# Red Hat Enterprise Linux Atomic Host 7

## Getting Started with Containers

Use podman, skopeo, and buildah to work with containers in Red Hat Enterprise Linux 7 and RHEL Atomic Host

Last Updated: 2019-11-02

# Red Hat Enterprise Linux Atomic Host 7 Getting Started with Containers

Use podman, skopeo, and buildah to work with containers in Red Hat Enterprise Linux 7 and RHEL Atomic Host

## Legal Notice

Copyright © 2019 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

Containers and Container Development

# Table of Contents

CHAPTER 1. GET STARTED WITH LINUX CONTAINERS ..... 4
1.1. OVERVIEW 4
1.2. BACKGROUND 4
1.3. SUPPORTED ARCHITECTURES FOR CONTAINERS ON RHEL 5
1.4. GETTING CONTAINER TOOLS IN RHEL 7 9
1.5. ENABLING CONTAINER SETTINGS 10
1.6. USING CONTAINERS AS ROOT OR ROOTLESS 10
1.7. WORKING WITH CONTAINER IMAGES 10
1.7.1. Getting images from registries 11
1.7.2. Investigating images 12
1.7.2.1. Listing images 12
1.7.2.2. Inspecting local images 12
1.7.2.3. Inspecting remote images 14
1.7.3. Tagging images 14
1.7.4. Saving and importing images 15
1.7.5. Removing Images 15
1.8. WORKING WITH CONTAINERS 16
1.8.1. Running containers 16
1.8.2. Investigating running and stopped containers 18
1.8.2.1. Listing containers 18
1.8.2.2. Inspecting containers 18
1.8.2.3. Investigating within a container 19
1.8.3. Starting and stopping containers 20
1.8.3.1. Starting containers 20
1.8.3.2. Stopping containers 21
1.8.4. Removing containers 21

CHAPTER 2. USING RED HAT UNIVERSAL BASE IMAGES (STANDARD, MINIMAL, AND RUNTIMES) ..... 22
2.1. WHAT ARE RED HAT BASE IMAGES? 22
2.1.1. Using standard Red Hat base images 22
2.1.2. Using minimal Red Hat base images 23
2.1.3. Using Init Red Hat base images 24
2.2. HOW ARE UBI IMAGES DIFFERENT? 24
2.3. GET UBI IMAGES 25
2.4. PULL UBI IMAGES 25
2.5. REDISTRIBUTING UBI IMAGES 25
2.6. RUN UBI IMAGES 26
2.7. ADD SOFTWARE TO A RUNNING UBI CONTAINER 26
2.7.1. Adding software to a UBI container (subscribed host) 26
2.7.2. Adding software inside the standard UBI container 27
2.7.3. Adding software inside the minimal UBI container 27
2.7.4. Adding software to a UBI container (unsubscribed host) 28
2.8. BUILD A UBI-BASED IMAGE 28
2.9. USING RED HAT SOFTWARE COLLECTIONS RUNTIME IMAGES 29
2.10. GETTING UBI CONTAINER IMAGE SOURCE CODE 30
2.11. TIPS AND TRICKS FOR USING UBI IMAGES 30
2.12. HOW TO REQUEST NEW FEATURES IN UBI? 31
2.13. HOW TO FILE A SUPPORT CASE FOR UBI? 31

CHAPTER 3. INSTALL AND DEPLOY AN APACHE WEB SERVER CONTAINER ..... 32
3.1. OVERVIEW 32

3.2. CREATING AND RUNNING THE APACHE WEB SERVER CONTAINER 32
3.3. TIPS FOR THIS CONTAINER 34
3.4. ATTACHMENTS 34

**CHAPTER 4. INSTALL AND DEPLOY A MARIADB CONTAINER 36**
4.1. OVERVIEW 36
4.2. CREATING AND RUNNING THE MARIADB DATABASE SERVER CONTAINER 36
4.3. TIPS FOR THIS CONTAINER 38
4.4. ATTACHMENTS 38

**CHAPTER 5. USING THE DOCKER COMMAND AND SERVICE 39**
5.1. OVERVIEW 39
5.2. GETTING DOCKER IN RHEL 7 39
5.3. GETTING DOCKER IN RHEL ATOMIC HOST 41
5.4. CHANGING THE DOCKER SERVICE 42
5.5. MODIFYING THE DOCKER DAEMON OPTIONS (/ETC/SYSCONFIG/DOCKER) 43
5.5.1. Default options 43
5.5.2. Access port options 43
5.5.2.1. Exposing the docker daemon through a TCP port 43
5.5.3. Registry options 43
5.5.4. User namespaces options 44
5.6. WORKING WITH DOCKER REGISTRIES 45
5.6.1. Creating a private Docker registry (optional) 46
5.6.2. Getting images from remote Docker registries 47
5.7. INVESTIGATING DOCKER IMAGES 48
5.8. INVESTIGATING THE DOCKER ENVIRONMENT 49
5.9. WORKING WITH DOCKER FORMATTED CONTAINERS 50
5.10. CREATING DOCKER IMAGES 57
5.10.1. Building an image from a Dockerfile 57
5.10.2. Creating an image from a container 60
5.11. TAGGING IMAGES 61
5.12. SAVING AND IMPORTING IMAGES 61
5.13. REMOVING IMAGES 62
5.14. SUMMARY 62

# CHAPTER 1. GET STARTED WITH LINUX CONTAINERS

## 1.1. OVERVIEW

Linux Containers have emerged as a key open source application packaging and delivery technology, combining lightweight application isolation with the flexibility of image-based deployment methods.

Red Hat Enterprise Linux implements Linux Containers using core technologies such as Control Groups (Cgroups) for Resource Management, Namespaces for Process Isolation, SELinux for Security, enabling secure multi-tenancy and reducing the risk of security exploits. All this is meant to provide you with an environment for producing and running enterprise-quality containers.

Red Hat OpenShift provides powerful command-line and Web UI tools for building, managing and running containers in units referred to as **pods**. However, sometimes you might want to build and manage individual containers and images outside of OpenShift. Some tools provided to perform those tasks that run directly on RHEL systems are described in this guide.

Unlike other container tools implementations, tools described here do not center around the monolithic Docker container engine and **docker** command. Instead, we provide a set of command-line tools that can operate without a container engine. These include:

- **podman** - For directly managing pods and container images (run, stop, start, ps, attach, exec, and so on)
- **buildah** - For building, pushing and signing container images
- **skopeo** - For copying, inspecting, deleting, and signing images
- **runc** - For providing container run and build features to podman and buildah

Because these tools are compatible with the Open Container Initiative (OCI), they can be used to manage the same Linux containers that are produced and managed by Docker and other OCI-compatible container engines. However, they are especially suited to run directly on Red Hat Enterprise Linux, in single-node use cases.

For a multi-node container platform, see OpenShift. Instead of relying on the single-node, daemonless tools described in this document, OpenShift requires a daemon-based container engine. Please see Using the CRI-O Container Engine for details.

While this guide introduces you to container tools and images, see Managing Containers for more details on those tools.

If you are still interested in using the **docker** command and docker service, refer to Using the docker command and service for information on how to use those features in RHEL 7.

## 1.2. BACKGROUND

Containers provide a means of packaging applications in lightweight, portable entities. Running applications within containers offers the following advantages:

- **Smaller than Virtual Machines** Because container images include only the content needed to run an application, saving and sharing is much more efficient with containers than it is with virtual machines (which include entire operating systems)

- **Improved performance**: Likewise, since you are not running an entirely separate operating system, a container will typically run faster than an application that carries with it the overhead of a whole new virtual machine.
- **Secure**: Because a container typically has its own network interfaces, file system, and memory, the application running in that container can be isolated and secured from other activities on a host computer.
- **Flexible**: With an application's run time requirements included with the application in the container, a container is capable of being run in multiple environments.

Currently, you can run containers on Red Hat Enterprise Linux 7 (RHEL 7) Server, Workstation, and Atomic Host systems. If you are unfamiliar with RHEL Atomic Host, you can learn more about it from RHEL Atomic Host 7 Installation and Configuration Guide or the upstream Project Atomic site. Project Atomic produces smaller derivatives of RPM-based Linux distributions (RHEL, Fedora, and CentOS) that is made specifically to run containers in OpenStack, VirtualBox, Linux KVM and several different cloud environments.

This topic will help you get started with containers in RHEL 7 and RHEL Atomic Host. Besides offering you some hands-on ways of trying out containers, it also describes how to:

- Access RHEL-based container images from the Red Hat Registry
- Incorporate RHEL-entitled software into your containers

## 1.3. SUPPORTED ARCHITECTURES FOR CONTAINERS ON RHEL

Red Hat provides container images and container-related software for the following architectures:

- X86 64-bit (base and layered images) (no support for X86 32-bit)
- PowerPC 8 64-bit (base image and most layered images)
- PowerPC 9 64-bit (base image and most layered images)
- IBM s390x (base image and most layered images)
- ARM 64-bit (base image only, separate container repo)

**NOTE**

Container images for all architectures are available from the same repository in the Red Hat Registry, with one exception. ARM 64-bit images are available from the rhel7-aarch64 repo in the Red Hat Registry. Currently, only the rhel base image is available with the ARM architecture: registry.redhat.io/rhel7-aarch64

For RHEL and RHEL Atomic 7.5.3, many of the Red Hat-supported container images that were available only on X86_64 became available in the other supported architectures as well. These images are described in this guide and the Managing Containers Guide. Table 1 notes which Red Hat container images are supported on each architecture.

**Table 1.1. Red Hat container images and supported architectures**

| Image name | X86_64 | PowerPC 8 & 9 | s390x | ARM 64 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| rhel7/flannel | Yes | No | No | No |
| rhel7/ipa-server | Yes | No | No | No |
| rhel7/open-vm-tools | Yes | No | No | No |
| rhel7/rhel-tools | Yes | No | No | No |
| rhel7/support-tools | Yes | No | No | No |
| rhel7/rhel | Yes | Yes | Yes | No |
| rhel7-init | Yes | Yes | Yes | No |
| rhel7/net-snmp | Yes | Yes | Yes | No |
| rhel7/sssd | Yes | Yes | Yes | No |
| rhel7/sadc | Yes | Yes | Yes | No |
| rhel7/etcd | Yes | Yes | Yes | No |
| rhel7/rsyslog | Yes | Yes | Yes | No |
| rhel7/cockpit-ws | Yes | Yes | Yes | No |
| rhel7-minimal / rhel7-atomic | Yes | Yes | Yes | No |
| rhel7/openscap | Yes | Yes | Yes | No |
| rhel7-aarch64 | No | No | No | Yes |

As you use the Red Hat containers on the different architectures, you should be aware of a few issues:

- The Power 8 and Power 9 base image was originally referred to as **rhel7-ppc64le**. That image name will be retired in RHEL 7.6. For RHEL 7.5.2 and later, simply refer to it as rhel7/rhel.
- There is no RHEL Atomic Host installation medium available for IBM Power or s390x systems. You must use installation media from Red Hat that provides the IBM Power or s390x versions of RHEL Server.
- The container-related software repositories that you enable with subscription-manager are different for X86_64 systems, Power, ARM, and s390x systems. See the Table 2 for the repository names to use in place of the X86_64 repository names for Power, ARM, and s390x systems.

Table 1.2. RHEL Server container-related software repos for different architectures

| Repository Name | Description |
|---|---|
| **Power 8** | **Red Hat Enterprise Linux Server** |
| rhel-7-for-power-le-rpms | Red Hat Enterprise Linux 7 for IBM Power LE (RPMs) |
| rhel-7-for-power-le-debug-rpms | Red Hat Enterprise Linux 7 for IBM Power LE (Debug RPMs) |
| rhel-7-for-power-le-source-rpms | Red Hat Enterprise Linux 7 for IBM Power LE (Source RPMs) |
| rhel-7-for-power-le-extras-rpms | Red Hat Enterprise Linux 7 for IBM Power LE - Extras (RPMs) |
| rhel-7-for-power-le-extras-debug-rpms | Red Hat Enterprise Linux 7 for IBM Power LE - Extras (Debug RPMs) |
| rhel-7-for-power-le-extras-source-rpms | Red Hat Enterprise Linux 7 for IBM Power LE - Extras (Source RPMs) |
| rhel-7-for-power-le-optional-rpms | Red Hat Enterprise Linux 7 for IBM Power LE - Optional (RPMs) |
| rhel-7-for-power-le-optional-debug-rpms | Red Hat Enterprise Linux 7 for IBM Power LE - Optional (Debug RPMs) |
| rhel-7-for-power-le-optional-source-rpms | Red Hat Enterprise Linux 7 for IBM Power LE - Optional (Source RPMs) |
| **Power 9** | **Red Hat Enterprise Linux Server** |
| rhel-7-for-power-9-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 (RPMs) |
| rhel-7-for-power-9-debug-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 (Debug RPMs) |
| rhel-7-for-power-9-source-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 (Source RPMs) |
| rhel-7-for-power-9-extras-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 - Extras (RPMs) |
| rhel-7-for-power-9-extras-debug-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 - Extras (Debug RPMs) |
| rhel-7-for-power-9-extras-source-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 - Extras (Source RPMs) |
| rhel-7-for-power-9-optional-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 - Optional (RPMs) |
| rhel-7-for-power-9-optional-debug-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 - Optional (Debug RPMs) |

| | |
|---|---|
| rhel-7-for-power-9-optional-source-rpms | Red Hat Enterprise Linux 7 for IBM Power 9 - Optional (Source RPMs) |
| **s390x (RHEL-ALT)** | **Red Hat Enterprise Linux Server** |
| rhel-7-for-system-z-a-rpms | Red Hat Enterprise Linux 7 for IBM s390x (RPMs) |
| rhel-7-for-system-z-a-debug-rpms | Red Hat Enterprise Linux 7 for IBM s390x (Debug RPMs) |
| rhel-7-for-system-z-a-source-rpms | Red Hat Enterprise Linux 7 for IBM s390x (Source RPMs) |
| rhel-7-for-system-z-a-extras-rpms | Red Hat Enterprise Linux 7 for IBM s390x - Extras (RPMs) |
| rhel-7-for-system-z-a-extras-debug-rpms | Red Hat Enterprise Linux 7 for IBM s390x - Extras (Debug RPMs) |
| rhel-7-for-system-z-a-extras-source-rpms | Red Hat Enterprise Linux 7 for IBM s390x - Extras (Source RPMs) |
| rhel-7-for-system-z-a-extras-optional-rpms | Red Hat Enterprise Linux 7 for IBM s390x - Optional (RPMs) |
| rhel-7-for-system-z-a-extras-optional-debug-rpms | Red Hat Enterprise Linux 7 for IBM s390x - Optional (Debug RPMs) |
| rhel-7-for-system-z-a-extras-optional-source-rpms | Red Hat Enterprise Linux 7 for IBM s390x - Optional (Source RPMs) |
| **ARM (RHEL-ALT)** | **Red Hat Enterprise Linux Server** |
| rhel-7-for-arm-64-rpms | Red Hat Enterprise Linux 7 for ARM (RPMs) |
| rhel-7-for-arm-64-debug-rpms | Red Hat Enterprise Linux 7 for ARM (Debug RPMs) |
| rhel-7-for-arm-64-source-rpms | Red Hat Enterprise Linux 7 for ARM (Source RPMs) |
| rhel-7-for-arm-64-extras-rpms | Red Hat Enterprise Linux 7 for ARM - Extras (RPMs) |
| rhel-7-for-arm-64-extras-debug-rpms | Red Hat Enterprise Linux 7 for ARM - Extras (Debug RPMs) |
| rhel-7-for-arm-64-extras-source-rpms | Red Hat Enterprise Linux 7 for ARM - Extras (Source RPMs) |
| rhel-7-for-arm-64-extras-optional-rpms | Red Hat Enterprise Linux 7 for ARM - Optional (RPMs) |
| rhel-7-for-arm-64-extras-optional-debug-rpms | Red Hat Enterprise Linux 7 for ARM - Optional (Debug RPMs) |

| | |
|---|---|
| rhel-7-for-arm-64-extras-optional-source-rpms | Red Hat Enterprise Linux 7 for ARM - Optional (Source RPMs) |

## 1.4. GETTING CONTAINER TOOLS IN RHEL 7

To get an environment where you can work with individual containers, you can install a Red Hat Enterprise Linux 7 system. Using the RHEL 7 subscription model, if you want to create images or containers, you must properly register and entitle the host computer on which you build them. When you use **yum install** within a container to add packages, the container automatically has access to entitlements available from the RHEL 7 host, so it can get RPM packages from any repository enabled on that host.

