Oracle® Linux 6
Docker User’s Guide
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Oracle Legal Notices

Copyright © 2012, 2019, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Oracle® Linux 6

6.2.2 Adding or removing container instances ................................................................. 45
7 For More Information About Docker .................................................................................. 47
Preface

The Oracle Linux 6 Docker User's Guide describes how to use Docker, which is an open-source, distributed-application platform that leverages Linux kernel technology to provide resource isolation management. Detail is provided on the advanced features of Docker and how it can be installed, configured and used on Oracle Linux 6.


Audience

This document is intended for administrators who need to install, configure and use the Docker engine on Oracle Linux 6. It is assumed that readers are familiar with web and virtualization technologies and have a general understanding of the Linux operating system.

Related Documents

The documentation for this product is available at:


Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><code>monospace</code></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.
Chapter 1 About Docker

Table of Contents

1.1 Notable Updates .............................................................. 1

Docker allows you to create and distribute applications across Oracle Linux systems and other operating systems that support Docker. Docker consists of the Docker Engine, which packages and runs the applications, and the Docker Hub, which shares the applications in a Software-as-a-Service (SaaS) cloud.

The Docker Engine is designed primarily to run single applications in a similar manner to LXC application containers that provide a degree of isolation from other processes running on a system.

The Docker Hub hosts applications as Docker images and provides services that allow you to create and manage a Docker environment. You must register with the Docker Hub Registry to be able to access its resources and services.

Note

The Docker Hub is owned and maintained by Docker, Inc. Oracle makes Docker images available on the Docker Hub that you can download and use with the Docker Engine. Oracle does not have any control otherwise over the content of the Docker Hub Registry site or its repositories.

For more information, see https://docs.docker.com.

Oracle provides access to the Oracle Container Registry for customers that have a Single Sign-On account at Oracle. The Oracle Container Registry contains Docker images for licensed commercial Oracle software products that you may use in your enterprise. Images may also be used for development and testing purposes. The license covers both production and non-production use. The Oracle Container Registry provides a web interface where customers are able to select Oracle Docker images and agree to terms of use before pulling the images using the standard Docker client software. More information on this service is provided in Section 5.1, “Using the Oracle Container Registry”.

1.1 Notable Updates

Changes to the Docker Engine tend to retain backward compatibility as far as possible. Changes are usually well documented and a detailed changelog is maintained at https://github.com/moby/moby/blob/master/CHANGELOG.md. In this section, changes that are considered significant, or of interest to users of the Docker Engine on Oracle Linux systems, are highlighted for convenience.

Docker 1.12

The focus of this release was to simplify and improve container orchestration, providing facilities such as load-balancing, service discovery, high availability and scalability out of the box. Features to handle multi-host and multi-container orchestration have been built right into the Docker engine to allow administrators to deploy and manage applications on a group of Docker Engines called a swarm. Docker swarm mode provides much of the functionality included in the original standalone Docker Swarm service that ran separately to the Docker Engine itself and includes additional features such as built-in load-balancing. By integrating this technology into the Docker Engine, deployment of a high availability clustering technology is simplified and these features are unified within a single API and CLI. All communications within the

1
Docker swarm are encrypted using Transport Layer Security (TLS) and cluster nodes are protected using cryptographic node fingerprint key technology to prevent node spoofing.

Important
The Docker Swarm functionality is released as a technology preview for Oracle Linux. As a technology preview, this feature is still under development but is made available for testing and evaluation purposes.

Docker 1.11

The Docker Engine has been rearchitected to run on top of a combination of the docker-containerd and docker-runc binaries. While this change is transparent and docker commands continue to work as they did in previous releases, the underlying technology further modularizes the Docker architecture in line with the Open Container Initiative (OCI) specification. These changes open up new possibilities for container execution backends and container management, including the potential to perform engine restarts and upgrades without the need to restart running containers.

Other notable changes in this version of the Docker Engine are:

• Experimental support for the MacVlan and IPVlan network drivers to take advantage of existing VLAN networking infrastructure

• Support for AAAA Records (aka IPv6 Service Discovery) in embedded DNS Server, which allows for IPv6 queries to be resolved locally without being forwarded to external servers

• Multiple A/AAAA records from embedded DNS Server for DNS Round robin to facilitate load-balancing between containers.

• Source the forwarded DNS queries from the container net namespace

• Better handling of low disk space to allow the device mapper to fail more gracefully in the case where there is insufficient disk space.
Chapter 2 Installing and Upgrading Docker

Table of Contents

2.1 Installing and Configuring the Docker Engine on Oracle Linux 6 .................................................... 3
2.2 Upgrading Docker ................................................................................................................................. 7

This chapter describes the steps required to perform an installation or an upgrade of Docker on an Oracle Linux 6 host.

2.1 Installing and Configuring the Docker Engine on Oracle Linux 6

Note

Docker version 1.9 and later require that you configure the system to use the Unbreakable Enterprise Kernel Release 4 (UEK R4) and boot the system with this kernel.

To install and configure the Docker Engine on an Oracle Linux 6 system:

1. If you want to install Docker, configure the system to use the Unbreakable Enterprise Kernel Release 4 (UEK R4) and boot the system with this kernel:

   a. If your system is registered with ULN, disable access to the ol6_x86_64_UEKR3_latest or ol6_x86_64_UEK_latest channels and enable access to the ol6_x86_64_UEKR4 channel.

      If you use the Oracle Linux yum server, disable the ol6_UEKR3_latest repository and enable the ol6_UEKR4 repository in the repository configuration files in /etc/yum.repos.d/uek-ol6.repo, for example:

      ```
      [ol6_UEKR4]
      name=Latest Unbreakable Enterprise Kernel Release 4 for Oracle Linux $releasever ($basearch)
      gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
      gpgcheck=1
      enabled=1
      
      [ol6_UEKR3_latest]
      name=Latest Unbreakable Enterprise Kernel for Oracle Linux $releasever ($basearch)
      gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
      gpgcheck=1
      enabled=0
      
      [ol6_UEK_latest]
      name=Latest Unbreakable Enterprise Kernel for Oracle Linux $releasever ($basearch)
      gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
      gpgcheck=1
      enabled=0
      ```

   b. Run the following command to upgrade the system to UEK R4:

      `# yum update`
c. To ensure that UEK R4 is the default boot kernel, edit `/boot/grub/grub.conf` and change the value of the `default` directive to index the entry for the UEK R4 kernel. For example, if the UEK R4 kernel is the first entry, set the value of `default` to 0.

d. Reboot the system, selecting the UEK R4 kernel if this is not the default boot kernel.

```bash
# reboot
```

2. If your system is registered with ULN, enable the `ol6_x86_64_addons` channel.

If you use the Oracle Linux yum server, enable the `ol6_addons` repository in the repository configuration file at `/etc/yum.repos.d/oracle-linux-ol6.repo`, for example:

```
[ol6_addons]
name=Oracle Linux $releasever Add ons ($basearch)
baseurl=https://yum.oracle.com/repo/OracleLinux/OL6/addons/$basearch/
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
gpgcheck=1
enabled=1
```

3. Install the `docker-engine` package.

```bash
# yum install docker-engine
```

4. By default, the Docker Engine uses the device mapper as a storage driver to manage Docker containers. As with LXC, there are benefits to using the snapshot features of btrfs instead.

```
Note
Oracle recommends using btrfs because of the stability and maturity of the technology. If a new device for btrfs is not available, you should use `overlay2` as the storage driver instead of `devicemapper` for performance reasons. You can configure `overlay2` by adding the `--storage-driver=overlay2` option to `other_args` in `/etc/sysconfig/docker`. The overlayfs file system is available with UEK R4.

For more information, see https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/.
```

To configure the Docker Engine to use btrfs instead of the device mapper:

a. Use `yum` to install the `btrfs-progs` package.

```bash
# yum install btrfs-progs
```

b. Create a btrfs file system on a suitable device such as `/dev/sdb` in this example:

```bash
# mkfs.btrfs /dev/sdb
```

```
Note
Any unused block device that is large enough to store several containers is suitable. The suggested minimum size is 1GB but you might require more space to implement complex Docker applications. If the system is a virtual machine, Oracle recommends that you create, partition, and format a new virtual disk. Alternatively, convert an existing ext3 or ext4 file system to btrfs. See https://docs.oracle.com/cd/E37670_01/E37355/html/
Installing and Configuring the Docker Engine on Oracle Linux 6

Oracle recommends that you upgrade to a current supported release.

The software described in this documentation is either no longer supported or is in extended support.

If an LVM volume group has available space, you can create a new logical volume and format it as a btrfs file system.

If a running container fills /var/lib/docker, a restart of the Docker Engine fails with an error similar to the following:

```
Error starting daemon: write /var/lib/docker/volumes/metadata.db: no space left on device
```

Without the Docker Engine running it becomes difficult to clean up or remove existing or obsolete images. If this happens, you can try to gain some space by removing the contents of /var/lib/docker/tmp. However, the best solution is to avoid reaching this situation by implementing quotas that prevent a scenario where the Docker Engine runs out of the disk space required to run.

5. Edit /etc/sysconfig/docker to configure global networking options, for example:

   - If your system needs to use a web proxy to access the Docker Hub, add the following lines:

     ```
     export HTTP_PROXY="proxy_URL:port"
     export HTTPS_PROXY="proxy_URL:port"
     ```

     Replace `proxy_URL` and `port` with the appropriate URL and port number for your web proxy.

   - To configure IPv6 support in version 1.5 and later of Docker, add the `--ipv6` option to `OPTIONS`, for example:

     ```
     OPTIONS="--ipv6"
     ```

     With IPv6 enabled, Docker assigns the link-local IPv6 address `fe80::1` to the bridge `docker0`.

     If you want Docker to assign global IPv6 addresses to containers, additionally specify the IPv6 subnet to the `--fixed-cidr-v6` option, for example:

     ```
     OPTIONS="--ipv6 -- fixed-cidr-v6='2001:db8:1::/64'"
     ```

     For more information about configuring Docker networking, see [https://docs.docker.com/engine/userguide/networking/](https://docs.docker.com/engine/userguide/networking/).

6. In version 1.5 and later of Docker, the docker service unshares its mount namespace to resolve `device busy` issues with the device mapper storage driver. However, this configuration breaks
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Installing and Configuring the Docker Engine on Oracle Linux 6

**autofs** in the host system and prevents you from accessing subsequently mounted volumes in Docker containers. The workaround is to stop the Docker service from unsharing its mount namespace.

Edit `/etc/init.d/docker` and remove the `$unshare -m --` parameters from the line that starts the daemon. For example, change the line that reads similar to the following:

```
"$unshare" -m -- $exec $other_args &>> $logfile &
```

so that it reads:

```
$exec $other_args &>> $logfile &
```

**Note**

You might need to reapply this workaround if you update the docker package and the change to `/etc/init.d/docker` is overwritten.

7. Start the docker service and configure it to start at boot time:

```
# service docker start
# chkconfig docker on
```

**Note**

If you have installed the mlocate package, it is recommended that you modify the PRUNEPATHS entry in `/etc/updatedb.conf` to prevent updatedb from indexing directories below `/var/lib/docker`, for example:

```
PRUNEPATHS="/media /tmp /var/lib/docker /var/spool /var/tmp"
```

This entry prevents locate from reporting files that belong to Docker containers.

To check that the docker service is running, use the following command:

```
# service docker status
docker (pid 1958) is running...
```

You can also use the docker command to display information about the configuration and version of the Docker Engine, for example:

```
# docker info
Containers: 0
Images: 6
Storage Driver: btrfs
Execution Driver: native-0.2
Kernel Version: 3.8.13-35.3.1.el7uek.x86_64
Operating System: Oracle Linux Server 6.6
```

```
# docker version
Client version: 1.3.3
Client API version: 1.15
Go version (client): go1.3.3
Git commit (client): 4e9bbfa/1.3.3
OS/Arch (client): linux/amd64
Server version: 1.3.3
Server API version: 1.15
Go version (server): go1.3.3
Git commit (server): 4e9bbfa/1.3.3
```

For more information, see the docker(1) manual page.
2.2 Upgrading Docker

Docker version 1.10 introduces content addressability to the way in which image data is stored on disk. This functionality provides better security and helps to ensure data integrity for Docker images and layers. Since the way in which files are stored on disk and are referenced within Docker has changed, any existing Docker images created using a prior version of Docker must be migrated to the new format. This new feature and the migration process are described in more detail at https://github.com/docker/docker/wiki/Engine-v1.10.0-content-addressability-migration.

Migration of Docker images is performed automatically after the upgrade when the Docker engine is first restarted. The upgrade process requires that all Docker containers are offline during the process and might take a significant period of time to complete. If you cannot afford the downtime required for the migration, you might use the migration utility referenced in the link provided above. However, you should note that Oracle does not package or support this utility.

Docker version 1.9 and later require that you configure the system to use the Unbreakable Enterprise Kernel Release 4 (UEK R4) and boot the system with this kernel.

The Docker package for version 1.8.1 and later is `docker-engine`, which conflicts with the `docker` package used by previous versions of Docker.

To upgrade Docker on Oracle Linux 6

1. Configure the system to use the Unbreakable Enterprise Kernel Release 4 (UEK R4) and boot the system with this kernel:
   a. If your system is registered with ULN, disable access to the `ol6_x86_64_UEKR3_latest` or `ol6_x86_64_UEK_latest` channels and enable access to the `ol6_x86_64_UEKR4` channel.

   If you use the Oracle Linux yum server, disable the `ol6_UEKR3_latest` repository and enable the `ol6_UERK4` repository in the repository configuration files in `/etc/yum.repos.d/uek-ol6.repo`, for example:

   ```
   [ol6_UERK4]
   name=Latest Unbreakable Enterprise Kernel Release 4 for Oracle Linux $releasever ($basearch)
   gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
   gpgcheck=1
   enabled=1
   
   [ol6_UERK3_latest]
   name=Latest Unbreakable Enterprise Kernel for Oracle Linux $releasever ($basearch)
   gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
   gpgcheck=1
   enabled=0
   
   [ol6_UERK_latest]
   name=Latest Unbreakable Enterprise Kernel for Oracle Linux $releasever ($basearch)
   gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
   gpgcheck=1
   enabled=0
   ```

   b. Run the following command to upgrade the system to UEK R4:

   ```
   # yum update
   ```
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.
To upgrade Docker on Oracle Linux 6

c. To ensure that UEK R4 is the default boot kernel, edit `/boot/grub/grub.conf` and change the value of the `default` directive to index the entry for the UEK R4 kernel. For example, if the UEK R4 kernel is the first entry, set the value of `default` to 0.

d. Reboot the system, selecting the UEK R4 kernel (version 4.1.12) if this is not the default boot kernel.

