Oracle Linux

Oracle Container Runtime for Docker User's Guide

E87205-30
June 2023

Oracle Linux Oracle Container Runtime for Docker User's Guide,

E87205-30

Contents

3 Upgrading Oracle Container Runtime for Docker

4 Managing the Docker Engine Service

5 Working With Containers and Images

Preface

Oracle Linux: Oracle Container Runtime for Docker User's Guide describes how to use Oracle Container Runtime for 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 7.

Note that Oracle recommends that you consider using Podman, Buildah and Skopeo on Oracle Linux 8 for your container requirements.

Conventions

The following text conventions are used in this document:

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

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at https://www.oracle.com/corporate/accessibility/templates/t2-11535.html.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

About Oracle Container Runtime for Docker

Oracle Container Runtime for Docker allows you to create and distribute applications across Oracle Linux systems and other operating systems that support Docker. Oracle Container Runtime for Docker consists of the Docker Engine, which packages and runs the applications, and integrates with the Docker Hub and Oracle Container Registry to share 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.

> **Important:**
>
> Oracle Container Runtime for Docker releases 17.03 and later are only available on Oracle Linux 7 (x86_64). Oracle Linux 6 is not supported for Oracle Container Runtime for Docker version 17.03 and later.

The Docker Hub hosts applications as *Docker images* and provides services that allow you to create and manage a Docker environment. If you register for an account with the Docker Hub, you are able to use it to store your own private images. You do not need an account at Docker to access publicly accessible images on the Docker Hub. The Docker Hub also hosts enterprise-ready applications that are certified as trusted and supported. These applications are made available by the verified publishers. Some applications shipped on the Docker Hub may require payment.

> **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.

The Oracle Container Registry contains images for licensed commercial, and open source, Oracle software products. Images may also be used for development and testing purposes. The commercial license covers both production and non-production use. The Oracle Container Registry provides a web interface where customers are able to select Oracle images, and, if required, agree to terms of use, before pulling the images using the standard Docker client software. More information on this service is provided in Pulling Images From the Oracle Container Registry.

Technical Preview Releases

Oracle makes interim releases of Oracle Container Runtime for Docker available as technical previews. These releases are **not** supported by Oracle and are not intended for production use.

Preview releases can be obtained by subscribing to the `ol7_preview` repository on the Oracle Linux yum server. You can install the appropriate package to obtain the correct repository configuration before enabling the repository:

```
sudo yum install oraclelinux-developer-release-el7
sudo yum-config-manager --enable ol7_preview
```

The installation and upgrade procedures described in this guide should continue to apply for each preview release.

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://docs.docker.com/release-notes/. 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.

Oracle Container Runtime for Docker 19.03

The current release of Oracle Container Runtime for Docker is based on the upstream Docker 19.03 release and incorporates the changes present in subsequent upstream releases since the previous release. The notable changes in this release are:

- The `docker run` and `docker create` commands now include an option to set the domain name, using the `--domainname` option.
- The `docker image pull` command now includes an option to quietly pull an image, using the `--quiet` option.
- Faster context switching using the `docker context` command.
- Added ability to list kernel capabilities with `--capabilities` instead of `--capadd` and `--capdrop`.
- Added ability to define sysctl options with `--sysctl list`, `--sysctl-add list`, and `--sysctl-rm list`.
- Added inline cache support to builder with the `--cache-from` option.
- The IPVLAN driver is now supported and no longer considered experimental.
- Deprecated image manifest v2 schema 1 in favor of v2 schema 2.
- Removed v1.10 migrator.
- CVE-2020-13401 is resolved in the 19.03.11 errata release package.

Oracle Container Runtime for Docker 18.09

This release of Oracle Container Runtime for Docker was based on the upstream Docker 18.09 release and incorporated the changes present in subsequent upstream releases since the 18.03 release.