1. **Install RHEL**: If you are ready to begin, you can start by installing a Red Hat Enterprise Linux system (Workstation or Server edition) as described in the following: Red Hat Enterprise Linux 7 Installation Guide

**NOTE**

Running containers on RHEL 7 Workstations has some limitations:

- Standard single-user, single-node rules apply to running containers on RHEL Workstations.
- Only Universal Base Image (UBI) content is supported when you build containers on RHEL workstations. In other words, you cannot include RHEL Server RPMS.
- You can run containers supported by third party ISVs, such as compilers.

2. **Register RHEL**: Once RHEL 7 is installed, register the system. You will be prompted to enter your user name and password. Note that the user name and password are the same as your login credentials for Red Hat Customer Portal.

```
# subscription-manager register
Registering to: subscription.rhsm.redhat.com:443/subscription
Username: ********
Password: **********
```

3. **Choose pool ID**: Determine the pool ID of a subscription that includes Red Hat Enterprise Linux Server. Type the following at a shell prompt to display a list of all subscriptions that are available for your system, then attach the pool ID of one that meets that requirement:

```
# subscription-manager list --available    Find valid RHEL pool ID
# subscription-manager attach --pool=pool_id
```

4. **Enable repositories**: Enable the following repositories, which will allow you to install the docker package and related software:
**NOTE**: The repos shown here are for X86_64 architectures. See Supported Architectures for Containers on RHEL to learn the names of repositories for other architectures.

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

It is possible that some Red Hat subscriptions include enabled repositories that can conflict with eachother. If you believe that has happened, before enabling the repos shown above, you can disable all repos. See the How are repositories enabled solution for information on how to disable unwanted repositories.

5. **Install packages**: To install the **podman**, **skopeo**, and **buildah** packages, type the following:

```
# yum install podman skopeo buildah -y
```

## 1.5. ENABLING CONTAINER SETTINGS

No container engine (such as Docker or CRI-O) is required for you to run containers on your local system. However, configuration settings in the **/etc/containers/registries.conf** file let you define access to container registries when you work with container tools such as **podman** and **buildah**.

Here are example settings in the **/etc/containers/registries.conf** file:

```
[registries.search]
registries = ['registry.access.redhat.com', 'docker.io', 'registry.fedoraproject.org', 'quay.io',
'registry.centos.org']

[registries.insecure]
registries = []

[registries.block]
registries = []
```

By default, when you use **podman search** to search for images from a container registries, based on the **registries.conf** file, **podman** looks for the requested image in registry.access.redhat.com, docker.io, registry.fedoraproject.org, docker.io, and registry.centos.org in that order.

To add access to a registry that doesn't require authentication (an insecure registry), you must add the name of that registry under the **[registries.insecure]** section. Any registries that you want to disallow from access from your local system need to be added under the **[registries.block]** section.

## 1.6. USING CONTAINERS AS ROOT OR ROOTLESS

At first, root privilege (either as the root user or as a regular user with sudo privilege) was required to work with container tools in RHEL. As of RHEL 7.7, the rootless container feature (currently a Technology Preview) lets regular user accounts work with containers. All container tools described in this document can be run as root user. For restrictions on running these from regular user accounts, see the rootless containers section of the Managing Containers guide.

## 1.7. WORKING WITH CONTAINER IMAGES

Using podman, you can run, investigate, start, stop, and remove container images. If you are familiar with the **docker** command, you will notice that you can use the same syntax with **podman** to work with containers and container images.

### 1.7.1. Getting images from registries

To get images from a remote registry (such as Red Hat's own Docker registry) and add them to your local system, use the **podman pull** command:

```
# podman pull <registry>[:<port>]/[<namespace>/]<name>:<tag>
```

The *<registry>* is a host that provides the registry service on TCP *<port>* (default: 5000). Together, *<namespace>* and *<name>* identify a particular image controlled by *<namespace>* at that registry. Some registries also support raw *<name>*; for those, *<namespace>* is optional. When it is included, however, the additional level of hierarchy that *<namespace>* provides is useful to distinguish between images with the same *<name>*. For example:

| Namespace | Examples (*<namespace>*/*<name>*) |
|---|---|
| organization | **redhat/kubernetes**, **google/kubernetes** |
| login (user name) | **alice/application**, **bob/application** |
| role | **devel/database**, **test/database**, **prod/database** |

The registries that Red Hat supports are registry.redhat.io (requiring authentication) and registry.access.redhat.com (requires no authentication, but is deprecated). For details on the transition to registry.redhat.io, see Red Hat Container Registry Authentication . Before you can pull containers from registry.redhat.io, you need to authenticate. For example:

```
# podman login registry.redhat.io
Username: myusername
Password: ************
Login Succeeded!
```

To get started with container images, you can use the pull option to pull an image from a remote registry. To pull the RHEL 7 UBI base image and rsyslog image from the Red Hat registry, type:

```
# podman pull registry.access.redhat.com/ubi7/ubi
# podman pull registry.access.redhat.com/rhel7/rsyslog
```

An image is identified by a repository name (registry.access.redhat.com), a namespace name (rhel7) and the image name (rsyslog). You could also add a tag (which defaults to :latest if not entered). The repository name **rhel7**, when passed to the **podman pull** command without the name of a registry preceding it, is ambiguous and could result in the retrieval of an image that originates from an untrusted registry. If there are multiple versions of the same image, adding a tag, such as **latest** to form a name such as **rsyslog:latest**, lets you choose the image more explicitly.

To see the images that resulted from the above **podman pull** command, along with any other images on your system, type **podman images**:

```
# podman images
REPOSITORY
          TAG       IMAGE ID      CREATED       VIRTUAL SIZE
registry.access.redhat.com/rhel7/rsyslog
```

```
        latest     39ec6b2004a3  9 days ago    236 MB
registry.access.redhat.com/ubi7/ubi
        latest     967cb403b7ee  3 days ago     215 MB
```

### 1.7.2. Investigating images

Using **podman images** you can see which images have been pulled to your local system. To look at the metadata associated with an image, use **podman inspect**.

#### 1.7.2.1. Listing images

To see which images have been pulled to your local system and are available to use, type:

```
# podman images
REPOSITORY                                    TAG    IMAGE ID     CREATED      VIRTUAL SIZE
registry.access.redhat.com/rhel7/rsyslog              latest 39ec6b2004a3 10 days ago  236 MB
registry.access.redhat.com/ubi7/ubi                   latest 967cb403b7ee 10 days ago  215 MB
registry.access.redhat.com/rhscl/postgresql-10-rhel7  1-35   27b15d85ca6b 3 months ago 336 MB
```

#### 1.7.2.2. Inspecting local images

After you pull an image to your local system and before you run it, it is a good idea to investigate that image. Reasons for investigating an image before you run it include:

- Understanding what the image does
- Checking what software is inside the image

The **podman inspect** command displays basic information about what an image does. You also have the option of mounting the image to your host system and using tools from the host to investigate what's in the image. Here is an example of investigating what a container image does before you run it:

1. **Inspect an image**: Run **podman inspect** to see what command is executed when you run the container image, as well as other information. Here are examples of examining the ubi7/ubi and rhel7/rsyslog container images (with only snippets of information shown here):

```
# podman inspect registry.access.redhat.com/ubi7/ubi
...
Cmd": [
        "/bin/bash"
        ],
        "Labels": {
          "architecture": "x86_64",
          "authoritative-source-url": "registry.access.redhat.com",
          "build-date": "2019-08-01T09:28:54.576292",
          "com.redhat.build-host": "cpt-1007.osbs.prod.upshift.rdu2.redhat.com",
          "com.redhat.component": "ubi7-container",
          "com.redhat.license_terms": "https://www.redhat.com/en/about/red-hat-end-user-
license-agreements#UBI",
          "description": "The Universal Base Image is designed and engineered to be ...
...
```

```
# podman inspect registry.access.redhat.com/rhel7/rsyslog
  "Cmd": [
```

```
      "/bin/rsyslog.sh"
    ],
    "Labels": {
      "License": "GPLv3",
      "architecture": "x86_64",
...
      "install": "docker run --rm --privileged -v /:/host -e HOST=/host \
        -e IMAGE=IMAGE -e NAME=NAME IMAGE /bin/install.sh",

...
      "run": "docker run -d --privileged --name NAME --net=host --pid=host \
        -v /etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf \
        -v /etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d \
        -v /var/log:/var/log -v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run \
        -v /etc/machine-id:/etc/machine-id -v /etc/localtime:/etc/localtime \
        -e IMAGE=IMAGE -e NAME=NAME --restart=always IMAGE /bin/rsyslog.sh",
      "summary": "A containerized version of the rsyslog utility
...
```

The ubi7/ubi container will execute the bash shell, if no other argument is given when you start it with **podman run**. If an Entrypoint were set, its value would be used instead of the Cmd value (and the value of Cmd would be used as an argument to the Entrypoint command).

In the second example, the rhel7/rsyslog container image has built-in **install** and **run** labels. Those labels give an indication of how the container is meant to be set up on the system (install) and executed (run). You would use the **podman** command instead of **docker**.

2. **Mount a container**: Using the **podman** command, mount an active container to further investigate its contents. This example runs and lists a running **rsyslog** container, then displays the mount point from which you can examine the contents of its file system:

```
# podman run -d registry.access.redhat.com/rhel7/rsyslog
# podman ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
1cc92aea398d ...rsyslog:latest /bin/rsyslog.sh 37 minutes ago Up 1 day ago myrsyslog
# podman mount 1cc92aea398d
/var/lib/containers/storage/overlay/65881e78.../merged
# ls /var/lib/containers/storage/overlay/65881e78*/merged
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr
var
```

After running the **podman mount** command, the contents of the container are accessible from the listed directory on the host. Use **ls** to explore the contents of the image.

3. **Check the image's package list** To check the packages installed in the container, tell the **rpm** command to examine the packages installed on the container's mount point:

```
# rpm -qa --root=/var/lib/containers/storage/overlay/65881e78.../merged
redhat-release-server-7.6-4.el7.x86_64
filesystem-3.2-25.el7.x86_64
basesystem-10.0-7.el7.noarch
ncurses-base-5.9-14.20130511.el7_4.noarch
glibc-common-2.17-260.el7.x86_64
nspr-4.19.0-1.el7_5.x86_64
libstdc++-4.8.5-36.el7.x86_64
```

#### 1.7.2.3. Inspecting remote images

To inspect a container image before you pull it to your system, you can use the **skopeo inspect** command. With **skopeo inspect**, you can display information about an image that resides in a remote container registry.

The following command inspects the **rhel-init** image from the Red Hat registry:

```
# skopeo inspect docker://registry.access.redhat.com/ubi7/ubi
  {
   "Name": "registry.access.redhat.com/ubi7/ubi",
   "Digest": "sha256:caf8d01ac73911f872d184a73a5e72a1eeb7bba733cbad13e8253b567d16899f",
   "RepoTags": [
       "latest",
       "7.6",
       "7.7"
   ],
   "Created": "2019-08-01T09:29:20.753891Z",
   "DockerVersion": "1.13.1",
   "Labels": {
       "architecture": "x86_64",
       "authoritative-source-url": "registry.access.redhat.com",
       "build-date": "2019-08-01T09:28:54.576292",
       "com.redhat.build-host": "cpt-1007.osbs.prod.upshift.rdu2.redhat.com",
       "com.redhat.component": "ubi7-container",
```

### 1.7.3. Tagging images

You can add names to images to make it more intuitive to understand what they contain. Tagging images can also be used to identify the target registry for which the image is intended. Using the **podman tag** command, you essentially add an alias to the image that can consist of several parts. Those parts can include:

*registryhost/username/NAME:tag*

You can add just *NAME* if you like. For example:

```
# podman tag 967cb403b7ee myrhel7
```

Using **podman tag**, the name **myrhel7** now also is attached to the ubi7/ubi image (image ID 967cb403b7ee) on your system. So you could run this container by name (myrhel7) or by image ID. Notice that without adding a :tag to the name, it was assigned :latest as the tag. You could have set the tag to 7.7 as follows:

```
# podman tag 967cb403b7ee myrhel7:7.7
```

To the beginning of the name, you can optionally add a user name and/or a registry name. The user name is actually the repository on Docker.io or other registry that relates to the user account that owns the repository. Here's an example of adding a user name:

```
# podman tag 967cb403b7ee jsmith/myrhel7
# podman images | grep 967cb403b7ee
localhost/jsmith/myrhel7            latest  967cb403b7ee   10 days ago  215 MB
```

```
localhost/myrhel7                    7.7     967cb403b7ee   10 days ago  215 MB
registry.access.redhat.com/ubi7/ubi  latest  967cb403b7ee   10 days ago  215 MB
```

Above, you can see all the image names assigned to the single image ID.

### 1.7.4. Saving and importing images

If you want to save a container image you created, you can use **podman save** to save the image to a tarball. After that, you can store it or send it to someone else, then reload the image later to reuse it. Here is an example of saving an image as a tarball:

```
# podman save -o myrsyslog.tar registry.access.redhat.com/rhel7/rsyslog
# ls
myrsyslog.tar
```

The **myrsyslog.tar** file should now be stored in your current directory. Later, when you are ready to reuse the tarball as a container image, you can import it to another podman environment as follows:

```
# cat myrsyslog.tar | podman import - rhel7/myrsyslog
Getting image source signatures
Copying blob baa75547a1df done
Copying config 6722efbc0c done
Writing manifest to image destination
Storing signatures
6722efbc0ce5591161f773ef6371390965dc54212ac710c7517ee6d55eee6485
# podman images | grep myrsyslog
docker.io/rhel7/myrsyslog   latest  6722efbc0ce5  50 seconds ago   236 MB
```

### 1.7.5. Removing Images

To see a list of images that are on your system, run the **podman images** command. To remove images you no longer need, use the **podman rmi** command, with the image ID or name as an option. (You must stop any containers run from an image before you can remove the image.) Here is an example:

```
# podman rmi rhel-init
7e85c34f126351ccb9d24e492488ba7e49820be08fe53bee02301226f2773293
```

You can remove multiple images on the same command line:

```
# podman rmi registry.access.redhat.com/rhel7/rsyslog support-tools
46da8e23fa1461b658f9276191b4f473f366759a6c840805ed0c9ff694aa7c2f
85cfba5cd49c84786c773a9f66b8d6fca04582d5d7b921a308f04bb8ec071205
```

If you want to clear out all your images, you could use a command like the following to remove all images from your local registry (make sure you mean it before you do this!):

```
# podman rmi $(podman images -a -q)
1ca061b47bd70141d11dcb2272dee0f9ea3f76e9afd71cd121a000f3f5423731
ed904b8f2d5c1b5502dea190977e066b4f76776b98f6d5aa1e389256d5212993
83508706ef1b603e511b1b19afcb5faab565053559942db5d00415fb1ee21e96
```

To remove images that have multiple names (tags) associated with them, you need to add the force option to remove them. For example:

```
# podman rmi $(podman images -a -q)
unable to delete eb205f07ce7d0bb63bfe5603ef8964648536963e2eee51a3ebddf6cfe62985f7 (must
force) - image is referred to in multiple tags
unable to delete eb205f07ce7d0bb63bfe5603ef8964648536963e2eee51a3ebddf6cfe62985f7 (must
force) - image is referred to in multiple tags

# podman rmi -f eb205f07ce7d
eb205f07ce7d0bb63bfe5603ef8964648536963e2eee51a3ebddf6cfe62985f7
```

## 1.8. WORKING WITH CONTAINERS

Containers represent a running or stopped process spawned from the files located in a decompressed container image. Tools for running containers and working with them are described in this section.