```
# reboot
```

2. Stop the `docker` service.

```
# service docker stop
```

3. If the older `docker` package is installed, remove it and install the `docker-engine` package.

```
# yum remove docker
# yum install docker-engine
```

4. Start the `docker` service.

```
# service docker start
```
Chapter 3 Docker Administration and Configuration

Table of Contents

3.1 Reloading or Restarting the Docker Engine ................................................................. 9
3.2 Enabling Non-root Users to Run Docker Commands .................................................. 9
3.3 Configuring User Namespace Remapping ................................................................. 9

This chapter describes common Docker Engine administration and configuration tasks with specific focus on usage on Oracle Linux 6.

3.1 Reloading or Restarting the Docker Engine

The Docker Engine must reload configuration information if any changes are made to the Docker configuration. To do this, you must restart the docker service.

If you edit the /etc/sysconfig/docker configuration file while the docker service is running, you must restart the service to make the changes take effect.

To restart the docker service, enter the following command:

```
# service docker restart
```

3.2 Enabling Non-root Users to Run Docker Commands

Warning

Users who can run Docker commands have effective root control of the system. Only grant this privilege to trusted users.

The following procedure applies to version 1.5 and later of Docker.

To enable users other than root and users with sudo access to be able to run Docker commands:

1. Create the docker group:

```
# groupadd docker
```

2. Restart the docker service:

```
# service docker restart
```

The UNIX socket /var/run/docker.sock is now readable and writable by members of the docker group.

3. Add the users that should have Docker access to the docker group:

```
# usermod -a -G docker user1 ...
```

3.3 Configuring User Namespace Remapping

The following procedure applies to version 1.10 and later of Docker.

To force processes running in Docker containers to run with an alternate user namespace mapping on the host system, use the --usersns-remap option as a startup parameter for the Docker engine. This
functionality provides an additional layer of security to the host system. The processes that are running in each container are run with the UIDs and GIDs of a subordinate mapping defined in /etc/subuid and /etc/subgid. The shadow-utils project provides subordinate user mappings, which are a function of user namespaces within the Linux kernel. For more information, see https://docs.docker.com/edge/engine/reference/commandline/dockerd/#daemon-user-namespace-options.

To implement user namespace remapping:

1. Create and edit the /etc/subuid file.

   Although the Docker documentation suggests that this file is created and populated automatically, this function is dependent on code available in the usermod command, not currently included in Oracle Linux. Create the file manually if it does not yet exist, and populate it with the user mapping that you require.

   
<table>
<thead>
<tr>
<th>user: start_uid: uid_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>dockremap: 100000: 65536</td>
</tr>
</tbody>
</table>

   Add an entry for the dockremap user if you plan to configure default user namespace remapping. Alternately, add an entry for the unprivileged user that you are going to use for this purpose. For example:

   
   dockremap: 100000: 65536

   In the example above, dockremap represents the unprivileged system user that is used for the remapping. 100000 represents the first UID in the range of available UIDs that processes within the container may run with. 65536 represents the maximum number of UIDs that may be used by a container. Based on this example entry, a process running as the root user within the container is launched so that on the host system it runs with the UID 100000. If a process within the container is run as a user with UID 500, on the host system it would run with the UID 100500.

2. Create and edit the /etc/subgid file. The same principles apply to group ID mappings as to user ID mappings.

   Add an entry for the dockremap group if you plan to configure default user namespace remapping. Alternately, add an entry for the group that you are going to use for this purpose. For example:

   
   dockremap: 100000: 65536

3. Configure the docker service to run with the --userns-remap parameter enabled, by editing /etc/sysconfig/docker and appending the relevant remap parameter to the OPTIONS line, for example:

   
   # /etc/sysconfig/docker
   # Modify these options if you want to change the way the docker daemon runs
   OPTIONS='--userns-remap=default'

   When --userns-remap is set to default, Docker automatically creates a user and group named dockremap. Entries for the dockremap user and group must exist in /etc/subuid and /etc/subgid. Alternately, set the --userns-remap option to run using another unprivileged user and group that already exist on the system. If you select to do this, replace the dockremap user in the /etc/subuid and /etc/subgid files with the appropriate user name and group name.

4. Restart the docker service to activate changes to the service configuration:

   
   # service docker restart

   The Docker engine applies the same user namespace remapping rules to all containers, regardless of who runs a container or who executes a command within a container.
Chapter 4 Working with Containers and Images

Table of Contents

4.1 Pulling Oracle Linux Images from the Docker Hub ................................................. 11
   4.1.1 Enabling or Disabling Docker Content Trust .................................................. 12
4.2 Creating and Running Docker Containers ............................................................. 12
   4.2.1 Configuring How Docker Restarts Containers .............................................. 15
   4.2.2 Controlling Capabilities and Making Host Devices Available to Containers ........... 15
   4.2.3 Accessing the Host's Process ID Namespace ............................................... 16
   4.2.4 Mounting a Host's root File System in Read-Only Mode ............................... 16
4.3 Creating a Docker Image from an Existing Container ........................................... 16
4.4 Creating a Docker Image from a Dockerfile .......................................................... 18
4.5 About Docker Networking ...................................................................................... 20
   4.5.1 About Multihost Networking ......................................................................... 21
4.6 Communicating Between Docker Containers .......................................................... 21
   4.6.1 Example of Linking Database and HTTP Server Containers ............................ 23
4.7 Accessing External Files from Docker Containers ................................................. 27
4.8 Creating and Using Data Volume Containers ......................................................... 28
4.9 Moving Data Between Docker Containers and the Host ....................................... 30
4.10 Using Labels to Define Metadata ......................................................................... 31
4.11 Defining the Logging Driver ................................................................................. 32
4.12 About Image Digests ............................................................................................ 32
4.13 Specifying Control Groups for Containers ............................................................ 32
4.14 Limiting CPU Usage by Containers ..................................................................... 32
4.15 Making a Container Use the Host's UTS Namespace ............................................ 32
4.16 Setting ulimit Values on Containers ..................................................................... 32
4.17 Building Images with Resource Constraints ....................................................... 33
4.18 Committing, Exporting and Importing Images ...................................................... 33

This chapter describes how to use the Docker Engine to run containers and how to obtain the images that are used to create a container. Other information specific to container and image configuration is also provided. In this chapter is assumed that images and containers are hosted on Oracle Linux 6.

4.1 Pulling Oracle Linux Images from the Docker Hub

An Internet connection is required to pull images from the Docker Hub.

You can obtain images for Oracle Linux for use with the Docker Engine from the oraclelinux repository at the Docker Hub. For a list of the Oracle Linux images that are available, see https://registry.hub.docker.com/_/oraclelinux/).

To download a Oracle Linux image, use the docker pull command, for example:

```bash
# docker pull oraclelinux:6
6: Pulling from library/oraclelinux
1ad695caeb02: Pull complete
1c84f45b0f092: Pull complete
Digest: sha256:cf6a9debfb397b64b7c0cc04e8696d22a1abfcfe210bd71751c630fcf8cc7ddc
Status: Downloaded newer image for oraclelinux:6
```
To display a list of the images that you have downloaded to a system, use the `docker images` command, for example:

```
[root@host ~]# docker images
REPOSITORY    TAG         IMAGE ID       CREATED       VIRTUAL SIZE
oraclelinux   6           768a3d7b605a   3 weeks ago   222.8 MB
oraclelinux   6.8         768a3d7b605a   3 weeks ago   222.8 MB
oraclelinux   6.6         9636e42b38e2   9 months ago  157.7 MB
```

Each image in a repository is distinguished by its `tag` value and its unique ID. In the following example, the tags `6` and `6.8` refer to the same image ID for Oracle Linux 6 as do the tags `7`, `7.2`, and `latest` for Oracle Linux 7.

```
[root@host ~]# docker images
REPOSITORY    TAG         IMAGE ID       CREATED       VIRTUAL SIZE
oraclelinux   6           768a3d7b605a   3 weeks ago   222.8 MB
oraclelinux   6.8         768a3d7b605a   3 weeks ago   222.8 MB
oraclelinux   latest      df602a268e64   4 weeks ago   276.1 MB
oraclelinux   7           df602a268e64   4 weeks ago   276.1 MB
oraclelinux   7.2         df602a268e64   4 weeks ago   276.1 MB
```

When new images are made available for Oracle Linux updates, the tags `6`, `7`, and `latest` are updated in the `oraclelinux` repository to refer to the appropriate newest version.

### 4.1.1 Enabling or Disabling Docker Content Trust

Docker version 1.8 introduces Content Trust, which allows you to verify the authenticity, integrity, and publication date of Docker images that are made available on the Docker Hub Registry.

By default, Content Trust is disabled. To enable Content Trust for signing and verifying Docker images that you build, push to, or pull from the Docker Hub, set the `DOCKER_CONTENT_TRUST` environment variable, for example:

```
# export DOCKER_CONTENT_TRUST=1
```

If you use `sudo` to run Docker commands, specify the `-E` option to preserve the environment or use `visudo` to add the following line to `/etc/sudoers`:

```
Defaults        env_keep += "DOCKER_CONTENT_TRUST"
```

For individual `docker build`, `docker push`, or `docker pull` commands, you can specify the `--disable-content-trust=false` and `--disable-content-trust=true` options to enable or disable Content Trust.

For more information, see [https://blog.docker.com/2015/08/content-trust-docker-1-8/](https://blog.docker.com/2015/08/content-trust-docker-1-8/) and [https://docs.docker.com/engine/security/trust/content_trust/](https://docs.docker.com/engine/security/trust/content_trust/).

### 4.2 Creating and Running Docker Containers

You use the `docker run` command to run an application inside a container, for example:

```
[root@host ~]# docker run -i -t --name guest oraclelinux:6.8 /bin/bash
[root@guest ~]# cat /etc/oracle-release
Oracle Linux Server release 6.8
[root@guest ~]#
```

This example runs an interactive `bash` shell using the Oracle Linux 6 image named `oraclelinux:6.8` to provide the container. The `-t` and `-i` options allow you to use a pseudo-terminal to run the container.
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Creating and Running Docker Containers

interactively. `{root@host ~}` and `{root@guest ~}` represent the prompts shown by the host and by the container respectively. The actual prompt displayed by the container might be different.

The `--name` option specifies the name `guest` for the container instance. Docker does not remove the container when it exits and we can restart it at a later time.

If an image does not already exist on your system, the Docker Engine performs a `docker pull` operation to download the image from the Docker Hub (or from another repository that you specify) as shown in the following example:

```
[root@host ~]# docker run -i -t --name oraclelinux:7.0
Unable to find image 'oraclelinux:7.0' locally
Pulling repository oraclelinux
073ded22ac0f: Download complete
51136ea3c5a: Download complete
ad98bd7101f2: Download complete
cba192df7f4b: Download complete
Status: Downloaded newer image for oraclelinux:7.0
[root@guest ~]# cat /etc/oracle-release
Oracle Linux Server release 7.0
[root@guest ~]# exit
exit
[root@host ~]#
```

Because we specified the `--rm` option instead of naming the container, Docker removes the container when it exits and we cannot restart it.

From another shell window, you can use the `docker ps` command to display information about the containers that are currently running, for example:

```
[root@host ~]# docker ps
CONTAINER ID        IMAGE              COMMAND                  CREATED          STATUS          PORTS               NAMES
768a3d7b605a        oraclelinux:6.8   /bin/bash                 14 minutes ago   Up 14 minutes  guest
```

The container named `guest` with the ID `77bacba845e2` is currently running the command `/bin/bash`. It is more convenient to manage a container by using its name than by its ID.

To display the processes that a container is running, use the `docker top` command:

```
[root@host ~]# docker top guest
UID   PID   PPID  C   STIME     TTY      TIME   CMD
root  7474   1958   1   15:40   pts/2    00:00:00 /bin/bash
```

In version 1.3.0 and later of Docker, you can use the `docker exec` command to run additional processes in a container that is already running, for example:

```
[root@host ~]# docker exec -i -t guest bash
[root@guest ~]#
```

In version 1.3.0 and later of Docker, you can use the `docker create` command to set up a container that you can start at a later time, for example:

```
[root@host ~]# docker create -i -t --name newguest oraclelinux:6 /bin/bash
af621dc988b019a4e8b58c5e9f95e26dd18c05c983761d5b8c7c046fcbf1176e0
[root@host ~]# docker start -a -i newguest
[root@newguest ~]#
```

The `-a` and `-i` options to `docker start` attach the current shell’s standard input, output, and error streams to those of the container and also cause all signals to be forwarded to the container.
You can exit a container by typing Ctrl-D or exit at the bash command prompt inside the container or by using the docker stop command:

```
[root@host ~]# docker stop guest
```

The -a option to docker ps displays all containers that are currently running or that have exited.