Notably, multi-registry support is no longer in technical preview and is enabled as a feature within this release. Additionally, Oracle introduces the `--default-registry` option, which can be used to change the default registry to point to an alternate registry to the standard Docker Hub registry. See Setting Container Registry Options for more information.

This release of Docker introduces an integrated SSH connection helper that allows any Docker client to connect to a remote Docker engine daemon securely over SSH. You can connect to a remote daemon using the `-H ssh://user@host` syntax. For example:

```
docker -H ssh://docker_user@host1.example.com run -it --rm busybox
```

To configure a client to use the same remote daemon always, you can set the `DOCKER_HOST` environment variable to contain the appropriate SSH URI. The SSH connection helper respects SSH options set for a host within the user's local SSH configuration file.

The Docker client application can now be installed as an independent package, `docker-cli`, so that the Docker engine daemon does not need to be installed on a system that may be used to manage a remote Docker daemon instance. The client package is automatically installed as a dependency when you install the Docker engine daemon package.

Docker 18.09 also introduces BuildKit, an overhaul of the build architecture used to build Docker images. The BuildKit mode is backward compatible with legacy build architecture, so that the Dockerfile format used to build previous images can continue to be used. BuildKit can be enabled on a system by setting the `DOCKER_BUILDKIT` environment variable to the value of `1`. BuildKit build output is enhanced to include progress and build times and many build processes can be run in parallel to greatly enhance performance and build time. The new Docker build architecture also includes improvements to security, including options to pass secret information to builds in a more secure manner. See the upstream documentation at https://docs.docker.com/develop/develop-images/build_enhancements/ for more information. This feature is available as a technical preview in this release of Oracle Container Runtime for Docker.

Docker 18.09 uses a new version of containerd, version 1.2.0. This version of the containerd package includes many enhancements for greater compatibility with the most recent Kubernetes release.

Oracle Container Runtime for Docker 18.03

This release of Oracle Container Runtime for Docker was based on the upstream Docker 18.03 release and incorporated the changes present in subsequent upstream releases since the 17.06 release.

Most notably, Oracle has implemented multi-registry support that makes it possible to run the daemon with the `--add-registry` flag, to include a list of additional registries to query when performing a pull operation. This functionality, enables Oracle Container Runtime for Docker to use the Oracle Container Registry as the default registry to search for container images, before falling back to alternate registry sources such as a local mirror, the Docker Hub. Other functionality available in this feature includes the `--block-registry` flag which can be used to prevent access to a particular Docker registry. Registry lists ensure that all images are

prefixed with their source registry automatically, so that a listing of Docker images always indicates the source registry from which an image was pulled. See Setting Container Registry Options for more information.

> **Important:**
>
> Docker registry list functionality is available as a technology preview and is not supported. As a technology preview, this feature is still under development but is made available for testing and evaluation purposes.

The `--insecure-registry` option is also included in this release and allows use of a registry over HTTPS without certificate-based authentication. This can be useful when working in development or testing environments, but should not be used in production.

Docker 18.03 introduces enhancements that allow for better integration with Kubernetes orchestration as an alternative to Docker Swarm, including changes to follow namespace conventions used across a variety of other containerization projects.

The `--chown` option is now supported for the ADD and COPY commands in a Dockerfile, giving users more control over file ownership when building images.

The Dockerfile can also now exist outside of the build-context, allowing you to store Dockerfiles together and to reference their paths in the `docker build` command on stdin.

Several improvements to logging and access to docker logs have been added, including the `--until` flag to limit the log lines to those that occurred before the specified timestamp.

Experimental Docker trust management commands have been added to better handle trust management on Docker images. See the `docker trust` command for more information.

Docker Swarm changes and improvements have gone into this release. Customers are reminded that Docker Swarm remains in technical preview in this release.

The deprecated `--enable-api-cors` daemon flag, which allowed cross-origin resource sharing to expose the API, has been removed in favor of the `--api-cors-header` option, which takes a string value to set the Access Control Allow Origin headers for the API and to determine access control for cross-origin resource sharing.