### 1.8.1. Running containers

When you execute a **podman run** command, you essentially spin up and create a new container from a container image. The command you pass on the **podman run** command line sees the inside the container as its running environment so, by default, very little can be seen of the host system. For example, by default, the running applications sees:

- The file system provided by the container image.
- A new process table from inside the container (no processes from the host can be seen).

If you want to make a directory from the host available to the container, map network ports from the container to the host, limit the amount of memory the container can use, or expand the CPU shares available to the container, you can do those things from the **podman run** command line. Here are some examples of **podman run** command lines that enable different features.

**EXAMPLE #1 (Run a quick command)**: This podman command runs the **cat /etc/os-release** command to see the type of operating system used as the basis for the container. After the container runs the command, the container exits and is deleted (**--rm**).

```
# podman run --rm registry.access.redhat.com/ubi7/ubi cat /etc/os-release
NAME="Red Hat Enterprise Linux Server"
VERSION="7.7 (Maipo)"
ID="rhel"
ID_LIKE="fedora"
VARIANT="Server"
VARIANT_ID="server"
VERSION_ID="7.7"
PRETTY_NAME="Red Hat Enterprise Linux Server 7.7 (Maipo)"
...
```

**EXAMPLE #2 (View the Dockerfile in the container)** This is another example of running a quick command to inspect the content of a container from the host. All layered images that Red Hat provides include the Dockerfile from which they are built in **/root/buildinfo**. In this case you do not need to mount any volumes from the host.

```
# podman run --rm registry.access.redhat.com/ubi7/ubi ls /root/buildinfo
Dockerfile-ubi7-7.7-99
```

Now you know what the Dockerfile is called, you can list its contents:

```
# podman run --rm registry.access.redhat.com/ubi7/ubi \
    cat /root/buildinfo/Dockerfile-ubi7-7.7-99
FROM sha256:94577870ec362083c6513cfadb00672557fc5dd360e67befde6c81b9b753d06e
RUN mv -f /etc/yum.repos.d/ubi.repo /tmp || :

MAINTAINER Red Hat, Inc.

LABEL com.redhat.component="ubi7-container"
LABEL name="ubi7"
LABEL version="7.7"
...
```

**EXAMPLE #3 (Run a shell inside the container)** Using a container to launch a bash shell lets you look inside the container and change the contents. This sets the name of the container to **mybash**. The **-i** creates an interactive session and **-t** opens a terminal session. Without **-i**, the shell would open and then exit. Without **-t**, the shell would stay open, but you wouldn't be able to type anything to the shell.

Once you run the command, you are presented with a shell prompt and you can start running commands from inside the container:

```
# podman run --name=mybash -it registry.access.redhat.com/ubi7/ubi /bin/bash
[root@ed904b8f2d5c/]#  yum install nmap
[root@ed904b8f2d5c/]#  ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 00:46 pts/0    00:00:00 /bin/bash
root        35     1  0 00:51 pts/0    00:00:00 ps -ef
[root@49830c4f9cc4/]# exit
```

Although the container is no longer running once you exit, the container still exists with the new software package still installed. Use **podman ps -a** to list the container:

```
# podman ps -a
CONTAINER ID IMAGE                COMMAND   CREATED        STATUS                PORTS NAMES
IS INFRA
1ca061b47bd7 .../ubi8/ubi:latest   /bin/bash 8 minutes ago  Exited 12 seconds ago       musing_brown
false
...
```

You could start that container again using **podman start** with the **-ai** options. For example:

```
# podman start -ai mybash
[root@ed904b8f2d5c/]#
```

**EXAMPLE #4 (Bind mounting log files)** One way to make log messages from inside a container available to the host system is to bind mount the host's /dev/log device inside the container. This example illustrates how to run an application in a RHEL container that is named **log_test** that generates log messages (just the logger command in this case) and directs those messages to the /dev/log device that is mounted in the container from the host. The **--rm** option removes the container after it runs.

```
# podman run --name="log_test" -v /dev/log:/dev/log --rm \
    registry.access.redhat.com/ubi7/ubi logger "Testing logging to the host"
# journalctl -b | grep Testing
Aug 11 18:22:48 rhel76_01 root[6237]: Testing logging to the host
```

### 1.8.2. Investigating running and stopped containers

After you have some running container, you can list both those containers that are still running and those that have exited or stopped with the **podman ps** command. You can also use the **podman inspect** to look at specific pieces of information within those containers.

#### 1.8.2.1. Listing containers

Let's say you have one or more containers running on your host. To work with containers from the host system, you can open a shell and try some of the following commands.

**podman ps**: The ps option shows all containers that are currently running:

```
# podman run -d registry.access.redhat.com/rhel7/rsyslog
# podman ps
CONTAINER ID IMAGE          COMMAND         CREATED       STATUS            PORTS NAMES
50e5021715e4 rsyslog:latest /bin/rsyslog.sh 5 seconds ago Up 3 seconds ago  epic_torvalds
```

If there are containers that are not running, but were not removed (--rm option), the containers are still hanging around and can be restarted. The **podman ps -a** command shows all containers, running or stopped.

```
# podman ps -a
CONTAINER ID IMAGE           COMMAND    CREATED        STATUS                    PORTS NAMES
86a6f6962042 ubi7/ubi:latest /bin/bash  11 minutes ago Exited (0) 6 minutes ago       mybash
```

#### 1.8.2.2. Inspecting containers

To inspect the metadata of an existing container, use the **podman inspect** command. You can show all metadata or just selected metadata for the container. For example, to show all metadata for a selected container, type:

```
# podman inspect 50e5021715e4
    ...
[
    {
        "Id": "50e5021715e4829a3a37255145056ba0dc634892611a2f1d71c647cf9c9aa1d5",
        "Created": "2019-08-11T18:27:55.493059669-04:00",
        "Path": "/bin/rsyslog.sh",
        "Args": [
            "/bin/rsyslog.sh"
        ],
        "State": {
            "OciVersion": "1.0.1-dev",
            "Status": "running",
            "Running": true,
    ...
```

You can also use inspect to pull out particular pieces of information from a container. The information is stored in a hierarchy. So to see the container's IP address (IPAddress under NetworkSettings), use the **--format** option and the identity of the container. For example:

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' 50e5021715e4
10.88.0.36
```

Examples of other pieces of information you might want to inspect include .Path (to see the command run with the container), .Args (arguments to the command), .Config.ExposedPorts (TCP or UDP ports exposed from the container), .State.Pid (to see the process id of the container) and .HostConfig.PortBindings (port mapping from container to host). Here's an example of .State.Pid and .State.StartedAt:

```
# podman inspect --format='{{.State.Pid}}' 50e5021715e4
7544
# ps -ef | grep 7544
root     7544 7531  0 10:30 ?        00:00:00 /usr/sbin/rsyslogd -n
# podman inspect --format='{{.State.StartedAt}}' 50e5021715e4
2019-08-11 18:27:57.946930227 -0400 EDT
```

In the first example, you can see the process ID of the containerized executable on the host system (PID 7544). The **ps -ef** command confirms that it is the rsyslogd daemon running. The second example shows the date and time that the container was run.

#### 1.8.2.3. Investigating within a container

To investigate within a running container, you can use the **podman exec** command. With **podman exec**, you can run a command (such as **/bin/bash**) to enter a running container process to investigate that container.

The reason for using **podman exec**, instead of just launching the container into a bash shell, is that you can investigate the container as it is running its intended application. By attaching to the container as it is performing its intended task, you get a better view of what the container actually does, without necessarily interrupting the container's activity.

Here is an example using **podman exec** to look into a running rsyslog, then look around inside that container.

1. **Launch a container**: Launch a container such the rsyslog container image described earlier. Type **podman ps** to make sure it is running:

```
# podman ps
CONTAINER ID   IMAGE          COMMAND          CREATED      STATUS       PORTS
NAMES
50e5021715e4   rsyslog:latest "/usr/rsyslog.sh   6 minutes ago Up 6 minutes          rsyslog
```

2. Enter the container with **podman exec**: Use the container ID or name to open a bash shell to access the running container. Then you can investigate the attributes of the container as follows:

```
# podman exec -it 50e5021715e4 /bin/bash
[root@50e5021715e4 /]# cat /etc/redhat-release
Red Hat Enterprise Linux release 7.7 (Maipo)
[root@50e5021715e4 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 15:30 ?        00:00:00 /usr/sbin/rsyslogd -n
root         8     0  6 16:01 pts/0    00:00:00 /bin/bash
root        21     8  0 16:01 pts/0    00:00:00 ps -ef
[root@50e5021715e4 /]# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          39G  2.5G   37G   7% /
tmpfs            64M     0   64M   0% /dev
```

```
tmpfs           1.5G  8.7M  1.5G   1% /etc/hosts
shm              63M     0   63M   0% /dev/shm
tmpfs           1.5G     0  1.5G   0% /sys/fs/cgroup
tmpfs           1.5G     0  1.5G   0% /proc/acpi
tmpfs           1.5G     0  1.5G   0% /proc/scsi
tmpfs           1.5G     0  1.5G   0% /sys/firmware
tmpfs           1.5G     0  1.5G   0% /sys/fs/selinux
[root@50e5021715e4 /]# uname -r
3.10.0-957.27.2.el7.x86_64
[root@50e5021715e4 /]# rpm -qa | more
tzdata-2019b-1.el7.noarch
setup-2.8.71-10.el7.noarch
basesystem-10.0-7.el7.noarch
ncurses-base-5.9-14.20130511.el7_4.noarch
...
bash-4.2# free -m
              total        used        free      shared  buff/cache   available
Mem:           7792        2305        1712          18        3774        5170
Swap:          2047           0        2047

[root@50e5021715e4 /]# exit
```

The commands just run from the bash shell (running inside the container) show you several things.

- The container was built from a RHEL release 7.7 image.
- The process table (ps -ef) shows that the /usr/sbin/rsyslogd command is process ID 1.
- Processes running in the host's process table cannot be seen from within the container. Although the rsyslogd process can be seen on the host process table (it was process ID 7544 on the host).
- There is no separate kernel running in the container (uname -r shows the host system's kernel).
- The rpm -qa command lets you see the RPM packages that are included inside the container. In other words, there is an RPM database inside of the container.
- Viewing memory (free -m) shows the available memory on the host (although what the container can actually use can be limited using cgroups).

### 1.8.3. Starting and stopping containers

If you ran a container, but didn't remove it (--rm), that container is stored on your local system and ready to run again. To start a previously run container that wasn't removed, use the start option. To stop a running container, use the stop option.

#### 1.8.3.1. Starting containers

A container that doesn't need to run interactively can sometimes be restarted after being stopped with only the **start** option and the container ID or name. For example:

```
# podman start myrhel_httpd
myrhel_httpd
```

To start a container so you can work with it from the local shell, use the -a (attach) and -i (interactive) options. Once the bash shell starts, run the commands you want inside the container and type exit to kill the shell and stop the container.

```
# podman start -a -i agitated_hopper
[root@d65aecc325a4 /]#  exit
```

#### 1.8.3.2. Stopping containers

To stop a running container that is not attached to a terminal session, use the stop option and the container ID or number. For example:

```
# podman stop 74b1da000a11
74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2
```

The **stop** option sends a SIGTERM signal to terminate a running container. If the container doesn't stop after a grace period (10 seconds by default), **podman** sends a SIGKILL signal. You could also use the **podman kill** command to kill a container (SIGKILL) or send a different signal to a container. Here's an example of sending a SIGHUP signal to a container (if supported by the application, a SIGHUP causes the application to re-read its configuration files):

```
# podman kill --signal="SIGHUP" 74b1da000a11
74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2
```

### 1.8.4. Removing containers

To see a list of containers that are still hanging around your system, run the **podman ps -a** command. To remove containers you no longer need, use the **podman rm** command, with the container ID or name as an option. Here is an example:

```
# podman rm goofy_wozniak
```

You can remove multiple containers on the same command line:

```
# podman rm clever_yonath furious_shockley drunk_newton
```

If you want to clear out all your containers, you could use a command like the following to remove all containers (not images) from your local system (make sure you mean it before you do this!):