```
[ root@host ~ ] # docker ps -a
CONTAINER ID  IMAGE              COMMAND    CREATED  STATUS             PORTS  NAMES
768a3d7b605a  oraclelinux:6.8    ...        ...      Exited (0) 9 seconds ago  guest
8a1b9b19bb70  oraclelinux:6.6    ...        ...      Up 38 seconds   ...       newguest
```

You can use docker start to restart a stopped container. After reattaching to it, the contents remain unchanged from the last time that you used the container.

```
[root@host ~]# docker start -a -i guest
[root@guest ~]# touch /tmp/foobar
[root@guest ~]# exit
[root@host ~]# docker start -a -i guest
[root@guest ~]# ls -l /tmp/foobar
-rw-r--r--. 1 root root 0 Aug 29 05:23 /tmp/foobar
```

Because the container preserves any changes that you make to it, you can reconfigure files and install packages in the container without worrying that your changes will disappear.

If you need to remove a container permanently so that you can create a new container with the same name, use the docker rm command:

```
[root@host ~]# docker rm guest
```

Note

If you specify the --rm option when you run a container, Docker removes the container when the container exits. You cannot combine the --rm option with the -d option.

In version 1.2.0 and later of Docker, specifying the -f option to docker rm kills a running container before removing it. In previous versions, the same command stops the container before removing it. If you want to stop a container safely, use docker stop.

You can use the docker logs command to watch what is happening inside a container, for example:

```
[root@host ~]# docker logs -f guest
... bash-4.x# touch /tmp/foobar
bash-4.x# exit
exit bash-4.x#
bash-4.x# ls -l /tmp/foobar
-rw-r--r--. 1 root root 0 Aug 29 05:23 /tmp/foobar
```

The -f option causes the command to update its output as events happen in the container. Type Ctrl-C to exit the command.

You can obtain full information about a container in JSON format by using the docker inspect command. This command also allows you to retrieve specified elements of the configuration, for example:

```
[root@host ~]# docker inspect --format='{{ .State.Running }}' guest
```
4.2.1 Configuring How Docker Restarts Containers

To specify how you want Docker to handle a container when it exits, you can use the `--restart` option with `docker run` in version 1.2.0 and later of Docker and with `docker create` in version 1.3.0 and later:

- `--restart=always` Docker always attempts to restart the container when the container exits.
- `--restart=no` Docker does not attempt to restart the container when the container exits. This is the default policy.
- `--restart=on-failure[:max-retry]` Docker attempts to restarts the container if the container returns a non-zero exit code. You can optionally specify the maximum number of times that Docker will try to restart the container.

4.2.2 Controlling Capabilities and Making Host Devices Available to Containers

If you specify the `--privileged=true` option to `docker create` or `docker run`, the container has access to all the devices on the host, which can present a security risk. For more precise control, you can use the `--cap-add` and `--cap-drop` options in version 1.2.0 and later of Docker to restrict the capabilities of a container, for example:

```
[root@host ~]# docker run --cap-add=ALL --cap-drop=NET_ADMIN -i -t --rm oraclelinux:6 /bin/bash
```

This example grants all capabilities except `NET_ADMIN` to the container so that it is not able to perform network-administration operations. For more information, see the `capabilities(7)` manual page.

To make only individual devices on the host available to a container, you can use the `--device` option with `docker run` in version 1.2.0 and later of Docker and with `docker create` in version 1.3.0 and later:

```
--device=host_devname[:container_devname[:permissions]]
```

`host_devname` is the name of the host device.

`container_devname` is an optional name for the name of the device in the container.

`permissions` optionally specifies the permissions that the container has on the device, which is a combination of the following codes:

- `m` Grants `mknod` permission. For example, you can use `mknod` to set permission bits or the SELinux context for the device file.
- `r` Grants read permission.
- `w` Grants write permission. For example, you can use a command such as `mkfs` to format the device.
4.2.3 Accessing the Host's Process ID Namespace

In version 1.5 and later of Docker, you can make the host's process ID namespace visible from inside a container by specifying the `--pid=host` option to `docker run`. A suggested use of this mode is to debug host processes by using containerized debugging tools.

**Warning**

Host mode is inherently insecure as it gives a container full access to D-Bus and other system services on the host.

4.2.4 Mounting a Host's root File System in Read-Only Mode

In version 1.5 and later of Docker, you can mount the host's root file system in read-only mode from a container by specifying the `--read-only=true` option to `docker create` or `docker run`. You can use this mode to restrict write access by a containerized application.

4.3 Creating a Docker Image from an Existing Container

If you modify the contents of a container, you can use the `docker commit` command to save the current state of the container as an image.

The following example demonstrates how to modify an container based on the `oraclelinux:6.6` image so that it can run an Apache HTTP server. After stopping the container, the image `mymod/httpd:v1` is created from it.

To create an Apache server image from an `oraclelinux:6.6` container:

1. Run the `bash` shell inside a container named `guest`:

   ```bash
   [root@host ~]# docker run -i -t --name guest oraclelinux:6.6 /bin/bash
   [root@guest ~]#
   ```


3. Install the `httpd` package:

   ```bash
   [root@guest ~]# yum install httpd
   ```

4. If required, create the web content to be displayed under the `/var/www/html` directory hierarchy on the guest.

5. Exit the guest by using the `docker stop` command on the host:

   ```bash
   [root@host ~]# docker stop guest
   ```
6. Create the image `mymod/httpd` with the tag `v1` using the ID of the container that you stopped:

   ```bash
   [root@host ~]# docker commit -m "ol6 + httpd" -a "A N Other" \
   `docker ps -l -q` mymod/httpd:v1
   8594abec905e6374db51bed1bf2b08804cfb60d96b285efb897db581a01e76e9
   ```

   Use the `-m` and `-a` options to document the image and its author. The command returns the full version of the new image's ID.

   If you use the `docker images` command, the new image now appears in the list:

   ```bash
   [root@host ~]# docker images
   REPOSITORY    TAG         IMAGE ID       CREATED       VIRTUAL SIZE
   mymod/httpd   v1          8594abec905e   2 minutes ago 938.5 MB
   oraclelinux   6           9ac13076d2b5   5 days ago    319.4 MB
   oraclelinux   6.6         9ac13076d2b5   5 days ago    319.4 MB
   oraclelinux   latest      073ded22ac0f   5 days ago    265.2 MB
   oraclelinux   7           073ded22ac0f   5 days ago    265.2 MB
   oraclelinux   7.0         073ded22ac0f   5 days ago    265.2 MB
   ```

7. Remove the container named `guest`.

   ```bash
   # docker rm guest
   ```

   You can now use the new image to create a container that works as a web server, for example:

   ```bash
   # docker run -d --name newguest -p 8080:80 mymod/httpd:v1 /usr/sbin/httpd -D FOREGROUND
   7afbbefec5191f632e149f85ae10ed0ba88f1c545daad18cb930e575ef6a3e63
   ```

   The `-d` option runs the command non-interactively in the background and displays the full version of the unique container ID. The `-p 8080:80` option maps port 80 in the guest to port 8080 on the host. You can view the port mapping by running `docker ps` or `docker port`, for example:

   ```bash
   [root@host ~]# docker ps
   CONTAINER ID  IMAGE         COMMAND    CREATED    STATUS   PORTS                  NAMES
   7afbbefec519  mymod/httpd:v1           ...        ...      0.0.0.0:8080->80/tcp   newguest
   [root@host ~]# docker port newguest 80
   0.0.0.0:8080
   ```

   **Note**

   The `docker ps` command displays the short version of the container ID. You can use the `--no-trunc` option to display the long version.

   The default IP address value of 0.0.0.0 means that the port mapping applies to all network interfaces on the host. You can restrict the IP addresses to which the remapping applies by using multiple `-p` options, for example:

   ```bash
   # docker run -d --name newguest -p 127.0.0.1:8080:80 -p 192.168.1.2:8080:80 \
   mymod/httpd:v1 /usr/sbin/httpd -D FOREGROUND
   ```

   You can view the web content served by the guest by pointing a browser at port 8080 on the host. If you access the content from a different system, you might need to allow incoming connections to the port on the host, for example:

   ```bash
   [root@host ~]# iptables -I INPUT -p tcp -m state --state NEW --m tcp --dport 8080 -j ACCEPT
   [root@host ~]# service iptables save
   ```

   If you need to remove an image, use the `docker rmi` command:
Creating a Docker Image from a Dockerfile

18

\[
\text{[root@host ~]# docker rmi mymod/httpd:v1}
\]
Untagged: mymod/httpd:v1
Deleted: 7afbbefec519f632e149f85ae10ed0ba88f1c545daad18cb930e575ef6a3e63

Note
From version 1.8 of Docker, you cannot remove the image of a running container.

In a production environment, using the `docker commit` command to create an image does not provide a convenient record of how you created the image so you might find it difficult to recreate an image that has been lost or become corrupted. The preferred method for creating an image is to set up a Dockerfile, in which you define instructions that allow Docker to build the image for you. See Section 4.4, “Creating a Docker Image from a Dockerfile”.

4.4 Creating a Docker Image from a Dockerfile

You use the `docker build` command to create a Docker image from the definition contained in a Dockerfile.

The following example demonstrates how to build an image named `mymod/httpd` with the tag `v2` based on the `oraclelinux:6` image so that it can run an Apache HTTP server.

To create a Docker image from a Dockerfile:

1. Make a directory where you can create the Dockerfile, for example:

\[
# mkdir -p /var/docker_projects/mymod/httpd
\]

Note
You do not need to create the Dockerfile on the same system on which you want to deploy containers that you create from the image. The only requirement is that the Docker Engine can access the Dockerfile.

2. In the new directory, create the Dockerfile, which is usually named `Dockerfile`. The following Dockerfile contents are specific to the example:

\[
\text{# Dockerfile that modifies oraclelinux:6 to include an Apache HTTP server}
\]

\[
\text{FROM oraclelinux:6}
\]

\[
\text{MAINTAINER A N Other <another@example.com>}
\]

\[
\text{RUN sed -i -e '/^\[main\]/aproxy=http://proxy.example.com:80' /etc/yum.conf}
\]

\[
\text{RUN yum -y install httpd}
\]

\[
\text{RUN echo "HTTP server running on guest" > /var/www/html/index.html}
\]

\[
\text{EXPOSE 80}
\]

\[
\text{ENTRYPOINT /usr/sbin/httpd -D FOREGROUND}
\]

The `#` prefix in the first line indicates that the line is a comment. The remaining lines start with the following instruction keywords that define how Docker creates the image:

\[
\text{ENTRYPOINT Specifies the command that a container created from the image always runs. In this example, the command is /usr/sbin/httpd -D FOREGROUND, which starts the HTTP server process.}
\]

\[
\text{EXPOSE Defines that the specified port is available to service incoming requests. You can use the -p or -P options with docker run to map this port to another port on the host. Alternatively, you can use the --link option with docker run to allow another container}
\]
Creating a Docker Image from a Dockerfile

3. Use the `docker build` command to create the image:

```bash
# docker build --tag=mymod/httpd:v2 /var/docker_projects/mymod/httpd/
```

Sending build context to Docker daemon 2.048 kB
Step 1 : FROM oraclelinux:6
 ---> 768a3d7b605a
Step 2 : MAINTAINER A N Other <another@example.com>
 ---> Running in bc28c7b6babf
 ---> 0eb554ad6a7b
Removing intermediate container bc28c7b6babf
Step 3 : RUN sed -i -e '/^\[main\]/aproxy=http://proxy.example.com:80' /etc/yum.conf
 ---> Running in 4bd1305ab0fc
 ---> 328ff7b80dc0
Removing intermediate container 4bd1305ab0fc
Step 4 : RUN yum -y install httpd
 ---> Running in 991bcace0d34
Loaded plugins: security, ulninfo
Setting up Install Process
Resolving Dependencies
 ---> Running transaction check
---> Package httpd.x86_64 0:2.2.15-53.0.1.el6 will be installed
---> Processing Dependency: httpd-tools = 2.2.15-53.0.1.el6 for package: httpd-2.2.15-53.0.1.el6.x86_64
...
Installed:
  httpd.x86_64 0:2.2.15-53.0.1.el6
Dependency Installed:
  apr.x86_64 0:1.3.9-5.el6_2
  apr-util.x86_64 0:1.3.9-3.el6_0.1
  apr-util-ldap.x86_64 0:1.3.9-3.el6_0.1
  httpd-tools.x86_64 0:2.2.15-53.0.1.el6
  mailcap.noarch 0:2.1.31-2.el6
Complete!
 ---> 153b5fa6ba92
Removing intermediate container 991bcace0d34
Step 5 : RUN echo "HTTP server running on guest" > /var/www/html/index.html
 ---> Running in 0a9d54558627
 ---> 6409e3ae1e1e
Removing intermediate container 0a9d54558627
Step 6 : EXPOSE 80
 ---> Running in da51ae63fbe0
 ---> d16342a6b776
Removing intermediate container da51ae63fbe0
Step 7 : ENTRYPOINT /usr/sbin/httpd -D FOREGROUND
 ---> Running in 211988caab1c
 ---> 6df513b1c3d1
Removing intermediate container 211988caab1c
Successfully built 6df513b1c3d1
Having built the image, you can test it by creating a container instance named `newguest2`:

```
[root@host ~]# docker run -d --name newguest2 -P mymod/httpd:v2
31b334b9933cfbecc71d7be47723c352c8de842823505b611a08bf960e398e7
```

**Note**

You do not need to specify `/usr/sbin/httpd -D FOREGROUND` as this command is now built into the container.

The `-P` option specifies that Docker should map the ports exposed by the guest to available ports in the range 49000 through 49900 on the host.

You can use `docker inspect` to return the host port that Docker maps to TCP port 80:

```
[root@host ~]# docker inspect --format='{{ .NetworkSettings.Ports }}' newguest2
map[80/tcp:[map[HostIp:0.0.0.0 HostPort:49153]]]
```

In this example, TCP port 80 in the guest is mapped to TCP port 49153 on the host.

You can view the web content served by the guest by pointing a browser at port 49153 on the host. If you access the content from a different system, you might need to allow incoming connections to the port on the host.

For example you can open the port by creating an `iptables` rule:

```
[root@host ~]# iptables -I INPUT -p tcp -m state --state NEW -m tcp --dport 49153 -j ACCEPT
[root@host ~]# service iptables save
```

You can also use `curl` to test that the server is working:

```
[root@host ~]# curl http://localhost:49153
HTTP server running on guest
```

For the `newguest2` example:

```
Last login: Fri Aug 29 13:48:58 2014 from 192.168.0.1
```

You can also use `curl` to test that the server is working:

```
[root@host ~]# ssh auser@10.0.0.23
Password
auser@10.0.0.23's password: password
Last login: Fri Aug 29 13:48:58 2014 from 192.168.0.1
```

```
Last login: Fri Aug 29 13:48:58 2014 from 192.168.0.1
```

```
HTTP server running on guest
```

### 4.5 About Docker Networking

The Docker networking features allow you to create secure networks of web applications that can communicate while running in separate containers. By default, Docker configures two types of network (as displayed by the `docker network ls` command):

- **host**
  - If you specify the `--net=host` option to the `docker create` or `docker run` commands, Docker uses the host's network stack for the container. The network configuration of the container is the same as that of the host and the container shares the service ports that are available to the host. This configuration does not provide any network isolation for a container.

- **bridge**
  - By default, Docker attaches containers to a bridge network named `bridge`. When you run a command such as `ip link show` on the host, the bridge is visible as the `docker0` network interface. You can use the bridge network to connect separate application containers. The `docker network inspect bridge` command allows you to examine the network configuration of the bridge, which is displayed in JSON format. Docker sets up a default subnet address, network mask,
and gateway for the bridge network and automatically assigns subnet addresses to containers that you add to the bridge network.

The default bridge network does not support automatic service discovery. To connect the containers on the network, you can use the `docker run --link` command. See Section 4.6, “Communicating Between Docker Containers”.

A container can communicate with other containers on a bridge network but not with other networks unless you also attach it to those networks. To define the networks that a container should use, specify a `--net=bridge-network-name` option for each network to the `docker create` or `docker run` commands. To attach a running container to a network, you can use the `docker network connect network-name container-name` command.

You can use the `docker network create --driver bridge bridge-network-name` command to create user-defined bridge networks that expose container network ports that can be accessed by external networks and other containers. You specify `--net=bridge-network-name` to `docker create` or `docker run` to attach the container to this network. User-defined bridge networks do not support linking by using the `docker run --link` command.

For more information, see https://docs.docker.com/engine/userguide/networking/.

### 4.5.1 About Multihost Networking

A bridge network provides network isolation but it limits container connections to a single host system unless you use a complex user-defined bridge. Docker version 1.9 includes the VXLAN-based `overlay` network driver that supports multihost networking, where you can attach separate application containers running on multiple Docker hosts to the same virtual overlay network. Before you can create an overlay network, you must configure a key-value (KV) service such as Consul, Etcd, or ZooKeeper that the Docker hosts can access to share configuration information. You can then configure the Docker daemon on each host to access the KV server by specifying appropriate values to the `--cluster-advertise` and `--cluster-store` options. Next you use the `docker network create --driver overlay multihost-network-name` command on one of the hosts to create the overlay network. Having created the overlay network, you can attach the container to this network by specifying `--net=multihost-network-name` to `docker create` or `docker run`.

For more information, see https://docs.docker.com/engine/userguide/networking/dockernetworks.

### 4.6 Communicating Between Docker Containers

You can use the `--link` option with `docker run` to make network connection information about a server container available to a client container. The client container uses a private networking interface to access the exposed port in the server container. Docker sets environment variables about the server container in the client container that describe the interface and the ports that are available.

The following example demonstrates how to link an `oraclelinux:6`-based client container with an HTTP server container based on the `mymod/httpd:v2` image that you created in Section 4.4, “Creating a Docker Image from a Dockerfile”.

For more information, see https://docs.docker.com/engine/userguide/networking/dockernetworks.
To create an HTTP server and client containers that are linked:

1. Create an HTTP server container named `http_server`:

```
[root@host ~]# docker run -d --name http_server mymmod/httpd:v2
```

```
a4716915422329eed66762128755cd9fdd24d0f27ff8e0f678ef136bcc66d03
```

2. Create a client container named `client1` that runs the `bash` shell and is linked to the `http_server` container:

```
[root@host httpd]# docker run --rm -t -i --name client1 --link http_server:server \
/oraclelinux:6 /bin/bash
```

```
[root@client1 ~]#
```

The argument `http_server:server` to the `--link` option aliases the name `http_server` as `server`. Docker converts the alias to upper case (`SERVER`) and uses this string when setting up the names of the environment variables on the client.

You can now view the environment variables in the `client1` container. You can also use `ping` to detect the server container by name or IP address, and use `curl` to access the web server running on the server:

```
[root@client1 ~]# env
HOSTNAME=10815c22e5b4
TERM=xterm
SERVER_PORT=tcp://172.17.0.16:80
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/
SERVER_PORT_80_TCP_PORT=80
SERVER_PORT_80_TCP_ADDR=172.17.0.16
SERVER_PORT_80_TCP=tcp://172.17.0.16:80
SERVER_PORT_80_TCP_PROTO=tcp
SHLVL=1
SERVER_NAME=/client1/server
HOME=/
_/=/usr/bin/env
```

```
[root@client1 ~]# ping -c 1 server
PING server (172.17.0.16) 56(84) bytes of data.
64 bytes from server (172.17.0.16): icmp_seq=1 ttl=64 time=0.105 ms
```

```
--- server ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.105/0.105/0.105/0.000 ms
```

```
[root@client1 ~]# ping -c 1 172.17.0.16
PING 172.17.0.16 (172.17.0.16) 56(84) bytes of data.
64 bytes from 172.17.0.16: icmp_seq=1 ttl=64 time=0.171 ms
```

```
--- 172.17.0.16 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.171/0.171/0.171/0.000 ms
```

```
[root@client1 ~]# curl http://server
HTTP server running on guest
```

```
[root@client1 ~]# curl http://172.17.0.16
HTTP server running on guest
```

You can start multiple client container instances with different names, each of which can access port 80 on the server container. Docker assigns a different IP address to each client. As shown in the following example output, Docker creates an entry for the server in the `/etc/hosts` files on each client but it does not create entries for the names of the client containers themselves:

```
[root@client1 ~]# cat /etc/hosts
172.17.0.17  10815c22e5b4
127.0.0.1  localhost
```

22
Example of Linking Database and HTTP Server Containers

By default, the clients are visible to each other on the private network only by their IP addresses.

The `docker ps` command shows the containers that are running:

```
[root@host ~]# docker ps
```

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
<th>CREATED</th>
<th>STATUS</th>
<th>PORTS</th>
<th>NAMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>449abeac3041</td>
<td>oraclelinux:6</td>
<td>/bin/bash</td>
<td>...</td>
<td>Up 1 minutes</td>
<td></td>
<td>client2</td>
</tr>
<tr>
<td>10815c22e5b4</td>
<td>oraclelinux:6</td>
<td>/bin/bash</td>
<td>...</td>
<td>Up 2 minutes</td>
<td></td>
<td>client1</td>
</tr>
<tr>
<td>a47169154222</td>
<td>mymod/httpd:v2</td>
<td>/usr/sbin/httpd</td>
<td>...</td>
<td>Up 3 minutes</td>
<td>80/tcp</td>
<td>client1/server,client2/server,http_server</td>
</tr>
</tbody>
</table>

The `NAMES` column shows that `http_server` is linked to `client1` and `client2` as server. The `PORTS` column shows that Docker has not remapped TCP port 80 on `http_server` to another port on the host.

4.6.1 Example of Linking Database and HTTP Server Containers

Note
This simple example demonstrates how to link containers. You should not use it as the basis of a production application.

The following example demonstrates how to link a container that is running a MySQL server with a container running an HTTP server.

First of all, we define a Dockerfile for the MySQL server, which we place in the `/var/docker_projects/mymod/mysql` directory:

```
FROM oraclelinux:6
ENV http_proxy http://proxy.example.com:80
RUN yum install -y mysql-server
ADD my.cnf /etc/my.cnf
ADD run.sh /opt/run.sh
RUN chmod 744 /opt/run.sh
ENTRYPOINT /opt/run.sh
```

The instruction keywords define how to create the image:

ADD Copy the files `my.cnf` and `run.sh` from the `/var/docker_projects/mymod/mysql` directory to `/etc/my.cnf` and `/opt/run.sh` in the container.

ENTRYPOINT Specify that the container always runs `/opt/run.sh`.

ENV Define the web proxy in the build environment (as an alternative to modifying `/etc/yum.conf`).
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Example of Linking Database and HTTP Server Containers

```bash
FROM oraclelinux:6

RUN Install the mysql-server package and make the /opt/run.sh script executable.