The deprecated `docker daemon` command, which was kept for backward compatibility, has been removed in this release.

Oracle Container Runtime for Docker 17.06

This release disables communication with legacy registries, running the v1 protocol, by default. While it is possible to allow communication using this version of the protocol by setting the `--disable-legacy-registry=false` daemon option, you should be aware that support for this is deprecated.

The `--graph` daemon option is also deprecated in favor of the `--data-root` option, as this is more descriptive and less confusing. The option indicates the path of the parent

directory that contains data for images, volumes, containers, networks, swarm cluster state and swarm node certificates.

One of the most significant changes in this release is the addition of support for multi-stage builds. This allows users to create Dockerfiles that pull intermediate build images that may be used to compile the final image, but which do not need to be included in the final image, itself. This can help to reduce image sizes and improve load times and performance of running containers. More information on multi-stage builds can be found in Creating Multi-Stage Docker Image Builds.

Other changes to the build environment include the ability to use build-time arguments in the form of `ARG` instructions in a Dockerfile, which allows you to pass environment variables into each image. `FROM` instructions support variables defined in `ARG` instructions that precede them in the Dockerfile.

Changes and improvements for Docker logging and networking are largely focused on improving Docker Swarm functionality. Numerous Docker Swarm changes and improvements have gone into this release. Customers are reminded that Docker Swarm remains in technical preview in this release.

In this release, the `overlay2` storage driver is supported in conjunction with SELinux. In previous releases, the Docker Engine did not start when SELinux was enabled and an overlay file system was in use. This check has been dropped as newer kernels have support for this combination and the packages for SELinux support have been updated.

Also included in this release is the `docker-storage-config` utility, that can be used to help new users correctly set up Docker storage for a new installation, so that the configuration follows Oracle guidelines. See Configuring Docker Storage Automatically for more information.

Docker 17.03

Changes to the upstream Docker release cycle bring about a new versioning scheme that uses date variables (YY.MM) in the version name to indicate when a version was released upstream.

The 17.03 release includes bug fixes for the 1.13 release and does not include any major feature changes. There are several improvements to the Docker Swarm functionality.

SELinux must be set to permissive mode or disabled when running the Docker Engine while using the `overlay2` storage driver.

Note that on XFS-formatted file systems, where dtype support is disabled, the default storage driver in this release is overridden from `overlay2` and is set to `devicemapper` for compatibility reasons. Storage driver override is only implemented on fresh installations of Docker and only where the underlying file system is detected as XFS without dtype support. See Configuring Docker Storage for more information.

The upstream default storage driver for Docker was changed from `devicemapper` to `overlay2`. This change can cause problems on systems where overlay is used in conjunction with a file system that does not have dtype support enabled. Since the root partition on Oracle Linux 7 is automatically formatted with `-n ftype=0` (disabling dtype support), where XFS is selected as the file system, the package installer checks the filesystem for dtype support and if this is not enabled the default storage driver is set to use `devicemapper`. This ensures that Docker is ready-to-use on newly installed systems and is achieved by setting the storage driver in the storage options in `/etc/sysconfig/docker-storage`.

It is possible to reconfigure Docker to use an alternate storage driver, by using the `--storage-driver` flag when running the Docker Engine daemon, or by setting the `storage-driver` option in the `daemon.json` configuration file. Oracle recommends that you use dedicated storage, formatted using Btrfs, for Docker. If you intend to use the `overlay2` storage driver with an XFS-formatted file system, you must ensure that dtype support is enabled. See Configuring Docker Storage for more information. Remember that if you wish to change the storage driver from `devicemapper`, you must remove the option set in `/etc/sysconfig/docker-storage`.

Other improvements were made to the Docker remote API and to the Docker client to add consistency to the command set. Also runtime improvements were made to the Docker Engine. Further developments on Docker Swarm mode are also noted.

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 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.

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.