```
# podman rm $(podman ps -a -q)
```

# CHAPTER 2. USING RED HAT UNIVERSAL BASE IMAGES (STANDARD, MINIMAL, AND RUNTIMES)

Red Hat Enterprise Linux (RHEL) base images are meant to form the foundation for the container images you build. As of April 2019, new Universal Base Image (UBI) versions of RHEL standard, minimal, init, and Red Hat Software Collections images are available that add to those images the ability to be freely redistributed. Characteristics of RHEL base images include:

- **Supported**: Supported by Red Hat for use with your containerized applications. Contains the same secured, tested, and certified software packages you have in Red Hat Enterprise Linux.
- **Cataloged**: Listed in the Red Hat Container Catalog , where you can find descriptions, technical details, and a health index for each image.
- **Updated**: Offered with a well-defined update schedule, so you know you are getting the latest software (see Red Hat Container Image Updates ).
- **Tracked**: Tracked by errata, to help you understand the changes that go into each update.
- **Reusable**: Only need to be downloaded and cached in your production environment once, where each base image can be reused by all containers that include it as their foundation.

Red Hat Universal Base Images (UBI) provide the same quality RHEL software for building container images as their predecessors (**rhel6**, **rhel7**, **rhel-init**, and **rhel-minimal** base images), but offer more freedom in how they are used and distributed.

## 2.1. WHAT ARE RED HAT BASE IMAGES?

Red Hat provides multiple base images that you can use as a starting point for your own images. These images are available through the Red Hat Registry (registry.access.redhat.com and registry.redhat.io) and described in the Red Hat Container Catalog .

For RHEL 7, there are two different versions of each standard, minimal and init base image available. Red Hat also provides a set of Red Hat Software Collections images that you can build on when you are creating containers for applications that require specific runtimes. These include python, php, nodejs, and others.

Although Red Hat does not offer tools for running containers on RHEL 6 systems, it does offer RHEL 6 container images you can use. There are standard (**rhel6**) and Init (**rhel6-init**) base image available for RHEL 6, but no minimal RHEL 6 image. Likewise, there are no RHEL 6 UBI images.

### 2.1.1. Using standard Red Hat base images

There is a legacy **rhel7**/**rhel** image and a UBI **ubi7** image on which you can add your own software or additional RHEL 7 software. The contents are nearly identical, with the main differences that the former requires a RHEL paid subscription and the two images draw from different image registries and yum repositories.

Standard RHEL base images have a robust set of software features that include the following:

- **init system**: All the features of the systemd initialization system you need to manage systemd services are available in the standard base images. The **rhel6** base images include a minimalistic init system similar to System V. These init systems let you install RPM packages that are pre-configured to start up services automatically, such as a Web server (httpd) or FTP server (vsftpd).

- **yum**: Software needed to install software packages is included via the standard set of **yum** commands (**yum**, **yum-config-manager**, **yumdownloader**, and so on). When a legacy standard base image is run on a RHEL system, you will be able to enable repositories and add packages as you do directly on a RHEL system, while using entitlements available on the host. For the UBI base images, you have access to free yum repositories for adding and updating software.

- **utilities**: The standard base image includes some useful utilities for working inside the container. Utilities that are in this base image that are not in the minimal images include **ps**, **tar**, **cpio**, **dmidecode**, **gzip**, **lsmod** (and other module commands), **getfacl** (and other acl commands), **dmsetup** (and other device mapper commands), and others.

- **python**: Python runtime libraries and modules (currently Python 2.7) are included in the standard base image. No python packages are included in the minimal base image.

### 2.1.2. Using minimal Red Hat base images

The legacy **rhel7-minimal** (or **rhel7-atomic**) and UBI **ubi7-minimal** images are stripped-down RHEL images to use when a bare-bones base image in desired. If you are looking for the smallest possible base image to use as part of the larger Red Hat ecosystem, you can start with these minimal images.

RHEL minimal images provide a base for your own container images that is less than half the size of the standard image, while still being able to draw on RHEL software repositories and maintain any compliance requirements your software has.

Here are some features of the minimal base images:

- **Small size**: Minimal images are about 75M on disk and 28M compressed. This makes it less than half the size of the standard images.

- **Software installation (microdnf)**: Instead of including the full-blown yum facility for working with software repositories and RPM software packages, the minimal images includes the microdnf utility. Microdnf is a scaled-down version of dnf. It includes only what is needed to enable and disable repositories, as well as install, remove, and update packages. It also has a clean option, to clean out cache after packages have been installed.

- **Based on RHEL packaging**: Because minimal images incorporate regular RHEL software RPM packages, with a few features removed such as extra language files or documentation, you can continue to rely on RHEL repositories for building your images. This allows you to still maintain compliance requirements you have that are based on RHEL software. Features of minimal images make them perfect for trying out applications you want to run with RHEL, while carrying the smallest possible amount of overhead.

If your goal is just to try to run some simple binaries or pre-packaged software that doesn't have a lot of requirements from the operating system, the minimal images might suit your needs. If your application does have dependencies on other software from RHEL, you can simply use microdnf to install the needed packages at build time.

Here are some challenges related to using minimal images:

- **Common utilities missing**: What you don't get with minimal images is an initialization and service management system (systemd or System V init), a Python run-time environment, and a bunch of common shell utilities. Although you can install that software later, installing a large set of software, such as systemd, might actually make the container larger than it would be if you were to just use the init container.

- **Older minimal images not supported**: Red Hat intends for you to always use the latest version of the minimal images, which is implied by simply requesting **rhel-minimal** or **ubi7-minimal**. Red Hat does not expect to support older versions of minimal images going forward.

- **Modules for microdnf are not supported**: Modules used with the **dnf** command let you install multiple versions of the same software, when available. The **microdnf** utility included with minimal images does not support modules. So if modules are required, you should use a non-minimal image (such as the standard or init UBI images, which both include yum).

### 2.1.3. Using Init Red Hat base images

The legacy **rhel7-init** and UBI **ubi7-init** images contains the systemd initialization system, making them useful for building images in which you want to run systemd services, such as a web server or file server. The Init image contents are less than what you get with the standard images, but more than what is in the minimal images.

Historically, Red Hat Enterprise Linux base container images were designed for Red Hat customers to run enterprise applications, but were not free to redistribute. This can create challenges for some organizations that need to redistribute their applications. That's where the Red Hat Universal Base Images come in.

## 2.2. HOW ARE UBI IMAGES DIFFERENT?

UBI images were created so you can build your container images on a foundation of official Red Hat software that can be freely shared and deployed. From a technical perspective, they are nearly identical to legacy Red Hat Enterprise Linux images, which means they have great security, performance, and life cycles, but they are released under a different End User License Agreement. Here are some attributes of Red Hat UBI images:

- **Built from a subset of RHEL content** Red Hat Universal Base images are built from a subset of normal Red Hat Enterprise Linux content. All of the content used to build selected UBI images is released in a publicly available set of yum repositories. This lets you install extra packages, as well as update any package in UBI base images.

- **Redistributable**: The intent of UBI images is to allow Red Hat customers, partners, ISVs, and others to standardize on one container base image, allowing users to focus on application needs instead of distribution rules. These images can be shared and run in any environment capable of running those images. As long as you follow some basic guidelines, you will be able to freely redistribute your UBI-based images.

- **Base and RHSCL images** Besides the three types of base images, UBI versions of some Red Hat Software Collections (RHSCL) runtime images are available as well. These RHSCL images provide a foundation for applications that can benefit from standard, supported runtimes such as python, php, nodejs, and ruby.

- **Enabled yum repositories**: The following yum repositories are enabled within each RHEL 7 UBI image:

  - The **ubi-7** repo holds the redistributable subset of RHEL packages you can include in your container.

  - The **ubi-7-rhscl** repo holds Red Hat Software Collections packages that you can add to a UBI image to help you standardize the environments you use with applications that require particular runtimes.

  - The **ubi-7-rhah** repo includes RHEL Atomic Host packages needed to manage

subscriptions and **microdnf** (the tiny **yum** replacement used to install RPM packages on the minimal images). Note that some versions of ubi-minimal images do not have this repo enabled by default.

  - The **ubi-7-optional** repo includes packages from the RHEL server optional repository.

- **Licensing**: You are free to use and redistribute UBI images, provided you adhere to the Red Hat Universal Base Image End User Licensing Agreement.
- **Adding UBI RPMs**: You can add RPM packages to UBI images from preconfigured UBI repositories. If you happen to be in a disconnected environment, you must whitelist the UBI Content Delivery Network (https://cdn-ubi.redhat.com) to use that feature. See the Connect to https://cdn-ubi.redhat.com solution for details.

Although the legacy RHEL base images will continue to be supported, UBI images are recommended going forward. For that reason, examples in the rest of this chapter are done with UBI images.

## 2.3. GET UBI IMAGES

To find the current set of available Red Hat UBI images, refer to Universal Base Images (UBI): Images, repositories, and packages or search the Red Hat Container Catalog .

## 2.4. PULL UBI IMAGES

To pull UBI images to your system so you can use them with tools such as podman, buildah or skopeo, type the following:

```
# podman pull registry.access.redhat.com/ubi7/ubi:latest
# podman pull registry.access.redhat.com/ubi7/ubi-minimal:latest
```

To check that the images are available on your system, type:

```
# podman images
REPOSITORY                                  TAG    IMAGE ID      CREATED      SIZE
registry.access.redhat.com/ubi7/ubi-minimal  latest  c94a444803e3  8 hours ago  80.9 MB
registry.access.redhat.com/ubi7/ubi          latest  40b488f87628  17 hours ago 214 MB
```

When pulled in this way, images are available and usable by **podman**, **buildah**, **skopeo** and the CRI-O container image, but they are not available to the Docker service or **docker** command. To use these images with Docker, you can run **docker pull** instead.

## 2.5. REDISTRIBUTING UBI IMAGES

After you pull a UBI image, you are free to push it to your own registry and share it with others. You can upgrade or add to that image from UBI yum repositories as you like. Here is an example of how to push a UBI image to your own or another third-party repository:

```
# podman pull registry.redhat.io/ubi7/ubi
# podman tag registry.access.redhat.com/ubi7/ubi registry.example.com:5000/ubi7/ubi
# podman push registry.example.com:5000/ubi7/ubi
```

While there are few restrictions on how you use this image, there are some restrictions about how you can refer to it. For example, you can't call that image Red Hat certified or Red Hat supported unless you

certify it through the Red Hat Partner Connect Program , either with Red Hat Container Certification or Red Hat OpenShift Operator Certification.

## 2.6. RUN UBI IMAGES

To start a container from a UBI image and run the bash shell in that image (so you can look around inside), do the following (type exit when you are done):

```
# podman run --rm -it registry.access.redhat.com/ubi7/ubi-minimal:latest /bin/bash
[root@da9213157c51 /]#
# podman run --rm -it registry.access.redhat.com/ubi7/ubi:latest /bin/bash
bash-4.2#
```

While in the container:

- Run **rpm -qa** to see a list of package inside each container.
- Type **yum list available** to see packages available to add to the image from the UBI yum repos. (The yum command is not available in the **ubi-minimal** containers.)
- Get source code, as described in the "Getting UBI Container Image Source Code," later in this chapter.

On systems that include the Docker service, you can use **docker run** instead.

## 2.7. ADD SOFTWARE TO A RUNNING UBI CONTAINER

UBI images are built from 100% Red Hat content. These UBI images also provide a subset of Red Hat Enterprise Linux packages which are freely available to install for use with UBI. To add or update software, UBI images are pre-configured to point to the freely available yum repositories that hold official Red Hat RPMs.

To add packages from UBI repos to running UBI containers:

- On **ubi** images, the yum command is installed to let you draw packages
- On **ubi-minimal** images, the **microdnf** command (with a smaller feature set) is included instead of **yum**.

Keep in mind that installing and working with software packages directly in running containers is just for adding packages temporarily or learning about the repos. Refer to the “Build a UBI-based image” for more permanent ways of building UBI-based images.

When you add software to a UBI container, procedures differ for updating UBI images on a subscribed RHEL host or on an unsubscribed (or non-RHEL) system. Those two ways of working with UBI images are illustrated below.

### 2.7.1. Adding software to a UBI container (subscribed host)

If you are running a UBI container on a registered and subscribed RHEL host, the main RHEL Server repository is enabled inside the standard UBI container, along with all the UBI repos. So the full set of Red Hat packages is available. From the UBI minimal container, All UBI repos are enabled by default, but no repos are enabled from the host by default.

### 2.7.2. Adding software inside the standard UBI container

To ensure the containers you build can be redistributed, disable subscription management in the standard UBI image when you add software. If you disable the subscription-manager plugin, only packages from the freely available repos are used when you add software.

With a shell open inside a standard UBI base image container (**ubi7/ubi**) from a subscribed RHEL host, run the following command to add a package to that container (for example, the bzip2 package):

```
# yum install --disableplugin=subscription-manager bzip2
```

To add software inside a standard UBI container that is in the RHEL server repo, but not in UBI repos, leave the subscription-manager plugin intact and just install the package:

```
# yum install zsh
```

To install a package that is in a different host repo from inside the standard UBI container, you have to explicitly enable the repo you need. For example:

```
# yum install --enablerepo=rhel-7-server-optional-rpms zsh-html
```

**WARNING**

Installing Red Hat packages that are not inside the Red Hat UBI repos might limit how widely you can distribute the container outside of subscribed hosts.

### 2.7.3. Adding software inside the minimal UBI container

UBI yum repositories are enabled inside the UBI minimal image by default.

To install the same package demonstrated earlier (bzip2) from one of those UBI yum repositories on a subscribed RHEL host from the UBI minimal container, type:

```
# microdnf install bzip2
```

To install packages inside a minimal UBI container from repos available on a subscribed host that are not part of a UBI yum repo, you would have to explicitly enable those repos. For example:

```
# microdnf install --enablerepo=rhel-7-server-rpms zsh
# microdnf install --enablerepo=rhel-7-server-rpms \
      --enablerepo=rhel-7-server-optional-rpms zsh-html
```

**WARNING**

Using non-UBI RHEL repositories to install packages in your UBI images could restrict your ability to share those images to run outside of subscribed RHEL systems.

### 2.7.4. Adding software to a UBI container (unsubscribed host)

To add software packages to a running container that is either on an unsubscribed RHEL host or some other Linux system, you don't have to disable the subscription-manager plugin. For example:

```
# yum install bzip2
```

To install that package on a subscribed RHEL host from the UBI minimal container, type:

```
# microdnf install bzip2
```

As noted earlier, both of these means of adding software to a running UBI container are not intended for creating permanent UBI-based container images. For that, you should build new layers on to UBI images, as described in the following section.

## 2.8. BUILD A UBI-BASED IMAGE

You can build UBI-based container images in the same way you build other images, with one exception. You should disable Red Hat subscriptions when you actually build the images, if you want to be sure that your image only contains Red Hat software that you can redistribute.

Here's an example of creating a UBI-based Web server container from a Dockerfile with the **buildah** utility:

**NOTE**

For ubl7/ubi-minimal images, use microdnf instead of yum below:

```
RUN microdnf update -y && rm -rf /var/cache/yum
RUN microdnf install httpd -y && microdnf clean all
```

1. **Create a Dockerfile**: Add a **Dockerfile** with the following contents to a new directory:

```
FROM registry.access.redhat.com/ubi7/ubi
USER root
LABEL maintainer="John Doe"
# Update image
RUN yum update --disableplugin=subscription-manager -y && rm -rf /var/cache/yum
RUN yum install --disableplugin=subscription-manager httpd -y && rm -rf /var/cache/yum
# Add default Web page and expose port
RUN echo "The Web Server is Running" > /var/www/html/index.html
EXPOSE 80
```

```
# Start the service
CMD ["-D", "FOREGROUND"]
ENTRYPOINT ["/usr/sbin/httpd"]
```

2. **Build the new image** While in that directory, use **buildah** to create a new UBI layered image:

```
# buildah bud -t johndoe/webserver .
STEP 1: FROM registry.access.redhat.com/ubi7/ubi:latest
STEP 2: USER root
STEP 3: MAINTAINER John Doe
STEP 4: RUN yum update --disableplugin=subscription-manager -y
. . .
No packages marked for update
STEP 5: RUN yum install --disableplugin=subscription-manager httpd -y
Loaded plugins: ovl, product-id, search-disabled-repos
Resolving Dependencies
--> Running transaction check
================================================================
 Package              Arch          Version                  Repository              Size
================================================================
Installing:
 httpd                x86_64        2.4.6-88.el7             ubi-7          1.2 M
Installing for dependencies:
 apr                  x86_64        1.4.8-3.el7_4.1                ubi-7          103 k
 apr-util             x86_64        1.5.2-6.el7              ubi-7          92 k
 httpd-tools          x86_64        2.4.6-88.el7               ubi-7          90 k
 mailcap              noarch        2.1.41-2.el7             ubi-7          31 k
 redhat-logos         noarch          70.0.3-7.el7               ubi-7          13 M

Transaction Summary
...
Complete!
STEP 6: RUN echo "The Web Server is Running" > /var/www/html/index.html
STEP 7: EXPOSE 80
STEP 8: CMD ["-D", "FOREGROUND"]
STEP 9: ENTRYPOINT ["/usr/sbin/httpd"]
STEP 10: COMMIT
...
Writing manifest to image destination
Storing signatures
--> 36a604cc0dd3657b46f8762d7ef69873f65e16343b54c63096e636c80f0d68c7
```

3. **Test**: Test the UBI layered webserver image:

```
# podman run -d -p 80:80 johndoe/webserver
bbe98c71d18720d966e4567949888dc4fb86eec7d304e785d5177168a5965f64
# curl http://localhost/index.html
The Web Server is Running
```

## 2.9. USING RED HAT SOFTWARE COLLECTIONS RUNTIME IMAGES

Red Hat Software Collections offers another set of container images that you can use as the basis for your container builds. These images are built on RHEL standard base images, with some already updated as UBI images. Each of these images include additional software you might want to use for

specific runtime environments.

So, if you expect to build multiple images that require, for example, php runtime software, you can use provide a more consistent platform for those images by starting with a PHP software collections image.

Here are examples of Red Hat Software Collections container images built on UBI base images, that are available from the Red Hat Registry (registry.access.redhat.com or registry.redhat.io):

- **ubi7/php-72**: PHP 7.2 platform for building and running applications
- **ubi7/nodejs-8**: Node.js 8 platform for building and running applications. Used by Node.js 8 Source-To-Image builds
- **ubi7/ruby-25**: Ruby 2.5 platform for building and running applications
- **ubi7/python-27**: Python 2.7 platform for building and running applications
- **ubi7/python-36**: Python 3.6 platform for building and running applications
- **ubi7/s2i-core**: Base image with essential libraries and tools used as a base for builder images like perl, python, ruby, and so on
- **ubi7/s2i-base**: Base image for Source-to-Image builds

Because these UBI images container the same basic software as their legacy image counterparts, you can learn about those images from the Using Red Hat Software Collections Container Images guide. Be sure to use the UBI image names to pull those images.

Red Hat Software Collections container images are updated every time RHEL base images are updated. Search the Red Hat Container Catalog for details on any of these images. For more information on update schedules, see Red Hat Container Image Updates .

## 2.10. GETTING UBI CONTAINER IMAGE SOURCE CODE

You can download the source code for all UBI base images (excluding the minimal images) by starting up those images with a bash shell and running the following set of commands from inside that container:

```
for i in `rpm -qa`
   do
   yumdownloader --source $i
done
```

The source code RPM for each binary RPM package is downloaded to the current directory. Because the UBI minimal images include a subset of RPMs from the regular UBI images, running the **yumdownloader** loop just shown will get you the minimal image packages as well.

## 2.11. TIPS AND TRICKS FOR USING UBI IMAGES

Here are a few issues to consider when working with UBI images:

- Hundreds of RPM packages used in existing Red Hat Software Collections runtime images are stored in the yum repositories packaged with the new UBI images. Feel free to install those RPMs on your UBI images to emulate the runtime (python, php, nodejs, etc.) that interests you.
- Because some language files and documentation have been stripped out of the minimal UBI

image (**ubi7**/**ubi-minimal**), running rpm -Va inside that container will show the contents of many packages as being missing or modified. If having a complete list of files inside that container is important to you, consider using a tool such as Tripwire to record the files in the container and check it later.

- After a layered image has been created, use **podman history** to check which UBI image it was built on. For example, after completing the webserver example shown earlier, type **podman history johndoe/webserver** to see that the image it was built on includes the image ID of the UBI image you added on the FROM line of the Dockerfile.

## 2.12. HOW TO REQUEST NEW FEATURES IN UBI?

Red Hat partners and customers can request new features, including package requests, by filing a support ticket through standard methods. Non-Red Hat customers do not receive support, but can file requests through the standard Red Hat Bugzilla for the appropriate RHEL product. See also: Red Hat Bugzilla Queue

## 2.13. HOW TO FILE A SUPPORT CASE FOR UBI?

Red Hat partners and customers can file support tickets through standard methods when running UBI on a supported Red Hat platform (OpenShift/RHEL). Red Hat support staff will guide partners and customers

See also: Open a Support Case

# CHAPTER 3. INSTALL AND DEPLOY AN APACHE WEB SERVER CONTAINER

## 3.1. OVERVIEW

A Web server is one of the most basic examples used to illustrate how containers work. The procedure in this topic does the following:

- Builds an Apache (httpd) Web server inside a container
- Exposes the service on port 80 of the host
- Serves a simple index.html file
- Displays data from a backend server (needs additional MariaDB container described later)

## 3.2. CREATING AND RUNNING THE APACHE WEB SERVER CONTAINER

1. **Install system**: Install a RHEL 7 or RHEL Atomic system that includes the docker package and start the docker service.
2. **Pull image**: Pull the rhel7 image by typing the following:

```
# docker pull rhel7:latest
```

3. **Create Directory to hold Dockerfile** Create a directory (named mywebcontainer) that will hold a file names **Dockerfile** and another named **action**.