The my.cnf file in /var/docker_projects/mymod/mysql contains the database configuration:

```
[mysqld]
bind-address=0.0.0.0
console=1
general_log=1
general_log_file=/dev/stdout
log_error=/dev/stderr
collation-server=utf8_unicode_ci
collation-client=utf8
character-set-server=utf8
datadir=/var/lib/mysql
```

The run.sh file in /var/docker_projects/mymod/mysql contains the shell script for starting the database:

```
#!/bin/bash
chown -R mysql:mysql /var/lib/mysql
mysql_install_db --user=mysql > /dev/null
/usr/libexec/mysqld --user mysql --bootstrap << SQL
  FLUSH PRIVILEGES;
  GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION;
  CREATE USER dbuser IDENTIFIED BY 'secret';
  CREATE DATABASE MYDB;
  USE MYDB;
  GRANT ALL ON MYDB.* to 'dbuser'@'%';
SQL
/usr/bin/mysqld_safe --user mysql
```

Having set up the Dockerfile and the files my.cnf and run.sh, we can now build the image mymod/mysql:v1 and create an instance of this container named db that uses the standard MySQL connection port (3306):

```
# docker build --tag="mymod/mysql:v1" /var/docker_projects/mymod/mysql/
```

We next define a Dockerfile for the HTTP server, which we place in the /var/docker_projects/mymod/httpd2 directory:
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Example of Linking Database and HTTP Server Containers

```bash
FROM oraclelinux:6
ENV http_proxy http://proxy.example.com:80
RUN yum install -y httpd perl perl-DBI.x86_64 libdbi-db-dbd-mysql.x86_64 perl-DBD-MySQL.x86_64
ADD version.pl /var/www/cgi-bin/version.pl
ADD initdb.pl /var/www/cgi-bin/initdb.pl
ADD doquery.pl /var/www/cgi-bin/doquery.pl
RUN sed -i -e '/<Directory "/var/www/cgi-bin">/,/</Directory>/c\n    <Directory="/var/www/cgi-bin"/>
    Options +ExecCGI
    AddHandler cgi-script .pl .cgi
</Directory>' /etc/httpd/conf/httpd.conf
EXPOSE 80
ENTRYPOINT /usr/sbin/httpd -D FOREGROUND

This Dockerfile modifies the container's HTTP server configuration file (/etc/httpd/conf/httpd.conf) to allow the use of CGI scripts and installs the following Perl scripts from the /var/docker_projects/mymod/httpd2 directory:

version.pl

Connects to the database and returns its version.

```perl
#!/usr/bin/perl
use DBI;
print "Content-type: text/html

";
my $dbh = DBI->connect("dbi:mysql:dbname=MYDB:host=db",
"dbuser",
"secret",
{ RaiseError => 1 },
) or die $DBI::errstr;

my $sth = $dbh->prepare("SELECT VERSION()");
$sth->execute();
my $ver = $sth->fetch();
print "Version = ", @$ver, "\n"
$sth->finish();
$dbh->disconnect();
```

initdb.pl

Sets up the database and populates a table with several entries.

```perl
#!/usr/bin/perl
use strict;
use DBI;
print "Content-type: text/html

";
my $dbh = DBI->connect("dbi:mysql:dbname=MYDB:host=db",
"dbuser",
"secret",
{ RaiseError => 1 },
) or die $DBI::errstr;

$dbh->do("DROP TABLE IF EXISTS PEOPLE");
$dbh->do("CREATE TABLE People(Id INT PRIMARY KEY, Name TEXT, Age INT) ENGINE=InnoDB");
$dbh->do("INSERT INTO People VALUES(1,'Alice',42)");
$dbh->do("INSERT INTO People VALUES(2,'Bobby',27)");
$dbh->do("INSERT INTO People VALUES(3,'Carol',29)");
$dbh->do("INSERT INTO People VALUES(4,'Daisy',20)");
$dbh->do("INSERT INTO People VALUES(5,'Eddie',35)");
```

This Dockerfile modifies the container's HTTP server configuration file (/etc/httpd/conf/httpd.conf) to allow the use of CGI scripts and installs the following Perl scripts from the /var/docker_projects/mymod/httpd2 directory:

version.pl

Connects to the database and returns its version.

```perl
#!/usr/bin/perl
use DBI;
print "Content-type: text/html

";
my $dbh = DBI->connect("dbi:mysql:dbname=MYDB:host=db",
"dbuser",
"secret",
{ RaiseError => 1 },
) or die $DBI::errstr;

my $sth = $dbh->prepare("SELECT VERSION()");
$sth->execute();
my $ver = $sth->fetch();
print "Version = ", @$ver, "\n"
$sth->finish();
$dbh->disconnect();
```

initdb.pl

Sets up the database and populates a table with several entries.

```perl
#!/usr/bin/perl
use strict;
use DBI;
print "Content-type: text/html

";
my $dbh = DBI->connect("dbi:mysql:dbname=MYDB:host=db",
"dbuser",
"secret",
{ RaiseError => 1 },
) or die $DBI::errstr;

$dbh->do("DROP TABLE IF EXISTS PEOPLE");
$dbh->do("CREATE TABLE People(Id INT PRIMARY KEY, Name TEXT, Age INT) ENGINE=InnoDB");
$dbh->do("INSERT INTO People VALUES(1,'Alice',42)");
$dbh->do("INSERT INTO People VALUES(2,'Bobby',27)");
$dbh->do("INSERT INTO People VALUES(3,'Carol',29)");
$dbh->do("INSERT INTO People VALUES(4,'Daisy',20)");
$dbh->do("INSERT INTO People VALUES(5,'Eddie',35)");
```
Example of Linking Database and HTTP Server Containers

```perl
$dbh->do("INSERT INTO People VALUES(6,'Frank',21)");

my @noerr = ('Rows inserted in People table');
print @noerr;
print "\n";

my $sth = $dbh->prepare( "SELECT * FROM People" );
$sth->execute();
for ( 1 .. $sth->rows() ) {
    my ($id, $name, $age) = $sth->fetchrow();
    print "$id $name $age\n";
}

$sth->finish();
$dbh->disconnect();
```

doquery.pl

Performing a simple query on the database, using the command argument as data for the query.

```perl
#!/usr/bin/perl
use strict;
use DBI;

print "Content-type: text/html\n\n";

my $dbh = DBI->connect(
    "dbi:mysql:dbname=MYDB;host=db",
    "dbuser",
    "secret",
    { RaiseError => 1 },
) or die $DBI::errstr;

my $sth = $dbh->prepare( "SELECT * FROM People WHERE Age > $ARGV[0]" );
$sth->execute();
my $fields = $sth->{NUM_OF_FIELDS};
my $rows = $sth->rows();
print "Selected $rows row(s) with $fields field(s)\n";
for ( 1 .. $rows ) {
    my ($id, $name, $age) = $sth->fetchrow();
    print "$id $name $age\n";
}

$sth->finish();
$dbh->disconnect();
```

Having set up the Dockerfile and the Perl scripts, we can now build the image `mymod/httpd:v3` and create an instance of this container named `web`, which is linked to the `db` container and which uses the standard HTTP server port (80) on the host:

```
# docker build --tag="mymod/httpd:v3" /var/docker_projects/mymod/httpd2/
```

Uploading context 142.8 kB
Uploading context
Step 0 : FROM oraclelinux:6
    ---> d56e767abb61
Step 1 : ENV http_proxy http://proxy.example.com:80
    ---> Using cache
    ---> f92df8c449eb
...
Step 11 : ENTRYPOINT /usr/sbin/httpd -D FOREGROUND
    ---> Running in 3203c57a7204
    ---> 10dc2d7624d3
Removing intermediate container 3203c57a7204
4.7 Accessing External Files from Docker Containers

You can use the -v option with docker run to make a file or file system available inside a container. The following example demonstrates how to make web pages on the host available to an HTTP server running in a container.

Create the file /var/www/html/index.html on the host and run an HTTP server container that mounts this file:

```
[root@host ~]# echo "This text was created in a file on the host" > /var/www/html/index.html
[root@host ~]# docker run -d --name newguest3 -P -v /var/www/html/index.html:/var/www/html/index.html:ro mymod/httpd:v2
```

The :ro modifier specifies that a container mounts a file or file system read-only. To mount a file or file system read-writable, specify the :rw modifier instead or omit the modifier altogether.

Check that the HTTP server is not running on the host:

```
[root@host ~]# curl http://localhost
curl: (7) couldn't connect to host
[root@host ~]# service httpd status
httpd is stopped
```

Even though an HTTP server is not running directly on the host, you can display the new web page served by the newguest3 container:

```
[root@host ~]# docker inspect --format='{{ .NetworkSettings.Ports }}' newguest3
map[80/tcp:[map[HostIp:0.0.0.0 HostPort:49153]]]
[root@host ~]# curl http://localhost:49153
This text was created in a file on the host
```

Any changes that you make to the /var/www/html/index.html file on the host are reflected in the mounted file in the container:
Creating and Using Data Volume Containers

If you specify a single directory argument to the `-v` option of `docker run`, Docker creates the directory in the container and marks it as a *data volume* that other containers can mount. You can also use the `VOLUME` instruction in a Dockerfile to create this data volume in an image. A container that contains such a data volume is called a data volume container. After populating the data volume with files, you can use the `--volumes-from` option of `docker run` to have other containers mount the volume and access its data.

### Note

When you use `docker rm` to remove a container that has associated data volumes, specify the `-v` option to remove these volumes. Unassociated volumes waste disk space and are difficult to remove.

The following example creates a data volume container that an HTTP server container can use as the source of its web content.

To create a data volume container image and an instance of a data volume container from this image:

1. Make a directory where you can create the Dockerfile for the data volume container image, for example:

   ```
   # mkdir -p /var/docker_projects/mymod/dvc
   ```

2. In the new directory, create a Dockerfile that defines the image for a data volume container:

   ```
   # Dockerfile that modifies oraclelinux:6 to create a data volume container
   FROM oraclelinux:6
   MAINTAINER A N Other <another@example.com>
   RUN mkdir -p /var/www/html
   RUN echo "This is the content for file1.html" > /var/www/html/file1.html
   RUN echo "This is the content for file2.html" > /var/www/html/file2.html
   RUN echo "This is the content for index.html" > /var/www/html/index.html
   VOLUME /var/www/html
   ENTRYPOINT /usr/bin/tail -f /dev/null
   ```

   The `RUN` instructions create a `/var/www/html` directory that contains three simple files.

   The `VOLUME` instruction makes the directory available as a volume that other containers can mount by using the `--volumes-from` option to `docker run`.
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Creating and Using Data Volume Containers

The `ENTRYPOINT` instruction specifies the command that a container created from the image always runs. To prevent the container from exiting, the `/usr/bin/tail -f /dev/null` command blocks until you use a command such as `docker stop dvc1` to stop the container.

3. Use the `docker build` command to create the image:

```bash
[root@host ~]# docker build --tag="mymod/dvc:v1" /var/docker_projects/mymod/dvc/
```

```
Uploading context  2.56 kB
Uploading context
Step 0 : FROM oraclelinux:6
  ---> 3e4b5e722ab9
Step 1 : MAINTAINER A N Other <another@example.com>
  ---> Using cache
  ---> debe47ce9b8
Step 2 : RUN mkdir -p /var/www/html
  ---> Running in fa94df7dd3af
  ---> 503132e87939
Removing intermediate container fa94df7dd3af
Step 3 : RUN echo "This is the content for file1.html" > /var/www/html/file1.html
  ---> Running in f98a14371672
  ---> e63ba0d36d88
Removing intermediate container f98a14371672
Step 4 : RUN echo "This is the content for file2.html" > /var/www/html/file2.html
  ---> Running in d0dca96ad53c
  ---> 27f2e2b3b207
Removing intermediate container d0dca96ad53c
Step 5 : RUN echo "This is the content for index.html" > /var/www/html/index.html
  ---> Running in fe39aa35b577
  ---> 89f3cb1db1c3
Removing intermediate container fe39aa35b577
Step 6 : VOLUME /var/www/html
  ---> Using cache
  ---> 91d394fd412e
Step 7 : ENTRYPOINT /usr/bin/tail -f /dev/null
  ---> Running in 91b87b293b35
  ---> c6e914240bdf
Removing intermediate container 91b87b293b35
Successfully built 91d394fd412e
```

4. Create an instance of the data volume container, for example `dvc1`:

```bash
[root@host ~]# docker build --tag="mymod/dvc:v1" /var/docker_projects/mymod/dvc/
```

```
Uploading context  2.56 kB
Uploading context
Step 0 : FROM oraclelinux:6
  ---> 3e4b5e722ab9
Step 1 : MAINTAINER A N Other <another@example.com>
  ---> Using cache
  ---> debe47ce9b8
Step 2 : RUN mkdir -p /var/www/html
  ---> Running in fa94df7dd3af
  ---> 503132e87939
Removing intermediate container fa94df7dd3af
Step 3 : RUN echo "This is the content for file1.html" > /var/www/html/file1.html
  ---> Running in f98a14371672
  ---> e63ba0d36d88
Removing intermediate container f98a14371672
Step 4 : RUN echo "This is the content for file2.html" > /var/www/html/file2.html
  ---> Running in d0dca96ad53c
  ---> 27f2e2b3b207
Removing intermediate container d0dca96ad53c
Step 5 : RUN echo "This is the content for index.html" > /var/www/html/index.html
  ---> Running in fe39aa35b577
  ---> 89f3cb1db1c3
Removing intermediate container fe39aa35b577
Step 6 : VOLUME /var/www/html
  ---> Using cache
  ---> 91d394fd412e
Step 7 : ENTRYPOINT /usr/bin/tail -f /dev/null
  ---> Running in 91b87b293b35
  ---> c6e914240bdf
Removing intermediate container 91b87b293b35
Successfully built 91d394fd412e
```

To test that other containers can mount the data volume (`/var/www/html`) from `dvc1`, create a container named `websvr` that runs an HTTP server and mounts its data volume from `dvc1`.

