Red Hat Enterprise Linux Atomic Host 7
Getting Started with Containers

Getting Started with Containers

Red Hat Atomic Host Documentation Team

Getting Started with Containers

Legal Notice

Abstract

Containers and Container Development

Table of Contents

CHAPTER 1. GET STARTED WITH DOCKER FORMATTED CONTAINER IMAGES

1.1. OVERVIEW

Docker has quickly become one of the premier projects for containerizing applications. This topic provides a hands-on approach to start using Docker in Red Hat Enterprise Linux 7 and RHEL Atomic Host by getting and using Docker images and working with Docker containers.

1.2. BACKGROUND

The Docker project provides the means of packaging applications in lightweight containers. Running applications within Docker containers offers the following advantages:

- **Smaller than Virtual Machines**: Because Docker images contain only the content needed to run an application, saving and sharing is much more efficient with Docker 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 Docker 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 Docker container is capable of being run in multiple environments.

Currently, you can run Docker containers on Red Hat Enterprise Linux 7 (RHEL 7) Server and Red Hat Enterprise Linux Atomic (based on RHEL 7) 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 Docker containers in OpenStack, VirtualBox, Linux KVM and several different cloud environments.

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

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

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

- **Release Notes**: Refer to the Atomic Host and Containers section of the RHEL 7 Release Notes for an overview of Docker and related features in RHEL 7.
- **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 README**: After you install the docker package, refer to the README.md file in the /usr/share/doc/docker-1* directory.

- **Docker man pages**: Again, with docker installed, 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.

1.3. 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 Docker 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.

NOTE: The docker packages and other container-related packages are only available for the RHEL Server and RHEL Atomic Host editions. They are not available for Workstation or other variants of RHEL.

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 using Subscription Management tools and install the docker package. Also enable the software repositories needed. (Replace pool_id with the pool ID of your RHEL 7 subscription.) For example:

```
# subscription-manager register --username=rhnuser --
password=rhnpasswd
# subscription-manager list --available    Find valid RHEL pool
ID
# subscription-manager attach --pool=pool_id
# subscription-manager repos --enable=rhel-7-server-extras-rpms
# subscription-manager repos --enable=rhel-7-server-optional-rpms
```

   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.

3. **Install Docker**: The current release of RHEL and RHEL Atomic Host include two different versions of Docker. Here are the Docker packages you have to choose from:

   - **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 includes a later version of Docker that you can use if you want to work with newer features of Docker. This version is not compatible with the versions of Kubernetes and OpenShift that are available with the current release of Red Hat Enterprise Linux.

  NOTE: For more information on the contents of **docker** and **docker-latest** packages, see the Atomic Host and Containers section of the Red Hat Enterprise Linux Release Notes for more details on the differences between the two packages and how to enable the **docker-latest** package.

  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
```

4. **Start docker**:

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

5. **Enable docker**:

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

6. **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.

1.4. 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.

1.5. 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**: On occasion, a stable version of the docker service will be available in RHEL and RHEL Atomic Host from the **docker** software package, while a later version will also be available in the **docker-latest** package. When those packages offer different docker releases, you can switch from one to the other (only one running at a time). See Introducing docker-latest for RHEL 7 and RHEL Atomic Host for information on how those two packages differ.
- **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.

1.6. 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.

1.6.1. 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'
```

1.6.2. Registry options

When asked to search for or pull images, the docker command uses the Docker registry (docker.io) to complete those activities. In RHEL and RHEL Atomic Host, this entry in the /etc/sysconfig/docker file causes the Red Hat registry (registry.access.redhat.com) to be used first:

```
ADD_REGISTRY='--add-registry registry.access.redhat.com'
```

If you wanted to add a private registry that you installed yourself, just add another ADD_REGISTRY. For example:

```
ADD_REGISTRY='--add-registry myregistry.example.com'
```

If you want to prevent users from pulling images from the Docker registry, uncomment the BLOCK_REGISTRY entry so it appears as follows:

```
BLOCK_REGISTRY='--block-registry docker.io'
```

To access a registry that uses https protocol for security, but is not set up with certificates for authentication, you can still access that registry by defining it as an insecure registry in the /etc/sysconfig/docker file. For example:

```
INSECURE_REGISTRY='--insecure-registry newregistry.example.com'
```

1.6.3. User namespace options

There are times when an application you run from a container expects to be run as the root user inside that container. For reasons of security or resource usage, you might want the application to run as a regular user on the host system. To map user ID from within a container to different user IDs on the host system, you can add the **--userns** option to the /etc/sysconfig/docker file.

Before you can use the --userns feature for docker daemon, you need to create the user mapping in /etc/subuid and /etc/subgid files. This process is described in the Docker Daemon page.

1.7. 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.

1.7.1. 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:

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. You can open TCP port 5000 follows:

```
# 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 a firewall using iptables firewall rules directly, you could find a way to have the following command run each time you boot your system:

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

1.7.2. 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:

sha256:7296465ccce190e08a71e6b2cfba56aa8279a1b329827c0f1016b80044
c20cb9 size: 5458
...
```

1.7.3. 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.

1.7.4. 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
```

1.7.5. 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)
```

1.7.6. 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.

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 update -y; 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
```

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. The "RUN yum update -y" line shown in the Dockerfile example below is one way to update your packages each time the Docker image is rebuilt.

- **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.

    - **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
      ```

    - **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 update -y
      # 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'
      ```

    - **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.

    - **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 .
      Uploading context  2.56 kB
      ```

```
Uploading context
Step 0 : FROM registry.access.redhat.com/rhel
 ---> f5f7ddddef7d
Step 1 : MAINTAINER Chris Negus
 ---> Running in 3c605e879c72
 ---> 77828ebe8f6f
Removing intermediate container 3c605e879c72
Step 2 : RUN yum update -y
 ---> Running in 9f45bb262dc6
...
 ---> Running in f44ea9eb6155
 ---> 6a532e340ccf
Removing intermediate container f44ea9eb6155
Successfully built 6a532e340ccf
```

- **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
```

- **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.
```

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.

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
```

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)
```

1.8. 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.

CHAPTER 2. INSTALL AND DEPLOY AN APACHE WEB SERVER CONTAINER

2.1. 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)

2.2. 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 update -y
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**

2.3. 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
```

2.4. ATTACHMENTS

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

CHAPTER 3. INSTALL AND DEPLOY A MARIADB CONTAINER

3.1. 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

3.2. 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
```

3.3. 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.

3.4. ATTACHMENTS

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