```
# mkdir ~/mywebcontainer
# cd ~/mywebcontainer
# touch action Dockerfile
```

4. **Create action CGI script** Create the **action** file in the **~/mywebcontainer** directory, which will be used to get data from the backend database server container. This script assumes that the **docker0** interface on the host system is at IP address **172.17.42.1**, you can login to the database with the **dbuser1** user account and **redhat** as the password, and use the database named **gss**. If that is the IP address and you use the database container described later, you don't need to modify this script. (You can also just ignore this script and just use the Web server to get HTML content.)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import os

con = mdb.connect(os.getenv('DB_SERVICE_SERVICE_HOST','172.17.42.1'), 'dbuser1',
'redhat', 'gss')

with con:

    cur = con.cursor()
    cur.execute("SELECT MESSAGE FROM atomic_training")
```

```
    rows = cur.fetchall()

    print 'Content-type:text/html\r\n\r\n'
    print '<html>'
    print '<head>'
    print '<title>My Application</title>'
    print '</head>'
    print '<body>'

    for row in rows:
        print '<h2>' + row[0] + '</h2>'

    print '</body>'
    print '</html>'

    con.close()
```

5. **Check the Dockerfile** Create the Dockerfile file in the **~/mywebcontainer** directory as needed (perhaps only modify Maintainer_Name to add your name). Here are the contents of that file:

```
# Webserver container with CGI python script
# Using RHEL 7 base image and Apache Web server
# Version 1

# Pull the rhel image from the local registry
FROM rhel7:latest
USER root

MAINTAINER Maintainer_Name

# Fix per https://bugzilla.redhat.com/show_bug.cgi?id=1192200
RUN yum -y install deltarpm yum-utils --disablerepo=*-eus-* --disablerepo=*-htb-* *-sjis-*\
    --disablerepo=*-ha-* --disablerepo=*-rt-* --disablerepo=*-lb-* --disablerepo=*-rs-* --
disablerepo=*-sap-*

RUN yum-config-manager --disable *-eus-* *-htb-* *-ha-* *-rt-* *-lb-* *-rs-* *-sap-* *-sjis* >
/dev/null

# Update image
RUN yum install httpd procps-ng MySQL-python -y

# Add configuration file
ADD action /var/www/cgi-bin/action
RUN echo "PassEnv DB_SERVICE_SERVICE_HOST" >> /etc/httpd/conf/httpd.conf
RUN chown root:apache /var/www/cgi-bin/action
RUN chmod 755 /var/www/cgi-bin/action
RUN echo "The Web Server is Running" > /var/www/html/index.html
EXPOSE 80

# Start the service
CMD mkdir /run/httpd ; /usr/sbin/httpd -D FOREGROUND
```

6. **Build Web server container:**From the directory containing the Dockerfile file and other content, type the following:

```
# docker build -t webwithdb .
Sending build context to Docker daemon 4.096 kB
Sending build context to Docker daemon
Step 0 : FROM rhel7:latest
 ---> bef54b8f8a2f
Step 1 : USER root
 ---> Running in 00c28d347131
 ---> cd7ef0fcaf55
...
```

7. **Start the Web server container:**To start the container image, run the following command:

```
# docker run -d -p 80:80 --name=mywebwithdb webwithdb
```

8. **Test the Web server container:**To check that the Web server is operational, run the first curl command below. If you have the backend database container running, try the second command:

```
# curl http://localhost/index.html
The Web Server is Running
# curl http://localhost/cgi-bin/action
<html>
<head>
<title>My Application</title>
</head>
<body>
<h2>RedHat rocks</h2>
<h2>Success</h2>
</body>
</html>
</tt></pre>
```

If you have a Web browser installed on the localhost, you can open a Web browser to see as better representation of the few lines of output. Just open the browser to this URL: http://localhost/cgi-bin/action

## 3.3. TIPS FOR THIS CONTAINER

Here are some tips to help you use the Web Server container:

- **Modify for MariaDB:**To use this container with the MariaDB container (described later), you may need to edit the **action** script and change the IP address from 172.17.42.1 to the host IP on the docker0 interface. To find what that address is on your host, type the following:

```
# ip a | grep docker0 | grep inet
    inet 172.17.42.1/16 scope global docker0
```

- **Adding content:** You can include your own content, mounted from the local host, by using the -v option on the docker run command line. For example:

```
# docker run -d -p 80:80 -v /var/www/html:/var/www/html \
    --name=mywebwithdb webwithdb
```

## 3.4. ATTACHMENTS

- Apache Web container tar file: action CGI script and Dockerfile

# CHAPTER 4. INSTALL AND DEPLOY A MARIADB CONTAINER

## 4.1. OVERVIEW

Using MariaDB, you can set up a basic database in a container that can be accessed by other applications. The procedure in this topic does the following:

- Builds a MariaDB database server inside a docker formatted container
- Exposes the service on port 3306 of the host
- Starts up the database service to share a few pieces of information
- Allows a script from Web server to query the database (needs additional Web server container described later)
- Offers tips on how to use and extend this container

## 4.2. CREATING AND RUNNING THE MARIADB DATABASE SERVER CONTAINER

1. **Install system:** Install a Red Hat Enterprise Linux 7 or Red Hat Enterprise Linux Atomic Host system that includes the docker package and start the docker service.
2. **Pull image:** Pull the rhel7 image by typing the following:

```
# docker pull rhel7:latest
```

3. **Get tarball with supporting files**: Download the tarball file attached to this article (mariadb_cont_2.tgz), download it to a new mydbcontainer directory, and untar it as follows:

```
# mkdir ~/mydbcontainer
# cp mariadb_cont*.tgz ~/mydbcontainer
# cd ~/mydbcontainer
# tar xvf mariadb_cont*.tgz
gss_db.sql
Dockerfile
```

4. **Create the Dockerfile:** Create the Dockerfile file shown below in the **~/mydbcontainer** directory and modify it as needed (perhaps only modify Maintainer_Name to add your name). Here are the contents of that file:

```
# Database container with simple data for a Web application
# Using RHEL 7 base image and MariahDB database
# Version 1

# Pull the rhel image from the local repository
FROM rhel7:latest
USER root

MAINTAINER Maintainer_Name

# Update image
```

```
RUN yum update -y --disablerepo=*-eus-* --disablerepo=*-htb-* --disablerepo=*sjis* \
   --disablerepo=*-ha-* --disablerepo=*-rt-* --disablerepo=*-lb-* \
   --disablerepo=*-rs-* --disablerepo=*-sap-*

RUN yum-config-manager --disable *-eus-* *-htb-* *-ha-* *-rt-* *-lb-* \
   *-rs-* *-sap-* *-sjis-* > /dev/null

# Add Mariahdb software
RUN yum -y install net-tools mariadb-server

# Set up Mariahdb database
ADD gss_db.sql /tmp/gss_db.sql
RUN /usr/libexec/mariadb-prepare-db-dir
RUN test -d /var/run/mariadb || mkdir /var/run/mariadb; \
   chmod 0777 /var/run/mariadb; \
   /usr/bin/mysqld_safe --basedir=/usr & \
   sleep 10s && \
   /usr/bin/mysqladmin -u root password 'redhat' && \
   mysql --user=root --password=redhat < /tmp/gss_db.sql && \
   mysqladmin shutdown --password=redhat

# Expose Mysql port 3306
EXPOSE 3306

# Start the service
CMD test -d /var/run/mariadb || mkdir /var/run/mariadb; chmod 0777
/var/run/mariadb;/usr/bin/mysqld_safe --basedir=/usr
```

5. **Modify gss_db.sql:** Look at the **gss_db.sql** file in the **~/mydbcontainer** directory and modify it as needed:

6. **Build database server container:** From the directory containing the Dockerfile file and other content, type the following:

```
# docker build -t dbforweb .
Sending build context to Docker daemon 528.4 kB
Sending build context to Docker daemon
Step 0 : FROM rhel7:latest
 ---> bef54b8f8a2f
Step 1 : USER root
...
```

7. **Start the database server container:** To start the container image, run the following command:

```
# docker run -d -p 3306:3306 --name=mydbforweb dbforweb
```

8. **Test the database server container:** Assuming the docker0 interface on the host is 172.17.42.1 (yours may be different), check that the database container is operational by running the **nc** command (in RHEL 7, type **yum install nc** to get it) as shown here:

```
# nc -v 172.17.42.1 3306
Ncat: Version 6.40 ( http://nmap.org/ncat )
Ncat: Connected to 172.17.42.1:3306.
R
5.5.44-MariaDB?acL3YF31?X?FWbiiTIO2Kd6mysql_native_password Ctrl-C
```

## 4.3. TIPS FOR THIS CONTAINER

Here are some tips to help you use the Web Server container:

- **Adding your own database:** You can include your own MariaDB content by copying your database file to the build directory and changing the name of the database file from gss_db.sql to the name of your database (in several places in the Dockerfile file).
- **Orchestrate containers:** A better way to manage this container with other containers is to use Kubernetes to orchestrate them into pods.

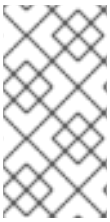
## 4.4. ATTACHMENTS

- Tar file containing gss_db.sql database and Dockerfile files for MariaDB container

# CHAPTER 5. USING THE DOCKER COMMAND AND SERVICE

## 5.1. OVERVIEW

The Docker project was responsible for popularizing container development in Linux systems. The original project defined a command and service (both named **docker**) and a format in which containers are structured. This chapter provides a hands-on approach to using the docker command and service to begin working with containers in Red Hat Enterprise Linux 7 and RHEL Atomic Host by getting and using container images and working with running containers.



**NOTE**

Keep in mind that the preferred tools for working with containers in Red Hat Enterprise Linux systems are podman, skopeo, buildah and related commands. In particular, podman supports many of the same command-line options available with the docker command. The docker command and service are not supported in Red Hat Enterprise Linux 8.

If you are interested in more details on how the docker command works, refer to the following:

- **Docker Project Site**: From the Docker site, you can learn about Docker from the What is Docker? page and the Getting Started page. There is also a Docker Documentation page you can refer to.
- **Docker man pages**: Again, with docker installed (RHEL only, not Atomic), type **man docker** to learn about the docker command. Then refer to separate man pages for each docker option (for example, type **man docker-image** to read about the **docker image** option).

**NOTE**

Currently, to run the docker command in RHEL 7 and RHEL Atomic Host you must have root privilege. In the procedure, this is indicated by the command prompt appearing as a hash sign (#). Configuring sudo will work, if you prefer not to log in directly to the root user account.

## 5.2. GETTING DOCKER IN RHEL 7

To get an environment where you can develop Docker containers, you can install a Red Hat Enterprise Linux 7 system to act as a development system as well as a container host. The docker package itself is stored in a RHEL Extras repository (see the Red Hat Enterprise Linux Extras Life Cycle article for a description of support policies and life cycle information for the Red Hat Enterprise Linux Extras channel).

Using the RHEL 7 subscription model, if you want to create container images, you must properly register and entitle the host computer on which you build them. When you use **yum install** within a container to add packages, the container automatically has access to entitlements available from the RHEL 7 host, so it can get RPM packages from any repository enabled on that host.

**NOTE:** The docker packages and other container-related packages were originally only available for the RHEL Server and RHEL Atomic Host editions. They are now available for RHEL 7 Workstation as well.

1. **Install RHEL Server edition**: If you are ready to begin, you can start by installing a Red Hat Enterprise Linux system (Server edition) as described in the following: Red Hat Enterprise Linux 7 Installation Guide

2. **Register RHEL**: Once RHEL 7 is installed, register the system. You will be prompted to enter your user name and password. Note that the user name and password are the same as your login credentials for Red Hat Customer Portal.

```
# subscription-manager register
Registering to: subscription.rhsm.redhat.com:443/subscription
Username: ********
Password: **********
```

3. **Choose pool ID**: Determine the pool ID of a subscription that includes Red Hat Enterprise Linux Server. Type the following at a shell prompt to display a list of all subscriptions that are available for your system, then attach the pool ID of one that meets that requirement:

```
# subscription-manager list --available    Find valid RHEL pool ID
# subscription-manager attach --pool=pool_id
```

4. **Enable repositories**: Enable the following repositories, which will allow you to install the docker package and related software:
**NOTE**: The repos shown here are for X86_64 architectures. See Supported Architectures for Containers on RHEL to learn the names of repositories for other architectures.

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

It is possible that some Red Hat subscriptions include enabled repositories that can conflict with eachother. If you believe that has happened, before enabling the repos shown above, you can disable all repos. See the How are repositories enabled solution for information on how to disable unwanted repositories.

**NOTE**: For information on the channel names required to get docker packages for Red Hat Satellite 5, refer to Satellite 5 repo to install Docker on Red Hat Enterprise Linux 7 .

5. **Install Docker**: Some releases of RHEL and RHEL Atomic Host include two different versions of Docker:

- **docker**: This package includes the version of Docker that is the default for the current release of RHEL. Install this package if you want a more stable version of Docker that is compatible with the current versions of Kubernetes and OpenShift available with Red Hat Enterprise Linux.

- **docker-latest**: This package traditionally included a later version of Docker that you could use if you wanted to work with newer features of Docker.
**NOTE**: As of RHEL 7.5, the docker-latest package is deprecated and should not be used. It also no longer supports a later version of Docker. For more information on the **docker-latest** package, see Introducing docker-latest for RHEL 7 and RHEL Atomic Host .

  To install and use the default **docker** package (along with a couple of dependent packages if they are not yet installed), type the following:

```
# yum install docker device-mapper-libs device-mapper-event-libs
```

6. **Start docker**:

```
# systemctl start docker.service
```

7. **Enable docker**:

```
# systemctl enable docker.service
```

8. **Check docker status**:

```
# systemctl status docker.service
   docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset:
disabled)
    Drop-in: /usr/lib/systemd/system/docker.service.d
             |-flannel.conf
     Active: active (running) since Thu 2016-05-09 22:398:47 EDT; 14s ago
       Docs: http://docs.docker.com
   Main PID: 13495 (sh)
     CGroup: /system.slice/docker.service
             └─13495 /bin/sh -c /usr/bin/docker-current daemon $OPTIONS
   ...
```

With the docker service running, you can obtain some Docker images and use the **docker** command to begin working with Docker images in RHEL 7.

## 5.3. GETTING DOCKER IN RHEL ATOMIC HOST

RHEL Atomic Host is a light-weight Linux operating system distribution that was designed specifically for running containers. It contains two different versions of the docker service, as well as some services that can be used to orchestrate and manage Docker containers, such as Kubernetes. Only one version of the docker service can be running at a time.