```bash
[root@host ~]# docker run -d --volumes-from dvc1 --name websvr -P mymod/httpd:v2
```

```
008ce3de1cbf98ce50f6e3f3cf7618d248ce9dcfca8c29c1d9d4d79118d4c1b3
```

After finding out the correct port to use on the host, use `curl` to test that `websvr` correctly serves the content of all three files that were set up in the image.

```bash
[root@host ~]# docker port websvr 80
0.0.0.0:49154
[root@host ~]# curl http://localhost:49154
This is the content for index.html
[root@host ~]# curl http://localhost:49154/file1.html
This is the content for file1.html
[root@host ~]# curl http://localhost:49154/file2.html
This is the content for file2.html
```
4.9 Moving Data Between Docker Containers and the Host

You can use the `-v` option of `docker run` to copy volume data between a data volume container and the host. For example, you might want to back up the data so that you can restore it to the same data volume container or to copy it to a different data volume container.

The examples in this section assume that Docker is running two instances of the data volume container image `mymod/dvc:v1` that is described in Section 4.8, “Creating and Using Data Volume Containers”. You can use the following commands to start these containers:

```
# docker run -d --name dvc1 mymod/dvc:v1
# docker run -d --name dvc2 mymod/dvc:v1
```

To copy the data from a data volume to the host, mount the volume from another container and use the `cp` command to copy the data to the host, for example:

```
[root@host ~]# docker run --rm -v /var/tmp:/host:rw oraclelinux:6 \
    --volumes-from dvc1 cp -r /var/www/html /host/dvc1_files
```

The container mounts the host directory `/var/tmp` read-writable as `/host`, mounts all the volumes, including `/var/www/html`, that `dvc1` exports, and copies the file hierarchy under `/var/www/html` to `/host/dvc1_files`, which corresponds to `/var/tmp/dvc1_files` on the host.

To copy the backup of `dvc1`'s data from the host to another data volume container `dvc2`, use a command such as the following:

```
[root@host ~]# docker run --rm -v /var/tmp:/host:ro --volumes-from dvc2 \
    oraclelinux:6 cp -a -T /host/dvc1_files /var/www/html
```

The container mounts the host directory `/var/tmp` read-only as `/host`, mounts the volumes exported by `dvc2`, and copies the file hierarchy under `/host/dvc1_files` to `/var/www/html`, which corresponds to a volume that `dvc2` exports.

You could also use a command such as `tar` to back up and restore the data as a single archive file, for example:

```
[root@host ~]# docker run --rm -v /var/tmp:/host:rw --volumes-from dvc1 \
    oraclelinux:6 tar -cPvf /host/dvc1_files.tar /var/www/html \
/var/www/html/ \
/var/www/html/file1.html \
/var/www/html/file2.html \
/var/www/html/index.html
[root@host ~]# ls -l /var/tmp/dvc1_files.tar
-rw-r--r--. 1 root root 10240 Aug 31 14:37 /var/tmp/dvc1_files.tar
[root@host ~]# docker run --rm -l -t --name guest --v /var/tmp:/host:ro \
    --volumes-from dvc2 oraclelinux:6 /bin/bash
[root@guest ~]# rm /var/www/html/*.html
[root@guest ~]# ls -l /var/www/html/*.html
-rw-r--r--. 1 root root 35 Aug 30 09:02 file1.html
-rw-r--r--. 1 root root 35 Aug 30 09:03 file2.html
-rw-r--r--. 1 root root 35 Aug 30 09:03 index.html
[root@guest ~]# docker run --rm -i -t --name guest -v /var/tmp:/host:ro \
    --volumes-from dvc2 oraclelinux:6 /bin/bash
[root@host ~]# ls -l /var/www/html
total 12
-rw-r--r--. 1 root root 35 Aug 30 09:02 file1.html
-rw-r--r--. 1 root root 35 Aug 30 09:03 file2.html
-rw-r--r--. 1 root root 35 Aug 30 09:03 index.html
```
This example uses a transient, interactive container named `guest` to extract the contents of the archive to `dvc2`.

### 4.10 Using Labels to Define Metadata

From version 1.6.0 of Docker, you can use labels to add metadata to the Docker daemon and to Docker containers and images.

In the Dockerfile, a `LABEL` instruction defines an image label that can contain one or more key-value pairs, for example:

```
LABEL com.mydom.dept="ITGROUP" \
  com.mydom.version="1.0.0-ga" \
  com.mydom.is-final \
  com.mydom.released="June 6, 2015"
```

In this example, each key name is prefixed by the domain name in reverse DNS form (`com.mydom.`) to guard against name-space conflicts. Key values are always expressed as strings and are not interpreted by Docker. If you omit the value, you can use the presence or absence of the key in the metadata to encode information such as the release status. The backslash characters allow you to extend the label definition across several lines.

You can use the `docker inspect` command to display the labels that are associated with an image, for example:

```
$ docker inspect 7ac15076dcc1
...
"Labels": {
  "com.mydom.dept": "ITGROUP",
  "com.mydom.version": "1.0.0-ga",
  "com.mydom.is-final": ",",
  "com.mydom.release-date": "June 6, 2015"
}
...
```

You can use the `--filter "label=key[=value]"` option with the `docker images` and `docker ps` commands to list the images and running containers on which a metadata value has been set, for example:

```
$ docker images --filter "label=com.mydom.dept='DEVGROUP'"
$ docker ps --filter "label=com.mydom.is-beta2"
$ docker ps --filter "label=env=Oracle Linux 6"
```

For containers, you can use `--label key=value` options with the `docker create` and `docker run` commands to define key-value pairs, for example:

```
$ docker run -i -t --rm testapp:1.0 --label run="11" --label platform="Oracle Linux 6"
```

For the Docker engine, you can use `--label key=value` options if you start `docker` from the command line or edit the docker configuration file `/etc/sysconfig/docker`.

```
OPTIONS="--label com.mydom.dept='DEVGROUP'"
```

As containers and the Docker daemon are transitory and run in a known environment, it is not usually necessary to apply reverse domain name prefixes to key names.
4.11 Defining the Logging Driver

From version 1.6.0 of Docker, you can use the `--log-driver` option with the `docker create` and `docker run` commands to specify the logging driver that a container should use:

- **json-file**: Write log messages to a JSON file that you can examine by using the `docker logs` command, for example:
  
  ```
  docker logs --follow --timestamps=false container_name
  ```

  This is the default logging driver.

- **none**: Disable logging.

- **syslog**: Write log messages to `syslog`.

4.12 About Image Digests

From version 1.6.0 of Docker, registry version 2 or later images can be identified by their digest (for example, `sha256:digest_value_in_hexadecimal`). You can list the digest by specifying the `--digests` option to the `docker images` command. You can use a digest with the `docker create`, `docker pull`, `docker rmi`, and `docker run` commands and with the `FROM` instruction in a Dockerfile.

4.13 Specifying Control Groups for Containers

From version 1.6.0 of Docker onward, you can use the `--cgroup-parent` option with the `docker create` and `docker run` commands from version 1.7.0 of Docker onward. This setting allows the container to track the UTS namespace of the host or to set the host name and domain from the container.

**Warning**

As the container has full access to the UTS namespace of the host, this feature is inherently insecure.

4.14 Limiting CPU Usage by Containers

To control a container's CPU usage, you can use the `--cpu-period` and `--cpu-quota` options with the `docker create` and `docker run` commands from version 1.7.0 of Docker onward.

The `--cpu-quota` option specifies the number of microseconds that a container has access to CPU resources during a period specified by `--cpu-period`. As the default value of `--cpu-period` is 100000, setting the value of `--cpu-quota` to 25000 limits a container to 25% of the CPU resources. By default, a container can use all available CPU resources, which corresponds to a `--cpu-quota` value of -1.

4.15 Making a Container Use the Host's UTS Namespace

By default, a container runs with a UTS namespace (which defines the system name and domain) that is different from the UTS namespace of the host. To make a container use the same UTS namespace as the host, you can use the `--uts=host` option with the `docker create` and `docker run` commands from version 1.7.0 of Docker onward. This setting allows the container to track the UTS namespace of the host or to set the host name and domain from the container.

**Warning**

As the container has full access to the UTS namespace of the host, this feature is inherently insecure.

4.16 Setting ulimit Values on Containers

The `--ulimit` option to `docker run` allows you to specify `ulimit` values for a container, for example:
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Building Images with Resource Constraints

```
$ docker run -i -t --rm myapp:2.0 --ulimit nofile=128:256 --ulimit nproc=32:64
```

This example sets a soft limit of 128 open files and 32 child processes and a hard limit of 256 open files and 64 child processes on the container.

From version 1.6.0 of Docker, you can set default `ulimit` values for all containers by specifying `--default-ulimit` options in the docker configuration file `/etc/sysconfig/docker` by appending the options to the OPTIONS line as follows:

```
OPTIONS="--ulimit nofile=1280:2560 --ulimit nproc=256:512"
```

Any `ulimit` values that you specify for a container override the default values that you set for the daemon.

### 4.17 Building Images with Resource Constraints

From version 1.6.0 of Docker, you can specify cgroup resource constraints to `docker build`, for example:

```
# docker build --cpu-shares=100 --memory=1024m 
   --tag="mymod/myapp:1.0" /var/docker_projects/mymod/myapp/
```

Any containers that you generate from the image inherit these resource constraints.

You can use the `docker stats` command to display a container's resource usage, for example:

```
# docker stats cntr1 cntr2
CONTAINER     CPU %      MEM USAGE/LIMIT     MEM %     NET I/O
cntr1         0.05%      504 KiB/128 MiB     0.39%     2.033 KiB/40 B
cntr2         0.08%      1.756 MiB/128 MiB   1.37%     5.002 KiB/92 B
```

### 4.18 Committing, Exporting and Importing Images

You can use the `docker commit` command to save the current state of a container to an image.

```
# docker commit [--author="name"] 
   [--change="instructions"]... 
   [--message="text"] 
   [--pause=false] container [repository[:tag]]
```

You can use this image to create new containers, for example to debug the container independently of the existing container.

You can use the `docker export` command to export a container to another system as an image tar file.

```
# docker export [--output="filename"] container
```

**Note**

You need to export separately any data volumes that the container uses. See Section 4.9, "Moving Data Between Docker Containers and the Host".

To import the image tar file, use `docker import` and specify the image URL or read the file from the standard input.

```
# docker import [--change="instructions"]... URL [repository[:tag]]
# docker import [--change="instructions"]... - [repository[:tag]] < filename
```
From version 1.6.0 of Docker, you can use `--change` options with `docker commit` and `docker import` to specify Dockerfile instructions that modify the configuration of the image, for example:

```
# docker commit --change "LABEL com.mydom.status='Debug'" 7ac15076dccc1 mymod/debugimage:v1
```

For `docker commit`, you can specify the following instructions: `ADD`, `CMD`, `COPY`, `ENTRYPOINT`, `ENV`, `EXPOSE`, `FROM`, `LABEL`, `MAINTAINER`, `RUN`, `USER`, `VOLUME`, and `WORKDIR`.

For `docker import`, you can specify the following instructions: `CMD`, `ENTRYPOINT`, `ENV`, `EXPOSE`, `ONBUILD`, `USER`, `VOLUME`, and `WORKDIR`. 
Chapter 5 Docker Registry

Table of Contents

5.1 Using the Oracle Container Registry .......................................................... 35
5.2 Setting up a local Docker Registry Server ................................................. 36
5.3 Importing images into the local Docker Registry ....................................... 40

A Docker registry is a store of Docker images. A Docker image is a read-only template, which is used to create a Docker container. A Docker registry is used to store Docker images, which are used to deploy containers as required.

The default Docker registry is hosted at https://hub.docker.com. Oracle hosts its own Docker registry for Oracle software that requires users to accept Oracle Standard Terms and Restrictions prior to deployment. This registry is located at https://container-registry.oracle.com.

Enterprise environments may consider setting up a local Docker registry. This provides the opportunity to convert customized containers into images that can be committed into a local registry, to be used for future container deployment, reducing the amount of customized configuration that may need to be performed for mass deployments. A local registry can also cache and host images pulled from an upstream registry. This can reduce network overhead and latency when deploying matching containers across a spread of local systems.

5.1 Using the Oracle Container Registry

The Oracle Container Registry contains Docker images for licensed commercial Oracle software products that you may use in your enterprise. To access the Oracle Registry Server, you must have an Oracle Single Sign-On account. The Oracle Container Registry provides a web interface that allows an administrator to authenticate and then to select the Docker images for the software that your organization wishes to use. Oracle Standard Terms and Restrictions terms must be agreed to via the web interface. Once Oracle Standard Terms and Restrictions have been agreed, it is possible to pull images of the software from the Oracle Container Registry using the standard Docker pull command.

To pull an image from the Oracle Container Registry

1. In a web browser, navigate to https://container-registry.oracle.com and login via the Oracle Single Sign-On authentication service.

2. Use the web interface to accept the Oracle Standard Terms and Restrictions for the Oracle software images that you intend to deploy. Your acceptance of these terms are stored in a database that links the software images to your Oracle Single Sign-On login credentials. Your acceptance of the Oracle Standard Terms and Restrictions is valid only for 8 hours from the time you last accepted it. This is subject to change without notice. If you have not pulled the image within the valid period for acceptance, you need to repeat the process before you attempt to pull the image.

3. Use the web interface to browse or search for Oracle software images.

4. On the host system, use the docker login command to authenticate against the Oracle Container Registry using the same credentials that you used to log into the web interface:

```
# docker login container-registry.oracle.com
```

The command prompts you for your username and password.
5. On the host system, run:

```
# docker pull container-registry.oracle.com/area/image[:tag]
```

Substitute `area` with the repository location in the registry and `image` with the name of the software image as hosted on the Oracle Container Registry. You may optionally specify a particular `[:tag]` for the image. For example:

```
# docker pull container-registry.oracle.com/os/oraclelinux:7
# docker pull container-registry.oracle.com/java/serverjre
```

Note that the `area` and `image` are nearly always specified in lower case. The command to pull an image is usually provided on the Repo Info page, when you are viewing the images in the web interface of the Oracle Container Registry. Other useful information about the image and how it should be run may also be available on the same page.

6. If your credentials can be verified and the Oracle Standard Terms and Restrictions have been accepted, the image is pulled from the server and stored locally, ready to be used to deploy containers.

### 5.2 Setting up a local Docker Registry Server

Before you set up the Docker Registry server, note the following:

- The registry server is a Docker container application. The host must have an Internet connection to download the registry image either from the public Docker Hub or, if support is required, from the Oracle Container Registry.

- The registry server runs on port 5000 by default. If you run alternative services that use the same TCP port, such as the OpenStack Keystone service, you may need to change the configuration to avoid a port conflict. All systems that require access to your registry server must be able to communicate freely on this port, so adjust any firewall rules that may prevent this.

- The registry host requires a valid Secure Sockets Layer (SSL) certificate and private key, similar to using SSL for a web server.

If the host already has an SSL certificate, you can use that. However, if the SSL certificate was issued by an intermediate Certificate Authority (CA), you must combine the host's SSL certificate with the intermediate CA's certificate to create a certificate bundle so that Docker can verify the host's SSL certificate. For example:

```
# cat registry.example.com.crt intermediate-ca.pem > domain.crt
```

If the host does not already have an SSL certificate, the following instructions provide details for creating a self-signed certificate for testing purposes.

- The registry server requires at least 15GB of available disk space to store registry data. This is usually located at `/var/lib/registry`. It is good practice to create a separate btrfs formatted file system for this purpose to allow you to easily scale your registry and to leverage features within this file system, such as snapshotting. The following instructions provide details for setting up a btrfs file system using one or more available devices. The device could be a disk partition, an LVM volume, a loopback device, a multipath device, or a LUN.

**To set up a Docker registry server:**

1. Create a btrfs file system for the registry.
You create a btrfs file system with the utilities available in the btrfs-progs package, which should be installed by default.

Create a btrfs file system on one or more block devices:

```
# mkfs.btrfs [-L label] block_device ...
```

where `-L label` is an optional label that can be used to mount the file system.

For example:

- To create a file system in a partition `/dev/sdc1`:

  ```
  # mkfs.btrfs -L var-lib-registry /dev/sdc1
  ```

  The partition must already exist. Use a utility such as fdisk (MBR partitions) or gdisk (GPT partitions) to create one if needed.

- To create a file system across two disk devices, `/dev/sdd` and `/dev/sde`:

  ```
  # mkfs.btrfs -L var-lib-registry /dev/sd[de]
  ```

  The default configuration is to stripe the file system data (`raid0`) and to mirror the file system metadata (`raid1`) across the devices. Use the `-d` (data) and `-m` (metadata) options to specify the required RAID configuration. For `raid10`, you must specify an even number of devices and there must be at least four devices.

- To create a file system in a logical volume named `docker-registry` in the `ol` volume group:

  ```
  # mkfs.btrfs -L var-lib-registry /dev/ol/docker-registry
  ```

  The logical volume must already exist. Use Logical Volume Manager (LVM) to create one if needed.

  For more information on creating btrfs file systems, see the Oracle Linux Administrator’s Guide for Release 6 at:

  https://docs.oracle.com/cd/E37670_01/E41138/html/ol_create_btrfs.html

2. Mount the btrfs file system on `/var/lib/registry`.

   a. Obtain the UUID of the device containing the btrfs file system.

      Use the `blkid` command to display the UUID of the device and make a note of this value, for example:

      ```
      # blkid /dev/sdc1
      /dev/sdc1: LABEL="var-lib-registry" UUID="50041443-b7c7-4675-95a3-bf3a30b96c17" \
      UUID_SUB="09de3cbl-2f9b-4bd8-8881-87e591841c75" TYPE="btrfs"
      ```

      If the btrfs file system is created across multiple devices, you can specify any of the devices to obtain the UUID. Alternatively you can use the `btrfs filesystem show` command to see the UUID. For a logical volume, specify the path to the logical volume as the device for example `/dev/ol/docker-registry`. Ignore any UUID_SUB value displayed.

   b. Edit the `/etc/fstab` file and add an entry to ensure the file system is mounted when the system boots.
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Setting up a local Docker Registry Server

<table>
<thead>
<tr>
<th>UUID=UUID_value  /var/lib/registry  btrfs  defaults  0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replace UUID_value with the UUID that you found in the previous step. If you created a label for the btrfs file system, you can also use the label instead of the UUID, for example:</td>
</tr>
<tr>
<td>LABEL=label  /var/lib/registry  btrfs  defaults  0 0</td>
</tr>
</tbody>
</table>

3. Add the host's SSL certificate and private key to Docker.

   a. Create the /var/lib/registry/conf.d directory.
   
   ```bash
   # mkdir /var/lib/registry/conf.d
   ```

   b. Copy the host's SSL certificate and private key to the /var/lib/registry/conf.d directory.
   
   ```bash
   # cp certfile /var/lib/registry/conf.d/domain.crt
   # cp keyfile /var/lib/registry/conf.d/domain.key
   ```

   where certfile is the full path to the host's SSL certificate and keyfile is the full path to the host's private key. For example:

   ```bash
   # cp /etc/pki/tls/certs/registry.example.com.crt \
   /var/lib/registry/conf.d/domain.crt
   # cp /etc/pki/tls/private/registry.example.com.key \
   /var/lib/registry/conf.d/domain.key
   ```

   If the host does not have an SSL certificate and private key, you can create a self-signed certificate for testing purposes, as follows:

   ```bash
   # cd /var/lib/registry/conf.d
   # openssl req -newkey rsa:4096 -nodes -sha256 -x509 -days 365 \
   -keyout domain.key -out domain.crt
   ```

   Generating a 4096 bit RSA private key
   ..............................................................
   ..............................................................
   ........................++
   ..............................................................
   ........................++
   writing new private key to 'domain.key'
   ----- You are about to be asked to enter information that will be incorporated into your certificate request. What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank
Setting up a local Docker Registry Server

For some fields there will be a default value, if you enter '.', the field will be left blank.

-----
Country Name (2 letter code) [XX]: US
State or Province Name (full name) []: Massachusetts
Locality Name (eg, city) [Default City]: Boston
Organization Name (eg, company) [Default Company Ltd]: Example Com
Organizational Unit Name (eg, section) []: DevOps
Common Name (eg, your name or your server's hostname) []: registry.example.com
Email Address []: admin@example.com

The Common Name must be the same as the fully-qualified domain name (FQDN) of the host.

c. Change the file permissions on the private key:

```
# chmod 600 /var/lib/registry/conf.d/domain.key
```

4. If you are running a firewall, you must make sure that the TCP port that you intend the Docker registry to listen on is accessible.

If you are running `iptables`, you could add a rule as follows:

```
# iptables -I INPUT -p tcp -m tcp --dport 5000 -j ACCEPT
# service iptables save
```

5. Create the Docker registry container.

```
# docker run -d -p 5000:5000 --name registry --restart=always \
-v /var/lib/registry:/registry_data \
-e REGISTRY_STORAGE_FILESSYSTEM_ROOTDIRECTORY=/registry_data \
-e REGISTRY_HTTP_TLS_KEY=/registry_data/conf.d/domain.key \
-e REGISTRY_HTTP_TLS_CERTIFICATE=/registry_data/conf.d/domain.crt \
registry:latest
```

The registry image is pulled from the public Docker Hub and the Docker registry container is started. The `--restart=always` option ensures that the registry container is started whenever Docker is started. Note that you can map an alternate port number for your docker registry, if required, by changing the `5000` in the command above to match the port number that you would prefer to use.

6. If the registry host uses a self-signed SSL certificate, you must distribute the SSL certificate to all hosts in your deployment that you intend to use the local Docker registry.

Perform the following steps on each host, where `registry_hostname` is the name of the registry host, and `port` is the port number you selected for your Docker registry server, by default 5000:

a. Create the `/etc/docker/certs.d/registry_hostname:port` directory.
5.3 Importing images into the local Docker Registry

Once you have set up a Docker registry server, you can import Docker images into the registry so that they can be used to deploy containers. You may either pull images from an upstream registry, such as the Oracle Container Registry, and then commit them to your local registry, or you may wish to create your own images based on upstream images.

**To import upstream images into a local Docker registry:**

1. Pull an image from the upstream registry. For instance, you can pull an image from the Oracle Container Registry:

   ```
   # docker pull container-registry.oracle.com/os/oraclelinux:latest
   ```

2. Tag the image so that it points to the local registry. For example:

   ```
   # docker tag container-registry.oracle.com/os/oraclelinux:latest localhost:5000/ol7image:v1
   ```

   In this example, `localhost` is the hostname where the local registry is located and `5000` is the port number that the registry listens on. If you are working on a Docker engine located on a different host to the registry, you must change the hostname to point to the correct host. Note the repository and tag name, `ol7image:v1` in the example, must all be in lower case to be a valid tag.

3. Push the image to the local registry. For example:

   ```
   # docker push localhost:5000/ol7image:v1
   ```

See Section 4.3, “Creating a Docker Image from an Existing Container” and Section 4.4, “Creating a Docker Image from a Dockerfile” for information on how you can create your own images. Once you have committed a customized image, you can tag it and push it to your local registry as indicated in the steps above.
Chapter 6 Docker Swarm

Table of Contents

6.1 Swarm Management ................................................................. 41
  6.1.1 Initialize the swarm on a host selected as the management node ............... 42
  6.1.2 Add nodes to the swarm ...................................................... 42
  6.1.3 Verify that the nodes have been added to the swarm .......................... 43
  6.1.4 Promoting nodes to manager status ........................................ 43
  6.1.5 Removing nodes ............................................................... 43
  6.1.6 Recovering quorum in a swarm ............................................ 44

6.2 Service Management .............................................................. 44
  6.2.1 Creating a service and deploying containers .................................. 44
  6.2.2 Adding or removing container instances .................................... 45

As of Docker 1.12, the Docker Engine provides features to simplify and improve container orchestration for environments deploying multiple identical containers across multiple hosts through a built-in clustering facility. This provides mechanisms to improve load handling and availability. It also helps to reduce or avoid downtime as services are upgraded or replaced, and makes it possible to roll back changes with minimum effort. The engine provides all of the features required to set up a swarm, add and remove nodes to and from the swarm and deploy containers to the swarm. Round-robin load balancing is automatically implemented and failover is handled directly by the swarm.

Important

The Docker Swarm functionality is released as a technology preview for Oracle Linux. As a technology preview, this feature is still under development but is made available for testing and evaluation purposes.

Full documentation for swarm mode within the Docker Engine is provided at https://docs.docker.com/engine/swarm/.

In this chapter, we provide a brief introduction to show how you can start using Docker swarm mode to take advantage of these features immediately. A swarm consists of multiple hosts, each running Docker Engine 1.12 or later. All hosts must be accessible to each other across the same network.

6.1 Swarm Management

A swarm consists of one or more manager nodes and several worker nodes. The manager node is used to dispatch tasks to worker nodes. The manager also performs all of the orchestration and cluster management functions to maintain the state of the swarm. A single manager node is elected as the leader manager node and other manager nodes remain on standby so that they are ready to take on the role of leader at any point in time. Manager nodes are elected to the leader role through node consensus. Although it possible to run an environment with a single manager node, ideally three or more nodes should run as manager nodes.

Worker nodes receive tasks from manager nodes and execute required actions for the swarm, such as starting or stopping a container. By default, manager nodes also behave as worker nodes, but this behavior is configurable. The leader manager node tracks the state of the cluster and in the event that a worker node becomes unavailable, the manager ensures that any containers that were running on the unavailable worker node are started on an alternative worker node.
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Initialize the swarm on a host selected as the management node

Tasks that affect the swarm can be executed via any manager node regardless of whether it is the leader. At any point, a manager node can be used to promote a worker node to management status. See Section 6.1.4, “Promoting nodes to manager status” for more information.

In this section, the steps to set up the nodes in a swarm are discussed in some detail. Instructions are also provided for some basic swarm management.

6.1.1 Initialize the swarm on a host selected as the management node

Creating a new swarm is a simple operation. On a host running Docker Engine 1.12 or later, run the following command:

