL_DATABASE</td>
<td>Database name</td>
</tr>
<tr>
<td>POSTGRESQL_ADMIN_PASSWORD</td>
<td>Password for the postgres admin account (optional)</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSTGRESQL_MIGRATION_REMOTE_HOST</td>
<td>Hostname/IP to migrate from</td>
<td></td>
</tr>
<tr>
<td>POSTGRESQL_MIGRATION_ADMIN_PASSWORD</td>
<td>Password for the remote <code>postgres</code> admin user</td>
<td></td>
</tr>
</tbody>
</table>
### Variable Name

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSTGRESQL_MIGRATION_IGNORE_ERRORS</td>
<td>Optional: Ignore sql import errors</td>
<td>no</td>
</tr>
</tbody>
</table>

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

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

**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:

<table>
<thead>
<tr>
<th>Volume Mount Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/var/lib/pgsql/data</td>
<td>PostgreSQL database cluster directory</td>
</tr>
</tbody>
</table>
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 <...>
```

12.4.4. 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 performs “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.
12.4.5. Upgrading the Database

The PostgreSQL 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-10-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:

<table>
<thead>
<tr>
<th>copy</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>hardlink</td>
<td>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.</td>
</tr>
</tbody>
</table>
NOTE
Make sure you have enough space for the copied data. Upgrade failure because of insufficient space might lead to a data loss.

12.4.6. 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:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>postgresql-pre-start/</td>
<td>Source all *.sh files from this directory during an early start of the container. There is no PostgreSQL daemon running in the background.</td>
</tr>
<tr>
<td>postgresql-cfg/</td>
<td>Contained configuration files (*.conf) will be included at the end of the image’s postgresql.conf file.</td>
</tr>
<tr>
<td>postgresql-init/</td>
<td>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).</td>
</tr>
<tr>
<td>postgresql-start/</td>
<td>Similar to postgresql-init/, except these scripts are always sourced (after the postgresql-init/ scripts, if they exist).</td>
</tr>
</tbody>
</table>

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.

12.5. REDIS

12.5.1. Description
The rhscl/redis-5-rhel7 image provides Redis 5, an advanced key-value store. The image is based on the rh-redis5 Software Collection.

12.5.2. Access
To pull the rhscl/redis-5-rhel7 image, run the following command as root:
12.5.3. Configuration and Usage

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

```bash
# podman run -d --name redis_database -p 6379:6379 rhscl/redis-5-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:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REDIS_PASSWORD</td>
<td>Password for the server access</td>
</tr>
</tbody>
</table>

To set a password, run:

```bash
# podman run -d --name redis_database -e REDIS_PASSWORD=strongpassword rhscl/redis-5-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.

<table>
<thead>
<tr>
<th>Volume Mount Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/var/lib/redis/data</td>
<td>Redis data directory</td>
</tr>
</tbody>
</table>

**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 13. 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 10 User Guide.

IMPORTANT

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

13.1. 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-10-toolchain-rhel7 /bin/bash -l
```

Example 13.1. 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-10-toolchain-rhel7
```

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

```
# podman run -ti rhscl/devtoolset-10-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:

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

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

## 13.2. 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 container images.

Red Hat Developer Toolset 10.0 for Red Hat Enterprise Linux 7 is distributed with the following Dockerfiles:

- `devtoolset-10-toolchain`
- `devtoolset-10-perftools`

### 13.2.1. Obtaining Dockerfiles

The Red Hat Developer Toolset Dockerfiles are provided in the Red Hat Container Catalog. Every tagged container image is provided with the Dockerfile used to build it.

To get the Dockerfiles, follow this procedure:

1. Open the Red Hat Container Catalog page website for the latest Red Hat Developer Toolset container image:
   - Red Hat Developer Toolset 10.0

2. Copy the Dockerfile contents from the webpage.

3. Paste the text to a suitably named plain text file on your system.

### 13.2.2. Building Container Images

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

```bash
# podman build -t image_name
```

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

**Example 13.2. Building a Container Image with a Red Hat Developer Toolset Component**
To build a container image for deploying the `perftools` tools from Red Hat Developer Toolset 10.0 in a container, follow the instructions below:

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. Copy the Dockerfile contents text from the Red Hat Container Catalog page for Red Hat Developer Toolset 10.0 perftools image.

3. Paste the text into a text editor and save it as the file `/home/user/devtoolset-10-toolchain-rhel7/Dockerfile`
   - Replace `user` with name of the current user. To determine this value, run the `whoami` command in the terminal.

4. Change to the directory with the file:
   ```
   # cd /home/user/devtoolset-10-toolchain-rhel7/
   ```

5. Build the container image:
   ```
   # podman build -t devtoolset-10-my-perftools
   ```
   - Replace `devtoolset-10-my-perftools` with the name you wish to assign to your resulting container image.

### 13.2.3. 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 13.2.2, “Building Container Images”), run the following command as `root`:

```
podman run image_name container-usage
```

To launch an interactive shell within a container image you built, 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 chose when building it.