Because RHEL Atomic Host is more like an appliance than a full-featured Linux system, it is not made for you to install RPM packages or other software on. Software is added to Atomic Host systems by running container images.

RHEL Atomic Host has a mechanism for updating existing packages, but not for allowing users to add new packages. Therefore, you should consider using a standard RHEL 7 server system to develop your applications (so you can add a full compliment of development and debugging tools), then use RHEL Atomic Host to deploy your containers into a variety of virtualization and cloud environment.

That said, you can install a RHEL Atomic Host system and use it to run, build, stop, start, and otherwise work with containers using the examples shown in this topic. To do that, use the following procedure to get and install RHEL Atomic Host.

1. **Get RHEL Atomic Host**: RHEL Atomic Host is available from the Red Hat Customer Portal. You have the option of running RHEL Atomic Host as a live image (in **.qcow2** format) or installing RHEL Atomic Host from an installation medium (in .iso format). You can get RHEL Atomic in those (and other formats) from here:
   RHEL Atomic Host Downloads

   Then follow the Red Hat Enterprise Linux Atomic Host Installation and Configuration Guide instructions for setting up Atomic to run in one of several different physical or virtual environments.

2. **Register RHEL Atomic Host**: Once RHEL Atomic Host is installed, register the system using Subscription Management tools. (This will allow you to run **atomic upgrade** to upgrade Atomic software, but it won't let you install additional packages using the yum command.) For example:

```
# subscription-manager register --username=rhnuser \
  --password=rhnpasswd --auto-attach
```

**IMPORTANT**: Running containers with the docker command, as described in this topic, does not specifically require you to register the RHEL Atomic Host system and attach a subscription. However, if you want to run **yum install** commands within a container, the container must get valid subscription information from the RHEL Atomic Host or it will fail. If you need to enable repositories other than those enabled by default with the RHEL version the host is using, you should edit the **/etc/yum.repos.d/redhat.repo** file. You can do that manually within the container and set enabled=1 for the repository you want to use. You can also use **yum-config-manager**, a command-line tool for managing Yum repo files. You can use the following command to enable repos:

```
# yum-config-manager --enable REPOSITORY
```

You can also use **yum-config-manager** to display Yum global options, add repositories and others. **yum-config-manager** is documented in detail in the Red Hat Enterprise Linux 7 System Administrator's Guide. Since **redhat.repo** is a big file and editing it manually can be error prone, it is recommended to use **yum-config-manager**.

3. **Start using Docker**: RHEL Atomic Host comes with the docker package already installed and enabled. So, once you have logged in and subscribed your Atomic system, here is the status of docker and related software:

   - You can immediately begin running the docker command to work with docker images and containers.
   - The docker-distribution package is not installed. If you want to be able to pull and push images between your Atomic system and a private registry, you can install the docker-distribution package on a RHEL 7 system (as described next) and access that registry to store your own container images.
   - A set of kubernetes packages, used to orchestrate Docker containers, are installed on RHEL Atomic Host, but Kubernetes services are not enabled by default. You need to enable and start several Kubernetes-related services to be able to orchestrate containers in RHEL Atomic Host with Kubernetes.

## 5.4. CHANGING THE DOCKER SERVICE

Whether you are using the docker service in RHEL Atomic Host or on a RHEL Server, you can change the behavior of the docker service. Ways of changing the behavior of the docker service include:

- **docker-latest**: Some releases of RHEL and RHEL Atomic Host include a stable version of docker, **docker** and a later version, **docker-latest**. At any time you can run either one of them. For more information on **docker** and **docker-latest**, their differences, and switching between them, see Introducing docker-latest for RHEL 7 and RHEL Atomic Host .
- **docker daemon settings**: Another way to change how the docker service behaves is to changes settings that are passed to the docker daemon in the /etc/sysconfig/docker file. To see a list of options available with **docker daemon**, type **docker daemon --help**. The next section shows examples of docker daemon features you might want to change.

## 5.5. MODIFYING THE DOCKER DAEMON OPTIONS (/ETC/SYSCONFIG/DOCKER)

When the docker daemon starts in RHEL or RHEL Atomic Host, it reads the settings in the **/etc/sysconfig/docker** file and adds them to the **docker daemon** command line. See available options by typing the following command:

```
$ docker daemon --help
```

The following are a few options you may want to consider adding to your **/etc/sysonfig/docker** file so that they are picked up when your docker daemon runs.

### 5.5.1. Default options

The OPTIONS value in **/etc/sysconfig/docker** sets the options that are sent by default to the docker daemon. These include **--selinux-enabled** (which enables the SELinux feature for the daemon) and **--log-driver** (which tells docker to pass log messages to the systemd journal). Any other options can be added (space-separated) to that line:

```
OPTIONS='--selinux-enabled --log-driver=journald'
```

### 5.5.2. Access port options

By default, the docker daemon only listens for API requests through a unix domain socket, which is only exposed to the local host and requires root user permissions or docker group permissions to access the daemon.

The Docker project does not provide an authentication method with the docker daemon because it expects you to do remote authentication through Docker Swarm. With Red Hat, you can gain outside access to container services using the **oc** command, which requires OpenShift authentication before accessing container services.

However, you can configure the docker daemon to listen on an external TCP port as well. *Red Hat recommends against doing this, because the docker daemon does no authentication, so any external process with access to the TCP port would have root access to the daemon.*

#### 5.5.2.1. Exposing the docker daemon through a TCP port

If you accept the risks, you can configure the docker daemon to listen on an external TCP port by adding the **-H** option to the daemon at run time. For example, to make docker always start with this configuration, open the **/etc/sysconfig/docker** file and modify the OPTIONS line:

```
OPTIONS='--selinux-enabled --log-driver=journald --signature-verification=false -H tcp://0.0.0.0:2375'
```

This configuration exposes the docker daemon to any requests on the encrypted TCP port 2376 for all external interfaces. Port 2375 can be used for unencrypted communication with the daemon.

### 5.5.3. Registry options

When asked to search for or pull images, the docker service is configured to use the Docker registry (docker.io) and the Red Hat registry (registry.access.redhat.com) to complete those activities. In RHEL and RHEL Atomic Host, registry options previously set in the the /etc/sysconfig/docker file are now set in the /etc/containers/registries.conf file.

The registries.conf file lets you set which registries to search, which insecure (non-authenticated) registries to allow, and which secure registries (TLS enabled) to allow. Setting in the registries.conf file not only apply to registries used by the docker service, but also to those used by other container tools (such as podman) and engines (such as CRI-O).

Here are some examples of settings in the registries.conf file:

```
[registries.search]
registries = ["reg1.example.com", "reg2.example.com"]

[registries.insecure]
registries = ["reg3.example.com"]

[registries.block]
registries = ['docker.io']
```

In the example, reg1 and reg2 are searched when you run **docker search** or **podman search** commands. The reg3 registry is set as an insecure registry, which means that images can be retrieved from that registry with standard web protocols, instead of TLS.

To prevent access to a registry (docker service only), you can set a system in the registries.insecure block. In this example, the docker services is prevented from pulling images from the docker.io registry.

### 5.5.4. User namespaces options

Enabling the user namespaces mapping option for the docker daemon allows you to run applications with root privilege inside a container, but have them run as a different, typically non-privileged, user on the host. To set up user namespaces mappings, you need to:

- Enable a few user namespaces kernel options
- Tell the docker daemon to remap user namespaces
- Set up the user namespace mapping, based on user/group names or IDs

Assigning **default** to the **--userns-remap** option creates a user and group named dockremap. The associated UID and GID numbers are mapped into that account in the **/etc/subuid** and **/etc/subgid** files, respectively. Currently, only a single UID and GID can be mapped per daemon. See Daemon user namespace options for details on how user namespaces work.

**NOTE**

The user namespaces kernel feature is fully supported (generally available) in RHEL 7.4. However, the specific implementation of user namespaces related to the docker service is identified as a technology preview while Red Hat locks down a few security issues associated with non-root user mounts.

Follow these instructions to enable user namespaces:

1. Add the **namespace.unpriv_enable=1** option to the kernel (vmlinuz*) command line. To do this, use the grubby command as follows (replacing the exact version of vmlinuz with the one on your system):

```
# grubby --args="namespace.unpriv_enable=1" \
  --update-kernel=/boot/vmlinuz-3.10.0-693.el7.x86_64
```

2. Add a value to the **user.max_user_namespaces** kernel tuneable so it is set permanently as follows:

```
# echo "user.max_user_namespaces=15076" >> /etc/sysctl.conf
```

3. Assign users and groups to be mapped by user namespaces. To not conflict with any existing UIDs and GIDs in this example, the new UID and GID are mapped into 808080.

```
# echo dockremap:808080:1000 >> /etc/subuid
# echo dockremap:808080:1000 >> /etc/subgid
```

4. Edit the **/etc/sysconfig/docker** file and add **--userns-remap** to the OPTIONS value, so it is picked up when the docker service runs. Here's how that line would look with other values on the OPTIONS line:

```
OPTIONS='--selinux-enabled --log-driver=journald --signature-verification=false --userns-remap=default'
```

5. Reboot the system. After the system comes up, check that the kernel options were properly assigned and that the docker service is running with user namespaces enabled.

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-3.10.0-693.el7.x86_64 ... quiet LANG=en_US.UTF-8
namespace.unpriv_enable=1
$ sysctl -a | grep user.max_user_namespaces
user.max_user_namespaces = 15076
$ systemctl status docker | grep userns
           ├─1044 /usr/bin/dockerd-current --add-runtime docker-
runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --authorization-
plugin=rhel-push-plugin --exec-opt native.cgroupdriver=systemd --userland-proxy-
path=/usr/libexec/docker/docker-proxy-current --selinux-enabled --log-driver=journald --
signature-verification=false --userns-remap=default --add-registry registry.access.redhat.com
```

6. Try running a container to make sure the docker service is working:

```
$ sudo docker run --rm -it rhel7 /bin/bash
[root@8b351bc1599f /]#
```

Open a separate shell (outside the container) to check that the assigned user ID (in this case, 808080) is used to run the containerized bash shell in the example:

```
$ ps -ef | grep /bin/bash
root      9234  3888  0 13:32 pts/0    00:00:00 /usr/bin/docker-current run --rm -it rhel7
/bin/bash
808080    9300  9286  0 13:32 pts/1    00:00:00 /bin/bash
```

## 5.6. WORKING WITH DOCKER REGISTRIES

A Docker registry provides a place to store and share docker containers that are saved as images that can be shared with other people. While you can build and store container images on your local system without installing a registry, or use the Docker Hub Registry to share your images with the world, installing a private registry lets you share your images with a private group of developers or users.

With the registry software available with RHEL and RHEL Atomic Host, you can pull images from the Red Hat Customer Portal and push or pull images to and from your own private registry. You see what images are available to pull from the Red Hat Customer Portal (using **docker pull**) by searching the Red Hat Container Images Search Page.

This section describes how to start up a local registry, load Docker images to your local registry, and use those images to start up docker containers. The version of the Docker Registry that is currently available with Red Hat Enterprise Linux is Docker Registry 2.0.

### 5.6.1. Creating a private Docker registry (optional)

To create a private Docker registry you can use the docker-distribution service. You can install the docker-distribution package in RHEL 7 (it's not available in Atomic) and enable and start the service as follows:

**NOTE**

RHEL Atomic Host does not support the **yum** command for installing packages. To get around this issue, you could use the **yumdownloader docker-distribution** command to download the package to a RHEL system, copy it to the Atomic system, install it on the Atomic system using **rpm-ostree install ./docker-distribution*rpm** and reboot. You could then set up the docker-distribution service as described below.

1. **Install docker-distribution**: To install the docker-distribution package you must have enabled the **rhel-7-server-extras-rpms** repository (as described earlier). They you can install the package as follows:

```
# yum install -y docker-distribution
```

2. **Enable and start the docker-distribution service**: Type the following to enable, start and check the status of the docker-distribution service:

```
# systemctl enable docker-distribution
# systemctl start docker-distribution
# systemctl status docker-distribution
● docker-distribution.service - v2 Registry server for Docker
   Loaded: loaded (/usr/lib/systemd/system/docker-distribution.service;
      disabled; vendor preset: disabled)
   Active: active (running) since Tue 2016-05-10 06:30:26 EDT; 1min 10s ago
 Main PID: 8923 (registry)
   CGroup: /system.slice/docker-distribution.service
           └─8923 /usr/bin/registry /etc/docker-distribution/registry/config.yml
   ...
```

3. **Registry firewall issues**: The docker-distribution service listens on TCP port 5000, so access to that port must be open to allow clients outside of the local system to be able to use the registry. This applies regardless of whether you are running docker-distribution and docker on the same system or on different systems. The firewalld service is available, but disabled by default in Atomic Host. You can enable and start firewalld, then open TCP port 5000 follows:

```
# systemctl enable firewalld
# systemctl start  firewalld
# firewall-cmd --zone=public --add-port=5000/tcp
```

```
# firewall-cmd --zone=public --add-port=5000/tcp --permanent
# firewall-cmd --zone=public --list-ports
5000/tcp
```

or if have enabled the legacy firewall service, you could add the following command to the /etc/sysconfig/iptables file to open access to that port each time the service starts:

```
iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 5000 -j ACCEPT
```

### 5.6.2. Getting images from remote Docker registries

To get Docker images from a remote registry (such as Red Hat's own Docker registry) and add them to your local system, use the **docker pull** command:

```
# docker pull <registry>[:<port>]/[<namespace>/]<name>:<tag>
```

The *<registry>* is a host that provides the docker-distribution service on TCP *<port>* (default: 5000). Together, *<namespace>* and *<name>* identify a particular image controlled by *<namespace>* at that registry. Some registries also support raw *<name>*; for those, *<namespace>* is optional. When it is included, however, the additional level of hierarchy that *<namespace>* provides is useful to distinguish between images with the same *<name>*. For example:

| Namespace | Examples (*<namespace>*/*<name>*) |
|---|---|
| organization | **redhat/kubernetes**, **google/kubernetes** |
| login (user name) | **alice/application**, **bob/application** |
| role | **devel/database**, **test/database**, **prod/database** |

The only Docker registry that Red Hat supports at the moment is the one at registry.access.redhat.com. If you have access to a Docker image that is stored as a tarball, you can load that image into your Docker registry from your local file system.

**docker pull**: Use the pull option to pull an image from a remote registry. To pull the rhel base image from the Red Hat registry, type **docker pull registry.access.redhat.com/rhel7/rhel**. To make sure that the image originates from the Red Hat registry, type the hostname of the registry, a slash, and the image name. The following command demonstrates this and pulls the **rhel** image for the Red Hat Enterprise Linux 7 release from the Red Hat registry:

```
# docker pull registry.access.redhat.com/rhel7/rhel
```

An image is identified by a repository name (registry.access.redhat.com), a namespace name (rhel7) and the image name (rhel). You could also add a tag (which defaults to :latest if not entered). The repository name **rhel**, when passed to the **docker pull** command without the name of a registry preceding it, is ambiguous and could result in the retrieval of an image that originates from an untrusted registry. If there are multiple versions of the same image, adding a tag, such as **latest** to form a name such as **rhel:latest**, lets you choose the image more explicitly.

To see the images that resulted from the above **docker pull** command, along with any other images on your system, type **docker images**:

```
# docker images
REPOSITORY
        TAG       IMAGE ID      CREATED       VIRTUAL SIZE
registry.access.redhat.com/rhel7/rhel
        latest    95612a3264fc  6 weeks ago   203.3 MB
registry.access.redhat.com/rhel7/rhel-tools
        latest    3b7bd2d69242  6 weeks ago   1.219 GB
registry.access.redhat.com/rhel7/cockpit-ws
        latest    3bf463e43334  6 weeks ago   220.1 MB
registry.access.redhat.com/aep3_beta/aep-docker-registry
        latest    3c272743b20a  6 weeks ago   478.5 MB
registry.access.redhat.com/rhel7/etcd
        latest    c0a7c32e9eb9  9 weeks ago   241.7 MB
```

**docker load**: If you have a container image stored as a tarball on your local file system, you can load that image tarball so you can run it with the docker command on your local system. Here is how:

1. With the Docker image tarball in your current directory, you can load that tarball to the local system as follows:

   ```
   # docker load -i rhel-server-docker-7.2.x86_64.tar.gz
   ```

2. To push that same image to the registry running on your localhost, tag the image with your hostname (or "localhost") plus the port number of the docker-distribution service (TCP port 5000). **docker push** uses that tag information to push the image to the proper registry:

   ```
   # docker tag bef54b8f8a2f localhost:5000/myrhel7
   docker push localhost:5000/myrhel7
   The push refers to a repository [localhost:5000/myrhel7] (len: 1)
   Sending image list
   Pushing repository localhost:5000/myrhel7 (1 tags)
   bef54b8f8a2f: Image successfully pushed
   latest: digest:
     sha256:7296465ccce190e08a71e6b2cfba56aa8279a1b329827c0f1016b80044c20cb9 size:
   5458
   ...
   ```

## 5.7. INVESTIGATING DOCKER IMAGES

If images have been pulled or loaded into your local registry, you can use the docker command **docker images** to view those images. Here's how to list the images on your local system:

```
# docker images
REPOSITORY                                TAG     IMAGE ID     CREATED      VIRTUAL SIZE
registry.access.redhat.com/rhel7/rhel-tools  latest  3b7bd2d69242 6 weeks ago  1.219 GB
registry.access.redhat.com/rhel7/cockpit-ws  latest  3bf463e43334 6 weeks ago  220.1 MB
registry.access.redhat.com/rhel7/rhel        latest  95612a3264fc 6 weeks ago  203.3 MB
```

NOTE

The default option to push an image or repository to the upstream Docker.io registry (**docker push**) is disabled in the Red Hat version of the docker command. To push an image to a specific registry, identify the registry, its port number, and a tag that you designate in order to identify the image.

## 5.8. INVESTIGATING THE DOCKER ENVIRONMENT

Now that you have the docker and docker-distribution services running, with a few containers available, you can start investigating the Docker environment and looking into what makes up a container. Run **docker** with the **version** and **info** options to get a feel for your Docker environment.

**docker version**: The version option shows which versions of different Docker components are installed.