2

Installing Oracle Container Runtime for Docker

This chapter describes the steps required to perform an installation of Oracle Container Runtime for Docker on an Oracle Linux 7 host.

Before you install and configure the Docker Engine on an Oracle Linux 7 system, make sure you are running an appropriate release of the Unbreakable Enterprise Kernel. Instructions to install UEK are detailed in Setting Up the Unbreakable Enterprise Kernel.

If you are already running either UEK R4 or UEK R5, you can follow the instructions in Installing Oracle Container Runtime for Docker to complete your installation.

Setting Up the Unbreakable Enterprise Kernel

Configure the system to use the Unbreakable Enterprise Kernel Release 5 (UEK R5) or later and boot the system with this kernel. If you are using an earlier Unbreakable Enterprise Kernel (UEK) release, or the Red Hat Compatible Kernel (RHCK), you must upgrade the kernel.

To install or update UEK:

1. If your system is registered with ULN, disable access to the `ol7_x86_64_UEKR3` and `ol7_x86_64_UEKR4` channels, and enable access to the `ol7_x86_64_UEKR5` channel.

   Log into https://linux.oracle.com with your ULN user name and password and click on the Systems tab to select the system where you installing Oracle Container Runtime for Docker. Go to the **Manage Subscriptions** page and update the channel subscriptions for the system. Click on **Save Subscriptions** to save your changes.

2. If you use the Oracle Linux yum server, disable the `ol7_UEKR3` and `ol7_UEKR4` repositories and enable the `ol7_UEKR5` repository. You can do this easily using `yum-config-manager`:

   ```
   sudo yum-config-manager --disable ol7_UEKR3 ol7_UEKR4
   sudo yum-config-manager --enable ol7_UEKR5
   ```

3. Run the following command to upgrade the system to the selected UEK release:

   ```
   sudo yum update
   ```

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

   ```
   sudo systemctl reboot
   ```

Enabling Access to the Oracle Container Runtime for Docker Packages

To access to the Oracle Container Runtime for Docker packages, you must enable the appropriate ULN channel or yum repositories.

If your system is registered with ULN, enable the `ol7_x86_64_addons` channel. Use the ULN web interface to subscribe the system to the appropriate channel:

1. Log in to https://linux.oracle.com with your ULN user name and password.
2. On the Systems tab, click the link named for the system in the list of registered machines.
3. On the System Details page, click **Manage Subscriptions**.
4. On the System Summary page, select each required channel from the list of available channels and click the right arrow to move the channel to the list of subscribed channels.

   Subscribe the system to the `ol7_x86_64_addons` channel.
5. Click **Save Subscriptions**.

If you use the Oracle Linux yum server, enable the `ol7_addons` channel. To enable a yum repository on your system, use the `yum-config-manager` command. For example, run:

```
sudo yum-config-manager --enable ol7_addons
```

Removing the `docker` Package

The latest Docker package is `docker-engine`, which conflicts with the older `docker` package. If you have the older `docker` package installed, you must remove it before you install Docker Engine. To check if you have the older `docker` package installed, run:

```
sudo rpm -qi docker
```

If the older `docker` package is installed, stop the `docker` service and remove the package. To stop the `docker` service:

```
sudo systemctl stop docker
```

Remove the `docker` package.

```
sudo yum remove docker
```

You can now install the `docker-engine` package.

Installing Oracle Container Runtime for Docker

To install the `docker-engine` and `docker-cli` packages.

```
sudo yum install docker-engine docker-cli
```

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

```
sudo systemctl enable --now docker
```

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

```
sudo systemctl status docker
```

You can also use the `docker info` command to display information about the configuration and version of the Docker Engine.

```
sudo docker info
```

For more information, see the `docker(1)` manual page.

Configuring a Proxy Server

To configure web proxy networking options, create the drop-in file `/etc/systemd/system/docker.service.d/http-proxy.conf` that contains the following lines:

```
[Service]
Environment="HTTP_PROXY=proxy_URL:port"
Environment="HTTPS_PROXY=proxy_URL:port"
```

Replace *proxy_URL* and *port* with the appropriate URLs and port numbers for your web proxy.