**Example 13.3. Using elfutils in a Custom-Built Red Hat Developer Toolset Image**

This example illustrates how to launch a custom-built 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:
   ```
   # podman run -ti devtoolset-10-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 `podman` command:
$ sudo podman run -v ~/Source:/src -ti devtoolset-10-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
   [...]

### 13.3. ADDITIONAL RESOURCES

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

- [Red Hat Developer Toolset 10.0 User Guide](#)
- [Red Hat Developer Toolset 10.0 Release Notes](#)

### 13.4. RED HAT DEVELOPER TOOLSET TOOLCHAIN

#### 13.4.1. Description

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

The rhscl/devtoolset-10-toolchain-rhel7 image provides content corresponding to the following packages:

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>10.2.1</td>
<td>devtoolset-10-gcc</td>
</tr>
<tr>
<td>g++</td>
<td></td>
<td>devtoolset-10-gcc-c++</td>
</tr>
<tr>
<td>gfortran</td>
<td></td>
<td>devtoolset-10-gcc-fortran</td>
</tr>
<tr>
<td>gdb</td>
<td>9.2</td>
<td>devtoolset-10-gdb</td>
</tr>
</tbody>
</table>

Additionally, the devtoolset-10-binutils package is included as a dependency.

#### 13.4.2. Access

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

```
# podman pull registry.redhat.io rhscl/devtoolset-10-toolchain-rhel7
```
13.5. RED HAT DEVELOPER TOOLSET PERFORMANCE TOOLS

13.5.1. Description

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

The `rhsc1/devtoolset-10-perftools-rhel7` image provides the following components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwz</td>
<td>0.12</td>
<td>devtoolset-10-dwz</td>
</tr>
<tr>
<td>Dyninst</td>
<td>10.1.0</td>
<td>devtoolset-10-dyninst</td>
</tr>
<tr>
<td>elfutils</td>
<td>0.180</td>
<td>devtoolset-10-elfutils</td>
</tr>
<tr>
<td>ltrace</td>
<td>0.7.91</td>
<td>devtoolset-10-ltrace</td>
</tr>
<tr>
<td>make</td>
<td>4.2.1</td>
<td>devtoolset-10-make</td>
</tr>
<tr>
<td>memstomp</td>
<td>0.1.5</td>
<td>devtoolset-10-memstomp</td>
</tr>
<tr>
<td>OPProfile</td>
<td>1.4.0</td>
<td>devtoolset-10-oprofile</td>
</tr>
<tr>
<td>strace</td>
<td>5.7</td>
<td>devtoolset-10-strace</td>
</tr>
<tr>
<td>SystemTap</td>
<td>4.3</td>
<td>devtoolset-10-systemtap</td>
</tr>
<tr>
<td>Valgrind</td>
<td>3.16.1</td>
<td>devtoolset-10-valgrind</td>
</tr>
</tbody>
</table>

Additionally, the `devtoolset-10-gcc` and `devtoolset-10-binutils` packages are included as a dependency.

13.5.2. Access

To pull the `rhsc1/devtoolset-10-perftools-rhel7` image, run the following command as `root`:

```
# podman pull registry.redhat.io/rhsc1/devtoolset-10-perftools-rhel7
```

13.5.3. 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:

```bash
$ podman run --ti --privileged --ipc=host --net=host --pid=host devtoolset-10-my-perftools /bin/bash -l
```

To use the pre-built `perftools` image, substitute the image name for `devtoolset-10-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 host system's kernel:

   ```bash
   $ uname -r
   3.10.0-514.10.2.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`:

   ```bash
   $ 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:

   ```bash
   $ podman commit devtoolset-10-systemtap-$(uname -r)
   ```
CHAPTER 14. 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 15. REVISION HISTORY

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Change</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1-9</td>
<td>Apr 06 2021</td>
<td>Improved supported architectures.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-8</td>
<td>Jan 13 2021</td>
<td>Improved introductory chapters and extended information about building application images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-7</td>
<td>Dec 01 2020</td>
<td>Release of Using Red Hat Software Collections 3.6 Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-6</td>
<td>Oct 29 2020</td>
<td>Release of Using Red Hat Software Collections 3.6 Beta Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-5</td>
<td>May 26 2020</td>
<td>Release of Using Red Hat Software Collections 3.5 Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-4</td>
<td>Apr 21 2020</td>
<td>Release of Using Red Hat Software Collections 3.5 Beta Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-3</td>
<td>Dec 10 2019</td>
<td>Release of Using Red Hat Software Collections 3.4 Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-2</td>
<td>Nov 07 2019</td>
<td>Release of Using Red Hat Software Collections 3.4 Beta Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-1</td>
<td>Jun 11 2019</td>
<td>Release of Using Red Hat Software Collections 3.3 Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.1-0</td>
<td>Apr 16 2019</td>
<td>Release of Using Red Hat Software Collections 3.3 Beta Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.0-9</td>
<td>Nov 13 2018</td>
<td>Release of Using Red Hat Software Collections 3.2 Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.0-8</td>
<td>Oct 23 2018</td>
<td>Release of Using Red Hat Software Collections 3.2 Beta Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.0-8</td>
<td>Aug 29 2018</td>
<td>Added a known issue related to SystemTap in devtoolset-6-perftools.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.0-7</td>
<td>May 10 2018</td>
<td>Extended MongoDB image documentation.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.0-6</td>
<td>May 03 2018</td>
<td>Release of Using Red Hat Software Collections 3.1 Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.0-5</td>
<td>Apr 04 2018</td>
<td>Release of Using Red Hat Software Collections 3.1 Beta Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Change</td>
<td>Author</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>0.0-3</td>
<td>Nov 29 2017</td>
<td>Added the Extending Existing Container Images section.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.0-2</td>
<td>Oct 24 2017</td>
<td>Release of Using Red Hat Software Collections 3.0 Container Images.</td>
<td>Lenka Špačková</td>
</tr>
<tr>
<td>0.0-1</td>
<td>Oct 03 2017</td>
<td>Release of Using Red Hat Software Collections 3.0 Beta Container Images.</td>
<td>Lenka Špačková</td>
</tr>
</tbody>
</table>