```
# docker version
Client:
 Version:         1.10.3
 API version:     1.22
Package version: docker-common-1.10.3-44.el7.x86_64
 Go version:      go1.4.2
 Git commit:      7ffc8ee-unsupported
 Built:           Fri Jun 17 15:27:21 2016
 OS/Arch:         linux/amd64

Server:
 Version:         1.10.3
 API version:     1.22
 Package version: docker-common-1.10.3-44.el7.x86_64
 Go version:      go1.4.2
 Git commit:      7ffc8ee-unsupported
 Built:           Fri Jun 17 15:27:21 2016
 OS/Arch:         linux/amd64
```

**docker info**: The info option lets you see the locations of different components, such as how many local containers and images there are, as well as information on the size and location of Docker storage areas.

```
# docker info
Containers: 9
Images: 25
Server Version: 1.10.3
Storage Driver: devicemapper
 Pool Name: docker-253:0-21214-pool
 Pool Blocksize: 65.54 kB
 Base Device Size: 107.4 GB
 Backing Filesystem:
 Data file: /dev/loop0
 Metadata file: /dev/loop1
 Data Space Used: 5.367 GB
 Data Space Total: 107.4 GB
 Data Space Available: 5.791 GB
 Metadata Space Used: 4.706 MB
 Metadata Space Total: 2.147 GB
 Metadata Space Available: 2.143 GB
 Udev Sync Supported: true
 Deferred Removal Enabled: false
 Deferred Deletion Enabled: false
 Deferred Deleted Device Count: 0
 Data loop file: /var/lib/docker/devicemapper/devicemapper/data
 Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
 Library Version: 1.02.107-RHEL7 (2015-12-01)
Execution Driver: native-0.2
```

```
Logging Driver: json-file
Kernel Version: 3.10.0-327.18.2.el7.x86_64
Operating System: Red Hat Enterprise Linux Atomic Host 7.2
CPUs: 1
Total Memory: 1.907 GiB
Name: atomic-7.2-12
ID: JSDA:MGJV:ALYX:N6RC:YXER:M4OJ:GYR2:GYQK:BPZX:GQOA:F476:WLQY
```

## 5.9. WORKING WITH DOCKER FORMATTED CONTAINERS

Docker images that are now on your system (whether they have been run or not) can be managed in several ways. The **docker run** command lets you say which command to run in a container. Once a container is running, you can stop, start, and restart it. You can remove containers you no longer need (in fact you probably want to). Before you run an image, it is a good idea to investigate its contents.

**Investigate a container image**After you pull an image to your local system and before you run it, it is a good idea to investigate that image. Reasons for investigating an image before you run it include:

- Understanding what the image does
- Checking that the image has the latest security patches
- Seeing if the image opens any special privileges to the host system

Tools (such as openscap) are being integrated with container tools to allow them to scan a container image before you run it. In the mean time, however, you can use **docker inspect** to get some basic information about what an image does. You also have the option of mounting the image to your host system and using tools from the host to investigate what's in the image. Here is an example of investigating what a container image does before you run it:

1. **Inspect an image**: Run **docker inspect** to see what command is executed when you run the container image, as well as other information. Here are examples of examining the rhel7/rhel and rhel7/rsyslog container images (with only snippets of information shown here):

```
# docker inspect rhel7/rhel
...
"Cmd": [
        "/usr/bin/bash"
    ],
      "Image": "",
      "Volumes": null,
"Entrypoint": null,
 ...
```

```
# docker inspect rhel7/rsyslog
"INSTALL": "docker run --rm --privileged -v /:/host -e HOST=/host -e IMAGE=IMAGE
   -e NAME=NAME IMAGE /bin/install.sh",
"Name": "rhel7/rsyslog",
"RUN": "docker run -d --privileged --name NAME --net=host --pid=host
   -v /etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf
   -v /etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d
   -v /var/log:/var/log -v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run
   -v /etc/machine-id:/etc/machine-id -v /etc/localtime:/etc/localtime -e IMAGE=IMAGE
   -e NAME=NAME --restart=always IMAGE /bin/rsyslog.sh",
"Release": "21",
```

```
"UNINSTALL": "docker run --rm --privileged -v /:/host -e HOST=/host -e IMAGE=IMAGE -e
NAME=NAME IMAGE /bin/uninstall.sh",
"Vendor": "Red Hat, Inc.",
"Version": "7.2",
  ...
```

The rhel7/rhel container will execute the bash shell, if no other argument is given when you start it with **docker run**. If an Entrypoint were set, its value would be used instead of the Cmd value (and the value of Cmd would be used as an argument to the Entrypoint command).

In the second example, the rhel7/rsyslog container image is meant to be run with the **atomic** command. The INSTALL, RUN, and UNINSTALL labels show that special privileges are open to the host system and selected volumes are mounted from the host when you do **atomic install**, **atomic run**, or **atomic uninstall** commands.

2. **Mount an image**: Using the **atomic** command, mount the image to the host system to further investigate its contents. For example, to mount the rhel7/rhel container image to the **/mnt** directory locally, type the following:

```
# atomic mount rhel7/rhel /mnt
# ls /mnt
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
boot  etc  lib  media  opt  root  sbin  sys  usr
```

After the **atomic mount**, the contents of the rhel7/rhel container are accessible from the **/mnt** directory on the host. Use **ls** or other commands to explore the contents of the image.

3. **Check the image's package list** To check the packages installed in the container, you can tell the **rpm** command to examine the packages installed on the file system you just made available to the /mnt directory:

```
# rpm -qa --root /mnt | less
```

You can step through the packages in the container or search for particular versions that may require updating. When you are done with that, you can browse the image's file system for other software of interest.

4. **Unmount the image**: When you are done investigating the image, you can unmount it as follows:

```
# atomic umount /mnt
```

In the near future, look for software scanning features, such as Openscap or Black Duck, to be available for scanning your container images. When they are, you will be able to use the **atomic scan** command to scan your images.

#### Running Docker containers

When you execute a **docker run** command, you essentially spin up and create a new container from a Docker image. That container consists of the contents of the image, plus features based on any additional options you pass on the **docker run** command line.

The command you pass on the **docker run** command line sees the inside the container as its running environment so, by default, very little can be seen of the host system. For example, by default, the running applications sees:

- The file system provided by the Docker image.

- A new process table from inside the container (no processes from the host can be seen).
- New network interfaces (by default, a separate docker network interface provides a private IP address to each container via DHCP).

If you want to make a directory from the host available to the container, map network ports from the container to the host, limit the amount of memory the container can use, or expand the CPU shares available to the container, you can do those things from the **docker run** command line. Here are some examples of docker run command lines that enable different features.

**EXAMPLE #1 (Run a quick command)**: This docker command runs the **ip addr show eth0** command to see address information for the eth0 network interface within a container that is generated from the RHEL image. Because this is a bare-bones container, we mount the **/usr/sbin** directory from the RHEL 7 host system for this demonstration (mounting is done by the -v option), because it contains the **ip** command we want to run. After the container runs the command, which shows the IP address (**172.17.0.2/16**) and other information about eth0, the container stops and is deleted ( **--rm**).

```
# docker run -v /usr/sbin:/usr/sbin \
   --rm rhel /usr/sbin/ip addr show eth0
20: eth0:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
   link/ether 4e:90:00:27:a2:5d brd ff:ff:ff:ff:ff:ff
   inet 172.17.0.10/16 scope global eth0
      valid_lft forever preferred_lft forever
   inet6 fe80::4c90:ff:fe27:a25d/64 scope link tentative
      valid_lft forever preferred_lft forever
```

If you feel that this is a container you wanted to keep around and use again, consider assigning a name to it, so you can start it again later by name. For example, I named this container myipaddr:

```
# docker run -v /usr/sbin:/usr/sbin \
   --name=myipaddr rhel /usr/sbin/ip addr show eth0
20: eth0:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
   link/ether 4e:90:00:27:a2:5d brd ff:ff:ff:ff:ff:ff
   inet 172.17.0.10/16 scope global eth0
      valid_lft forever preferred_lft forever
   inet6 fe80::4c90:ff:fe27:a25d/64 scope link tentative
      valid_lft forever preferred_lft forever
```

```
# docker start -i myipaddr
22: eth0:  mtu 1500 qdisc pfifo_fast state UP qlen 1000
   link/ether 4e:90:00:27:a2:5d brd ff:ff:ff:ff:ff:ff
   inet 172.17.0.10/16 scope global eth0
      valid_lft forever preferred_lft forever
   inet6 fe80::4c90:ff:fe27:a25d/64 scope link tentative
      valid_lft forever preferred_lft forever
```

**EXAMPLE #2 (View the Dockerfile in the container)** This is another example of running a quick command to inspect the content of a container from the host. All layered images that Red Hat provides include the Dockerfile from which they are built in **/root/buildinfo**. In this case you do not need to mount any volumes from the host.

```
# docker run --rm registry.access.redhat.com/rhel7/rsyslog  ls /root/buildinfo
Dockerfile-rhel7-rsyslog-7.2-21
```

Now you know what the Dockerfile is called, you can list its contents:

```
# docker run --rm registry.access.redhat.com/rhel7/rsyslog \
    cat /root/buildinfo/Dockerfile-rhel7-rsyslog-7.2-21
FROM 6c3a84d798dc449313787502060b6d5b4694d7527d64a7c99ba199e3b2df834e
MAINTAINER Red Hat, Inc.
ENV container docker
RUN yum -y update; yum -y install rsyslog; yum clean all

LABEL BZComponent="rsyslog-docker"
LABEL Name="rhel7/rsyslog"
LABEL Version="7.2"
LABEL Release="21"
LABEL Architecture="x86_64"
...
```

**EXAMPLE #3 (Run a shell inside the container)** Using a container to launch a bash shell lets you look inside the container and change the contents. Here, I set the name of the container to **mybash**. The **-i** creates an interactive session and **-t** opens a terminal session. Without **-i**, the shell would open and then exit. Without **-t**, the shell would stay open, but you wouldn't be able to type anything to the shell.

Once you run the command, you are presented with a shell prompt and you can start running commands from inside the container:

```
# docker run --name=mybash -it rhel /bin/bash
[root@49830c4f9cc4/]#
```

Although there are very few applications available inside the base RHEL image, you can add more software using the **yum** command. With the shell open inside the container, run the following commands:

```
[root@49830c4f9cc4/]# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.2 (Maipo)
[root@49830c4f9cc4/]# nmap
bash: nmap: command not found
[root@49830c4f9cc4/]# yum install -y nmap
[root@49830c4f9cc4/]# # nmap 192.168.122.1

Starting Nmap 6.40 ( http://nmap.org ) at 2016-05-10 08:55 EDT
Nmap scan report for 192.168.122.1
Host is up (0.00042s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
53/tcp   open  domain
5000/tcp open  upnp
...
[root@49830c4f9cc4/]# exit
```

Notice that the container is a RHEL 7.2 container. The **nmap** command is not included in the RHEL base image. However, you can install it with **yum** as shown above, then run it within that container. To leave the container, type **exit**.

Although the container is no longer running once you exit, the container still exists with the new software package still installed. Use **docker ps -a** to list the container:

```
# docker ps -a
```

```
CONTAINER ID  IMAGE COMMAND      CREATED        STATUS                     PORTS  NAMES
49830c4f9cc4  rhel  "/bin/bash"  2 minutes ago  Exited (0) 2 seconds ago        mybash
...
```

You could start that container again using **docker start** with the **-ai** options. For example:

```
# docker start -ai mybash
[root@a0aee493a605/]#
```

**EXAMPLE #4 (Bind mounting log files)** One way to make log messages from inside a container available to the host system is to bind mount the host's /dev/log device inside the container. This example illustrates how to run an application in a RHEL container that is named **log_test** that generates log messages (just the logger command in this case) and directs those messages to the /dev/log device that is mounted in the container from the host. The **--rm** option removes the container after it runs.

```
# docker run --name="log_test" -v /dev/log:/dev/log --rm rhel logger "Testing logging to the host"
# journalctl -b | grep Testing
May 10 09:00:32 atomic-7.2-12 logger[15377]: Testing logging to the host
```

**Investigating from outside of a Docker container**

Let's say you have one or more Docker containers running on your host. To work with containers from the host system, you can open a shell and try some of the following commands.

**docker ps**: The ps option shows all containers that are currently running:

```
# docker ps
CONTAINER ID IMAGE       COMMAND     CREATED         STATUS       PORTS    NAMES
0d8b2ded3af0 rhel:latest "/bin/bash" 10 minutes ago  Up 3 minutes          mybash
```

If there are containers that are not running, but were not removed (--rm option), the containers are still hanging around and can be restarted. The **docker ps -a** command shows all containers, running or stopped.