```bash
# docker swarm init --advertise-addr IP Address:PORT
```

Substitute `IP Address` with the IP address or device name of the network interface that manager nodes should use to communicate between each other and to facilitate overlay networking. If the system has a single network interface or IP address, you do not need to use the `--advertise-addr` option. You can optionally specify the TCP port that should be used for API communication across the swarm, by default this is achieved on port 2377.

Two secret keys, or join tokens, are created when the swarm is initialized. These keys are used to authenticate nodes that are attempting to join the swarm and, depending on the key used, can control whether a node joins the swarm as a worker node or as a manager node. You are able to see or reset the key values at any point by running the `swarm join-token` command on any manager node. Example usage of this command follows below:

```bash
# docker swarm join-token worker
To add a worker to this swarm, run the following command:
  docker swarm join
    --token SWMTKN-1-0np2hom3z5q03qboqft2nm4n8flyv4n1grnubow2rzejwdd7j2-93gbsv348qg1pnp5zwyuhynj3192.168.1.104:2377

# docker swarm join-token manager
To add a manager to this swarm, run the following command:
  docker swarm join
    --token SWMTKN-1-0np2hom3z5q03qboqft2nm4n8flyv4n1grnubow2rzejwdd7j2-3szdu6g3q5szgvaritwas5of8192.168.1.104:2377
```

The command returns the docker command that should be run on the system that you wish to add to the swarm as either a manager or worker node.

Use the `--rotate` option to rotate or refresh the keys. If a key is rotated, the old key value can no longer be used to join a new node to the swarm. The key tokens are important to protecting the integrity of the swarm. Manager keys are particularly important to the security and stability of the swarm. If you think that a key has been stored or published in plain text somewhere, it is good practice to rotate the key.

6.1.2 Add nodes to the swarm

To add nodes to the swarm, on each host run the following command:

```bash
# docker swarm join --token TOKEN HOSTNAME:PORT
```

Substitute `TOKEN` with the appropriate key value to authenticate the node to the swarm as either a manager or worker node. To obtain the correct key value, you can use the `swarm join-token manager` or `swarm join-token worker` command on a management node. This command will actually provide you with the full syntax of the command required to join the system to the swarm as a node.
6.1.3 Verify that the nodes have been added to the swarm

From the manager node, you can run docker commands that control and report on the status of member nodes within a swarm. To check which nodes belong to the swarm, you can run the following command on the manager node where you initialized the swarm:

```
docker node ls
```

The following output shows an example of the listing, where it is clear that there is a single manager node that has been promoted to manager leader:

```
ID                  HOSTNAME         MEMBERSHIP  STATUS  AVAILABILITY  MANAGER STATUS
0epm3gi2jtc5p4pxfqqgsas1g6  o12.example.com  Accepted    Ready   Active
0wgtalpqjomyiem8r33730fiy  o13.example.com  Accepted    Ready   Active
ebukdrl4ge4ti2irc74qy944e  o11.example.com  Accepted    Ready   Active        Leader
eygqkcwm0xw547yg8eocfdh3l  o14.example.com  Accepted    Ready   Active
```

6.1.4 Promoting nodes to manager status

You can promote any worker nodes to manager status at any time by running the following command on an existing manager node:

```
docker node promote ID HOSTNAME
```

Substitute `ID HOSTNAME` with the ID or hostname of the worker node that you wish to promote. For example, a worker node is promoted based on its docker ID:

```
docker node promote 0eba8xkk048msqtz3talpi4h
```

You can also demote manager nodes to return them back to worker-only status. In the following example, a manager node is demoted based on its hostname:

```
docker node demote ol3.example.com
```

This command must be run on an alternate manager node, and not on the node that is being demoted. If only two nodes remain in manager status, you are unable to demote either node as quorum would be lost and the cluster would fail. Once you only have two nodes running in manager status, if the current manager leader goes offline, quorum is lost and the remaining manager is unable to promote itself to leader, resulting in a cluster failure. Therefore, a swarm should either only have a single manager or should have a minimum of three managers to maintain quorum for failover.

6.1.5 Removing nodes

To remove a node from a swarm, run the following command on the node that must be removed:

```
docker swarm leave
```

Note that to leave the swarm, the node should not be running in a manager role. Demote the node from manager status beforehand.

Once the node leaves the swarm, any containers running as part of a service within the swarm that were running on the node are started on one or more alternate nodes in the swarm automatically. If resources may be an issue, consider scaling down the service before removing the node.
In the case where a node is no longer online and must be removed from the swarm, the following command can be run on a manager node:

```
# docker node rm ID|HOSTNAME
```

Substitute `ID|HOSTNAME` with the ID or hostname of the node to be removed from the swarm. Note that if the node is brought online at a later stage, it is unaware of this operation but is not automatically accepted back into the swarm. In this case, run the `docker swarm leave` command on the node when it comes online again. Use the `docker swarm join` command to rejoin the swarm.

### 6.1.6 Recovering quorum in a swarm

The clustering facility within the swarm is maintained by the manager nodes. In the case where a manager leader becomes unavailable and there are not enough remaining managers to reach quorum and elect a new manager leader, quorum is lost and no manager node is able to control the swarm. In this case, it may be necessary to re-initialize the swarm and force the creation of a new cluster. This is best achieved by running the following command on the manager leader when it is brought online again:

```
# docker swarm init --force-new-cluster
```

Usually, worker nodes continue to function normally. Other manager nodes, should resume functionality once the swarm has been re-initialized, however it might be necessary to remove manager nodes from the swarm and rejoin them to the swarm. Note that when a node rejoins the swarm, it must join the swarm via a manager node.

### 6.2 Service Management

Containers are deployed as a service that is created within the swarm. The services running on the swarm are accessible via any of nodes that are running at least one container for the service in the swarm. If multiple replica containers for the same service are started on the swarm, requests for the service are automatically load-balanced among container instances running on the same node. If a node running a service becomes unavailable, the manager node starts an equal number of container instances of the service on the remaining nodes within the swarm.

#### 6.2.1 Creating a service and deploying containers

Containers can be deployed to the swarm in much the same way as containers are run on a single host. A service is created and the image that should be used to deploy the container is specified. Additionally, port mappings and the number of replica containers to deploy across the swarm can be specified as required. The following example shows how to create a service consisting of three replica containers deployed within the swarm. The service is given a name that makes it easy to identify. Port mappings are defined. In this example, the service uses the `nginxdemos/hello` image:

```
# docker service create --replicas 3 --name hello -p 80:80 nginxdemos/hello
```

If the image is not already available to each node, the image is automatically pulled from the Docker Hub. Therefore, for this example to work, all nodes must either have direct internet access or should be configured to use a proxy. In a production environment, images would usually be hosted in a local registry and each node could pull the image from the local registry.

You can check which services are running on a swarm at any time:

```
# docker service ls
```

Typical output for this command is as follows:
The software described in this documentation is either no longer supported or is in extended support. Oracle recommends that you upgrade to a current supported release.

Adding or removing container instances

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>REPLIACS</th>
<th>IMAGE</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>64w8p8amo49</td>
<td>hello</td>
<td>3/3</td>
<td>nginxdemos/hello</td>
<td></td>
</tr>
</tbody>
</table>

From the output, it is possible to see the ID assigned to the service, the name of the service, how many replica containers are running, the image that was used to spawn the containers and any commands issued to run in the container when it was started. For more detail on the status of each container instance running within the service, you list the tasks running in the service as follows:

```
# docker service ps ID|SERVICE
```

For example:

```
# docker service ps hello
```

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>SERVICE</th>
<th>IMAGE</th>
<th>LAST STATE</th>
<th>DESIRED STATE</th>
<th>NODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3lb26aj...vsl540</td>
<td>hello.1</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 13 seconds</td>
<td>Running</td>
<td>ol2.example.com</td>
</tr>
<tr>
<td>00hmy0p...b4kgsp</td>
<td>hello.3</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running About an hour</td>
<td>Running</td>
<td>ol4.example.com</td>
</tr>
<tr>
<td>cf8b8ctb...59nvwp</td>
<td>hello.4</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running About an hour</td>
<td>Running</td>
<td>ol3.example.com</td>
</tr>
</tbody>
</table>

### 6.2.2 Adding or removing container instances

It is possible to add replica container instances to a service, or to remove them, at any point to scale the service to meet demand. This is achieved using the `docker service scale` command on any manager node:

```
# docker service scale SERVICE=REPLICAS
```

Substitute `SERVICE` with the name or ID of the service to be scaled and substitute `REPLICAS` with the number of replica container instances that you wish to run in the service. For example:

```
# docker service scale hello=12
hello scaled to 12
# docker service ps hello
```

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>SERVICE</th>
<th>IMAGE</th>
<th>LAST STATE</th>
<th>DESIRED STATE</th>
<th>NODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5u710ts...weir0v</td>
<td>hello.1</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 17 minutes</td>
<td>Running</td>
<td>ol1.example.com</td>
</tr>
<tr>
<td>b1nqbkp...65gj3f</td>
<td>hello.2</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 17 minutes</td>
<td>Running</td>
<td>ol3.example.com</td>
</tr>
<tr>
<td>00hmy0p...b4kgsp</td>
<td>hello.3</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running About an hour</td>
<td>Running</td>
<td>ol4.example.com</td>
</tr>
<tr>
<td>8pv15jy...u75oz2</td>
<td>hello.4</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 3 seconds</td>
<td>Running</td>
<td>ol2.example.com</td>
</tr>
<tr>
<td>cqqyffbs...3w0m54</td>
<td>hello.5</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 4 seconds</td>
<td>Running</td>
<td>ol1.example.com</td>
</tr>
<tr>
<td>4pggall...b0yek3</td>
<td>hello.6</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 4 seconds</td>
<td>Running</td>
<td>ol3.example.com</td>
</tr>
<tr>
<td>9kds5sq...ybanef</td>
<td>hello.7</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 4 seconds</td>
<td>Running</td>
<td>ol4.example.com</td>
</tr>
<tr>
<td>44rebt5...dyv6e</td>
<td>hello.8</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 4 seconds</td>
<td>Running</td>
<td>ol2.example.com</td>
</tr>
<tr>
<td>8hoiml0...42ouu</td>
<td>hello.9</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 4 seconds</td>
<td>Running</td>
<td>ol4.example.com</td>
</tr>
<tr>
<td>43aa99b...jyi194</td>
<td>hello.10</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 3 seconds</td>
<td>Running</td>
<td>ol4.example.com</td>
</tr>
<tr>
<td>a7kw7y...60gnx5</td>
<td>hello.11</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 3 seconds</td>
<td>Running</td>
<td>ol1.example.com</td>
</tr>
<tr>
<td>cgg9snb...aersh4</td>
<td>hello.12</td>
<td>hello</td>
<td>nginxdemos/hello</td>
<td>Running 2 seconds</td>
<td>Running</td>
<td>ol2.example.com</td>
</tr>
</tbody>
</table>

In the example, the `hello` service is scaled to run 12 replica containers. When the tasks running in the `hello` service are listed, it is apparent that many new containers have been spawned as these have only been running for several seconds, while three original containers have already been running for significantly longer.

In these examples the `nginxdemos/hello` image is useful as it can be used to demonstrate the load-balancing functionality when the service is accessed on any particular node. The `nginxdemos/hello` image contains a web application that returns the hostname and IP address for the container where it is running. By simply pointing a web browser at any of the nodes in the swarm and repeatedly refreshing, it should become apparent that the hostname returned by the application changes using a round-robin algorithm. This can also be illustrated on the command line using `curl`, for example:

```
# curl --silent "http://ol1.example.com" | grep 'hostname'
```
Adding or removing container instances

To remove container instances from the service, scale down to a number lower than the number of instances currently running. To reduce the number of instances in the example, the `hello` service is scaled down to two instances:

```
# docker service scale hello=2
hello scaled to 2
```
Chapter 7 For More Information About Docker

For more information about Docker, see https://www.docker.com/ and the Docker manual pages.