After adding or modifying a `systemd` drop-in file while the `docker` service is running, you need to tell `systemd` to reload the configuration for the service.

```
sudo systemctl daemon-reload
```

Restart the `docker` service for the configuration changes to take effect.

```
sudo systemctl restart docker
```

Configuring IPv6 Networking

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

For more information about configuring Docker networking, see:

https://docs.docker.com/engine/userguide/networking/

To configure IPv6 networking:

1. Create or edit `/etc/docker/daemon.json`.

   If you are creating this file from scratch, it should look like this:

   ```
   {
     "ipv6": true
   }
   ```

   If this file already exists and contains other entries, be careful that adding a line for the `ipv6` configuration variable conforms with typical JSON formatting.

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

   ```
   {
     "ipv6": true,
     "fixed-cidr-v6": "2001:db8:1::/64"
   }
   ```

   Similarly, you can also configure the default IPv6 gateway that should be used by Docker, using the `default-gateway-v6` parameter in this configuration file.

   For more information on the format and options for this configuration file, see:

https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file

2. Check that the `--ipv6`, `--fixed-cidr-v6` and `default-gateway-v6` options are not being invoked as command line switches when starting the Docker engine daemon.

   You should check that these options do not appear in either the `/etc/sysconfig/docker` or `/etc/sysconfig/docker-networking` files. These files are deprecated and may be removed in future releases. If these files contain any other configuration parameters, consider whether you could move these into `/etc/docker/daemon.json` to future-proof your configuration.

   Also check that these options do not appear in any systemd drop-in files in `/etc/systemd/system/docker.service.d/`. While this is a supported configuration option, it is preferable to keep all Docker Engine configuration in the same place, where possible.

Configuring Docker Storage

The Docker Engine is configured to use `overlay2` as the default storage driver to manage Docker containers. This provides a performance and scalability improvement on earlier releases that used the device mapper as the default storage driver, but the technology is new and should be tested properly before use in production environments. For more information on `overlay2`, see:

https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/

Overlay file systems can corrupt when used in conjunction with any file system that does not have dtype support enabled.

> **Note:**
>
> For Oracle Linux 7.4 or earlier, the root partition is automatically formatted with `-n ftype=0` (**disabling** dtype support), where XFS is selected as the file system.

The Docker Engine installer checks the filesystem for dtype support, and if this is not enabled, the default storage driver is set to use `devicemapper`. This check is only performed on a fresh installation of Docker Engine. The configuration of an existing Docker installation is unaffected during upgrade.

This allows Docker to function on a default Oracle Linux 7 system without any additional configuration required, immediately after install. However, using the `devicemapper` is not recommended for production environments. Performance and scalability can be compromised by this configuration. Therefore, it is important to consider using dedicated storage for Docker, and to change the storage driver to use either `btrfs` or `overlay2`.

Important:

If you continue to use `devicemapper` as the storage driver, you should be aware that some Docker images, such as the image for Oracle Database, require that the base device size is set to 25GB or more. The default base device size for `devicemapper` is updated to 25GB, but this only meets a minimum requirement for some containers. Where additional capacity may be required, the base device size can be changed by setting the `dm.basesize` start option for a container or, globally, for the Docker Engine.

You can change this value globally, by adding it to the `storage-opts` configuration parameter in `/etc/docker/daemon.json`, for example:

```
{
...
"storage-opts" : [ "dm.basesize=50G" ],
...
}
```

The base device size is sparsely allocated, so an image may not initially use all of this space. You can check how much space is allocated to the `Base Device Size` by running the `docker info` command.

For more information on storage driver options, see:

https://docs.docker.com/engine/reference/commandline/dockerd/#storage-driver-options

Oracle recommends using Btrfs as a more stable and mature technology than overlayfs.