```
# docker ps -a
CONTAINER ID  IMAGE       COMMAND     CREATED      STATUS                  PORTS  NAMES
92b7ed0c039b  rhel:latest /bin/bash   2 days ago   Exited (0) 2 days ago        agitated_hopper
eaa96236afa6  rhel:latest /bin/bash   2 days ago   Exited (0) 2 days ago        prickly_newton
```

See the section "Working with Docker containers" for information on starting, stopping, and removing containers that exist on your system.

**docker inspect**: To inspect the metadata of an existing container, use the **docker inspect** command. You can show all metadata or just selected metadata for the container. For example, to show all metadata for a selected container, type:

```
# docker inspect mybash
[{
        "Labels": {
            "Architecture": "x86_64",
            "Authoritative_Registry": "registry.access.redhat.com",
            "BZComponent": "rhel-server-docker",
            "Build_Host": "rcm-img03.build.eng.bos.redhat.com",
            "Name": "rhel7/rhel",
```

```
            "Release": "56",
            "Vendor": "Red Hat, Inc.",
            "Version": "7.2"
    ...
```

**docker inspect --format**: You can also use inspect to pull out particular pieces of information from a container. The information is stored in a hierarchy. So to see the container's IP address (IPAddress under NetworkSettings), use the **--format** option and the identity of the container. For example:

```
# docker inspect --format='{{.NetworkSettings.IPAddress}}' mybash
172.17.0.2
```

Examples of other pieces of information you might want to inspect include .Path (to see the command run with the container), .Args (arguments to the command), .Config.ExposedPorts (TCP or UDP ports exposed from the container), .State.Pid (to see the process id of the container) and .HostConfig.PortBindings (port mapping from container to host). Here's an example of .State.Pid and .HostConfig.PortBindings:

```
# docker inspect --format='{{.State.Pid}}' mybash
5007
# docker inspect --format='{{.HostConfig.PortBindings}}' mybash
map[8000/tcp:[map[HostIp: HostPort:8000]]]
```

**Investigating within a running Docker container**

To investigate within a running Docker container, you can use the **docker exec** command. With **docker exec**, you can run a command (such as **/bin/bash**) to enter a running Docker container process to investigate that container.

The reason for using **docker exec**, instead of just launching the container into a bash shell, is that you can investigate the container as it is running its intended application. By attaching to the container as it is performing its intended task, you get a better view of what the container actually does, without necessarily interrupting the container's activity.

Here is an example using **docker exec** to look into a running container named myrhel_httpd, then look around inside that container.

1. **Launch a container**: Launch a container such as the myrhel_httpd container described in Building an image from a Dockerfile or some other Docker container that you want to investigate. Type **docker ps** to make sure it is running:

   ```
   # docker ps
   CONTAINER ID   IMAGE            COMMAND              CREATED        STATUS        PORTS
   NAMES
   1cd6aabf33d9   rhel_httpd:latest  "/usr/sbin/httpd -DF  6 minutes ago   Up 6 minutes
   0.0.0.0:80->80/tcp   myrhel_httpd
   ```

2. **Enter the container with docker exec** Use the container ID or name to open a bash shell to access the running container. Then you can investigate the attributes of the container as follows:

   ```
   # docker exec -it myrhel_httpd /bin/bash
   [root@1cd6aabf33d9 /]# cat /etc/redhat-release
   Red Hat Enterprise Linux Server release 7.2 (Maipo)
   [root@1cd6aabf33d9 /]# ps -ef
   ```

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 08:41 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache       7     1  0 08:41 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
...
root        12     0  0 08:54 ?        00:00:00 /bin/bash
root        35    12  0 08:57 ?        00:00:00 ps -ef
[root@1cd6aabf33d9 /]# df -h
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/docker-253:0-540464...  99G   414M   93G   1% /
tmpfs                           977M     0  977M   0% /dev
tmpfs                           977M     0  977M   0% /sys/fs/cgroup
tmpfs                           977M  320K  977M   1% /run/secrets
/dev/mapper/rhelah-root            14G   8.5G  5.2G  63% /etc/hosts
shm                             64M      0   64M   0% /dev/shm
[root@1cd6aabf33d9 /]# uname -r
3.10.0-327.18.2.el7.x86_64
[root@1cd6aabf33d9 /]# rpm -qa | more
redhat-release-server-7.2-9.el7.x86_64
filesystem-3.2-20.el7.x86_64
basesystem-10.0-7.el7.noarch
...
bash-4.2# free -m
        total   used    free  shared  buff/cache   available
Mem:     1953    134     354       0        1464        1655
Swap:    1055      0    1055
[root@1cd6aabf33d9 /]# exit
```

The commands just run from the bash shell (running inside the container) show you several things. The container holds a RHEL Server release 7.1 system. The process table (ps -ef) shows that the httpd command is process ID 1 (followed by five other httpd processes), /bin/bash is PID 12 and ps -ef is PID 35. Processes running in the host's process table cannot be seen from within the container. The container's file system consumes 414M of the 9.8G available root file system space.

There is no separate kernel running in the container (uname -r shows the host system's kernel: 3.10.0-229.1.2.el7.x86_64). The rpm -qa command lets you see the RPM packages that are included inside the container. In other words, there is an RPM database inside of the container. Viewing memory (free -m) shows the available memory on the host (although what the container can actually use can be limited using cgroups).

### Starting and stopping containers

If you ran a container, but didn't remove it (--rm), that container is stored on your local system and ready to run again. To start a previously run container that wasn't removed, use the **start** option. To stop a running container, use the **stop** option.

**Starting containers**: A docker container that doesn't need to run interactively can start with only the start option and the container ID or name:

```
# docker start myrhel_httpd
myrhel_httpd
```

To start a container so you can work with it from the local shell, use the -a (attach) and -i (interactive) options. Once the bash shell starts, run the commands you want inside the container and type exit to kill the shell and stop the container.

```
# docker start -a -i agitated_hopper
bash-4.2# exit
```

**Stopping containers**: To stop a running container that is not attached to a terminal session, use the stop option and the container ID or number. For example:

```
# docker stop myrhel_httpd
myrhel_httpd
```

The **stop** option sends a SIGTERM signal to terminate a running container. If the container doesn't stop after a grace period (10 seconds by default), docker sends a SIGKILL signal. You could also use the **docker kill** command to kill a container (SIGKILL) or send a different signal to a container. Here's an example of sending a SIGHUP signal to a container (if supported by the application, a SIGHUP causes the application to re-read its configuration files):

```
# docker kill --signal="SIGHUP" myrhel_httpd
```

**Removing containers**

To see a list of containers that are still hanging around your system, run the **docker ps -a** command. To remove containers you no longer need, use the **docker rm** command, with the container ID or name as an option. Here is an example:

```
# docker rm goofy_wozniak
```

You can remove multiple containers on the same command line:

```
# docker rm clever_yonath furious_shockley drunk_newton
```

If you want to clear out all your containers, you could use a command like the following to remove all containers (not images) from your local system (make sure you mean it before you do this!):

```
# docker rm $(docker ps -a -q)
```

## 5.10. CREATING DOCKER IMAGES

So far we have grabbed some existing docker container images and worked with them in various ways. To make the process of running the exact container you want less manual, you can create a Docker image from scratch or from a container you ran that combines an existing image with some other content or settings.

### 5.10.1. Building an image from a Dockerfile

Once you understand how images and containers can be created from the command line, you can try building containers in a more permanent way. Building container images from Dockerfile files is by far the preferred way to create Docker formatted containers, as compared to modifying running containers and committing them to images.

The procedure here involves creating a Dockerfile file that includes many of the features illustrated earlier:

- Choosing a base image

- Installing the packages needed for an Apache Web server (httpd)
- Mapping the server's port (TCP port 80) to a different port on the host (TCP port 8080)
- Launching the Web server

While many features for setting up a Docker development environment for RHEL 7 are in the works, there are some issues you should be aware of as you build your own docker containers:

- **Entitlements**: Here are a few issues associated with Red Hat entitlements as they relate to containers:
  - If you subscribe your Docker host system using Red Hat subscription manager, when you build a Docker image on that host, the build environment automatically has access to the same Red Hat software repositories you enabled on the host.
  - To make more repositories available when you build a container, you can enable those repositories on the host or within the container.
  - Because the subscription-manager command is not supported within a container, enabling a repo inside the /etc/yum.repos.d/redhat.repo file is one way to enable or disable repositories. Installing the yum-utils package in the container and running the yum-config-manager command is another.
  - If you build a RHEL 6 container on a RHEL 7 host, it will automatically pick up RHEL 6 versions of the repositories enabled on your host.
  - For more information on Red Hat entitlements within containers, refer to the Docker Entitlements solution.
- **Updates**: Docker containers in Red Hat Enterprise Linux do not automatically include updated software packages. It is your responsibility to rebuild your Docker images on occasion to keep packages up to date or rebuild them immediately when critical updates are needed.
- **Images**: By default, docker build will use the most recent version of the base image you identify from your local cache. You may want to pull (**docker pull** command) the most recent version of an image from the remote Docker registry before you build your new image. If you want a specific instance of an image, make sure you identify the tag. For example, just asking for the image "centos" will pull the centos:latest image. If you wanted the image for CentOS 6, you should specifically pull the centos:centos6 image.

1. **Create project directories**: On the host system where you have the docker and docker-distribution services running, create a directory for the project:

```
# mkdir -p httpd-project
# cd httpd-project
```

2. **Create the Dockerfile file** Open a file named Dockerfile using any text editor (such as **vim Dockerfile**). Assuming you have registered and subscribed your host RHEL 7 system, here's an example of what the Dockerfile file might look like to build a Docker container for an httpd server:

```
# My cool Docker image
# Version 1

# If you loaded redhat-rhel-server-7.0-x86_64 to your local registry, uncomment this
FROM line instead:
```

```
# FROM registry.access.redhat.com/rhel
# Pull the rhel image from the local registry
FROM registry.access.redhat.com/rhel

MAINTAINER Chris Negus
USER root

# Update image
RUN yum repolist --disablerepo=* && \
    yum-config-manager --disable \* > /dev/null && \
    yum-config-manager --enable rhel-7-server-rpms > /dev/null
# Add httpd package. procps and iproute are only added to investigate the image later.
RUN yum install httpd procps iproute -y
RUN echo container.example.com > /etc/hostname

# Create an index.html file
RUN bash -c 'echo "Your Web server test is successful." >> /var/www/html/index.html'
```

3. **Checking the Dockerfile syntax (optional)** Red Hat offers a tool for checking a Dockerfile file on the Red Hat Customer Portal. If you like, you can go to the Linter for Dockerfile page and check your Dockerfile file before you build it.

4. **Build the image**: To build the image from the Dockerfile file, you need to use the build option and identify the location of the Dockerfile file (in this case just a "." for the current directory):
   **NOTE**: Consider using the --no-cache option with docker build. Using --no-cache prevents the caching of each build layer, which can cause you to consume excessive disk space.

```
# docker build -t rhel_httpd .
Sending build context to Docker daemon  2.56 kB
Step 1 : FROM registry.access.redhat.com/rhel
 ---> a2d9f633eaab
Step 2 : MAINTAINER Chris Negus
 ---> Using cache
 ---> f6e85edac116
Step 3 : USER root
 ---> Using cache
 ---> 6347eb069a0b
...
Complete!
 ---> 07dff0b4fe81
Removing intermediate container cba7a631d183
Step 7 : RUN echo container.example.com > /etc/hostname
 ---> Running in 543c6724fbb0
 ---> 25742b5830dc
Removing intermediate container 543c6724fbb0
Step 8 : RUN bash -c 'echo "Your Web server test is successful." >>
/var/www/html/index.html'
 ---> Running in d0e6461338f1
 ---> 6f6a6cf60208
Removing intermediate container d0e6461338f1
Successfully built 6f6a6cf60208
```

5. **Run the httpd server in the image** Use the following command to run the httpd server from the image you just build (named rhel_httpd in this example):

```
# docker run -d -t --name=myrhel_httpd \
      -p 80:80 -i rhel_httpd:latest \
      /usr/sbin/httpd -DFOREGROUND
```

6. **Check that the server is running** From another terminal on the host, type the following to check that you can get access the httpd server:

```
# netstat -tupln | grep 80
tcp6       0      0 :::80       :::*       LISTEN      26137/docker-proxy
# curl localhost:80
Your Web server test is successful.
```

### 5.10.2. Creating an image from a container

The following procedure describes how to create a new image from an existing image (rhel:latest) and a set of packages you choose (in this case an Apache Web server, httpd).

**NOTE**: For the current release, the default RHEL 7 container image you pull from Red Hat will be able to draw on RHEL 7 entitlements available from the RHEL or RHEL Atomic Host system. So, as long as your Docker host is properly subscribed and the repositories are enabled that you need to get the software you want in your container (and have Internet access from your Docker host), you should be able to install packages from RHEL 7 software repositories.

1. **Install httpd on a new container**: Assuming you have loaded the **rhel** image from the Red Hat Customer Portal into your local system, and properly subscribed your host using Red Hat subscription management, the following command will:

   - Use that image as a base image
   - Get the latest versions of the currently installed packages (update)
   - Install the httpd package (along with any dependent packages)
   - Clean out all yum temporary cache files

```
# docker run -i rhel:latest /bin/bash -c "yum clean all;  \
   yum install -y httpd; yum clean all"
```

2. **Commit the new image**: Get the new container's ID or name ( **docker ps -l**), then commit that container to your local repository. When you commit the container to a new image, you can add a comment (-m) and the author name (-a), along with a new name for the image (rhel_httpd). Then type **docker images** to see the new image in your list of images.

```
# docker ps -l
CONTAINER ID  IMAGE          COMMAND             CREATED           STATUS
PORTS  NAMES
f6832df8da0a  redhat/rhel7:0 /bin/bash -c 'yum cl About a minute ago  Exited (0) 13 seconds
ago        backstabbing_ptolemy4
```

```
# docker commit -m "RHEL with httpd" -a "Chris Negus" f6832df8da0a rhel_httpd
630bd3ff318b8a5a63f1830e9902fec9a4ab9eade7238835fa6b7338edc988ac
```

```
# docker images
REPOSITORY   TAG     IMAGE ID      CREATED         VIRTUAL SIZE
```

```
rhel_httpd   latest  630bd3ff318b  27 seconds ago  170.8 MB
redhat/rhel  latest  e1f5733f050b  4 weeks ago     140.2 MB
```

3. **Run a container from new image** Using the image you just created, run the following **docker run** command to start the Web server (httpd) you just installed. For example:

```
# docker run -d -p 8080:80 rhel_httpd:latest \
      /usr/sbin/httpd -DFOREGROUND
```

In the example just shown, the Apache Web server (httpd) is listening on port 80 on the container, which is mapped to port 8080 on the host.

4. **Check that container is working** To make sure the httpd server you just launched is available, you can try to get a file from that server. Either open a Web browser from the host to address http://localhost:8080 or use a command-line utility, such as curl, to access the httpd server:

```
# curl http://localhost:8080
```

## 5.11. TAGGING IMAGES

You can add names to images to make it more intuitive to understand what they contain. Using the **docker tag** command, you essentially add an alias to the image, that can consist of several parts. Those parts can include:

*registryhost/username/NAME:tag*

You can add just *NAME* if you like. For example:

```
# docker tag 474ff279782b myrhel7
```

In the previous example, the **rhel7** image had a image ID of 474ff279782b. Using **docker tag**, the name **myrhel7** now also is attached to the image ID. So you could run this container by name (rhel7 or myrhel7) or by image ID. Notice that without adding a :tag to the name, it was assigned :latest as the tag. You could have set the tag to 7.2 as follows:

```
# docker tag 474ff279782b myrhel7:7.2
```

To the beginning of the name, you can optionally add a user name and/or a registry name. The user name is actually the repository on Docker.io that relates to the user account that owns the repository. Tagging an image with a registry name was shown in the "Tagging Images" section earlier in this document. Here's an example of adding a user name:

```
# docker tag 474ff279782b cnegus/myrhel7
# docker images | grep 474ff279782b
rhel7           latest  474ff279782b  7 months ago  139.6 MB
myrhel7         latest  474ff279782b  7 months ago  139.6 MB
myrhel7         7.1     474ff279782b  7 months ago  139.6 MB
cnegus/myrhel7  latest  474ff279782b  7 months ago  139.6 MB
```

Above, you can see all the image names assigned to the single image ID.

## 5.12. SAVING AND IMPORTING IMAGES

If you want to save a Docker image you created, you can use docker save to save the image to a tarball. After that, you can store it or send it to someone else, then reload the image later to reuse it. Here is an example of saving an image as a tarball:

```
# docker save -o myrhel7.tar myrhel7:latest
```

The **myrhel7.tar** file should now be stored in your current directory. Later, when you ready to reuse the tarball as a container image, you can import it to another docker environment as follows:

```
# cat myrhel7.tar | docker import - cnegus/myrhel7
```

## 5.13. REMOVING IMAGES

To see a list of images that are on your system, run the **docker images** command. To remove images you no longer need, use the **docker rmi** command, with the image ID or name as an option. (You must stop any containers using an image before you can remove the image.) Here is an example:

```
# docker rmi rhel
```

You can remove multiple images on the same command line:

```
# docker rmi rhel fedora
```

If you want to clear out all your images, you could use a command like the following to remove all images from your local registry (make sure you mean it before you do this!):

```
# docker rmi $(docker images -a -q)
```

## 5.14. SUMMARY

At this point, you should be able to get Red Hat Docker installed with the docker and docker-distribution services working. You should also have one or more Docker images to work with, as well as know how to run containers and build your own images.