In most cases, it is advisable to create a dedicated file system to manage Docker containers. This file system can be mounted at `/var/lib/docker` at boot time, before the Docker service is started.

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. For information on converting file systems, see the Oracle® Linux 7: Administrator's Guide.

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

Important:

XFS file systems must be created with the `-n ftype=1` option enabled for use as an overlay. The root partition on Oracle Linux 7 is automatically formatted with `-n ftype=0` where XFS is selected as the file system. Therefore, if you intend to use the `overlay2` storage driver in this environment, you must format a separate device for this purpose.

Configuring Docker Storage Automatically

The `docker-engine` package includes a utility that can help you to configure storage correctly for a new Docker deployment. The `docker-storage-config` utility can format a new block device, set up the mount point and correctly configure the Docker Engine to run with the appropriate storage driver so that your storage configuration follows Oracle guidelines.

For usage instructions, run `docker-storage-config` with the `-h` option:

```
sudo docker-storage-config -h
```

The `docker-storage-config` utility requires that you provide the path to a valid block device to use for Docker storage. The script formats the device with a new file system. This can be a destructive operation. Any existing data on the device may be lost. Use the `lsblk` command to help you correctly identify block devices currently attached to the system.

To automatically set up your Docker storage, before installation, run `docker-storage-config` as root:

```
sudo docker-storage-config -s btrfs -d /dev/sdb1
```

Substitute */dev/sdb1* with the path to the block device that you attached as dedicated storage.

You can substitute *btrfs* with `overlay2` if you would prefer to use this storage driver. If you do this, the block device is formatted with XFS and dtype support is enabled.

To overwrite an existing configuration, you can use the `-f` flag. If your Docker installation has already been used to set up images and containers, this option is destructive and may make these images and containers inaccessible to you, so the option should be used with caution.

Configuring Docker Storage Manually

This section discusses manually setting up a file system for Docker containers.

To manually prepare a dedicated file system to manage Docker containers:

1. Configure the Docker Engine to use Btrfs as the storage driver to manage containers. Use `yum` to install the `btrfs-progs` package:

   ```
   sudo yum install btrfs-progs
   ```

   If the root file system is not configured as a Btrfs file system, create a Btrfs file system on a suitable device or partition such as `/dev/sdb1` in this example:

   ```
   sudo mkfs.btrfs /dev/sdb1
   ```

2. Configure the Docker Engine to use a block device formatted with XFS in conjunction with the `overlay2` storage driver to manage containers. Format the block device with the XFS file system, for example to format a partition `/dev/sdb1`:

   ```
   sudo mkfs -t xfs -n ftype=1 /dev/sdb1
   ```

   It is essential that you use the `-n ftype=1` option when you create the file system or you cannot use overlayfs. To check if a mounted XFS partition has been

formatted correctly, run the following command and check the output to make sure that `ftype=1`:

```
xfs_info /dev/sdb1 | grep ftype
```

3. Use the `blkid` command to display the UUID and TYPE for the new file system and make a note of this value, for example:

```
blkid /dev/sdb1

/dev/sdb1: UUID="26fece06-e3e6-4cc9-bf54-3a353fdc5f82" TYPE="xfs" \
  PARTUUID="ee0d0d72-dc97-40d8-8cd9-39e29fbc660e"
```

The UUID for the file system on the device `/dev/sdb1` in this example is the `UUID` value `26fece06-e3e6-4cc9-bf54-3a353fdc5f82`. You can ignore the `PARTUUID` value, which is the UUID of the underlying partition. The TYPE of file system in this example is the `TYPE` value `xfs`.

4. Create an entry in your `/etc/fstab` file to make sure the file system is mounted at boot. Open `/etc/fstab` in an editor and add a line similar to the following:

```
UUID=UUID_value /var/lib/docker   fstype     defaults        0 0
```

Replace *UUID_value* with the UUID value. Replace *fstype* with the file system TYPE.

> **Note:**
>
> Previous versions of Docker required that dedicated storage used by Docker was mounted via a Systemd mount target and a Systemd drop-in file for the Docker service. This requirement was related to an issue where the storage was automatically unmounted when the Docker service was stopped. This issue no longer applies. If your storage is currently mounted using these methods, consider simplifying your environment by removing the Systemd drop-in and mount target and replacing this with an fstab entry.

This entry defines a mount for the file system on `/var/lib/docker`. You might need to create this directory if you are performing a fresh installation:

```
sudo mkdir /var/lib/docker
```

You must mount the file system to start using it:

```
sudo mount /var/lib/docker
```

Configuring a Docker Storage Driver

This section discusses setting up a storage driver for Docker.

To configure a Docker storage driver:

1. Create or edit `/etc/docker/daemon.json`.

If you are creating this file from scratch, it should look like this:

```
{
  "storage-driver": "btrfs"
}
```

Replace *btrfs* with your preferred storage driver. If you are using an XFS, ext3 or ext4 file system, you might replace *btrfs* with `overlay2`.

If this file already exists and contains other entries, be careful that adding a line for the `storage-driver` configuration variable conforms with typical JSON formatting.

For more information on the format and options for this configuration file, see:

https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file

2. Check that the `--storage-driver` option is not being invoked as a command line switch when starting the Docker Engine daemon.

   You should check that this option does not appear in either the `/etc/sysconfig/docker` or `/etc/sysconfig/docker-storage` files. These files are deprecated and may be removed in future releases. If these files contain any other configuration parameters, move these into `/etc/docker/daemon.json` to future-proof your configuration.

   Also check that this option does not appear in any systemd drop-in files in `/etc/systemd/system/docker.service.d/`. While this is a supported configuration option, it is preferable to keep all Docker Engine configuration consolidated and in the same place, where possible.

3. When you have started the Docker Engine and it is running, check that it is using the storage driver that you have configured:

   ```
   sudo docker info | grep Storage
   ```

   You can run the `docker info` command on its own to get a more detailed view of the configuration.

Excluding Docker Container Files From locate Output

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.

3
Upgrading Oracle Container Runtime for Docker

This chapter describes the steps required to perform an upgrade of Oracle Container Runtime for Docker on an Oracle Linux 7 host.

> **Note:**
>
> Docker requires that you configure the system to use the Unbreakable Enterprise Kernel Release 4 (UEK R4) or later and boot the system with this kernel.
>
> Using the Docker configuration files in `/etc/sysconfig` is deprecated. Instead, you should use the `/etc/docker/daemon.json` configuration file and `systemd` drop-in configuration files in `/etc/systemd/system/docker.service.d` as required.
>
> After adding or modifying a drop-in file while the `docker` service is running, run the command `systemctl daemon-reload` to tell `systemd` to reload the configuration for the service.

Upgrade Preqrequisites

Before upgrading, make sure you meet the requirements for the most current version of the Docker Engine. See the following sections to determine which steps may apply to your existing environment.

Updating the Unbreakable Enterprise Kernel

Configure the system to use the Unbreakable Enterprise Kernel Release 5 (UEK R5) or later and boot the system with this kernel. If you are using an earlier Unbreakable Enterprise Kernel (UEK) release, or the Red Hat Compatible Kernel (RHCK), you must upgrade the kernel.

To install or update UEK:

1. If your system is registered with ULN, disable access to the `ol7_x86_64_UEKR3` and `ol7_x86_64_UEKR4` channels, and enable access to the `ol7_x86_64_UEKR5` channel.

   Log into https://linux.oracle.com with your ULN user name and password and click on the Systems tab to select the system where you installing Oracle Container Runtime for Docker. Go to the **Manage Subscriptions** page and update the channel subscriptions for the system. Click on **Save Subscriptions** to save your changes.

2. If you use the Oracle Linux yum server, disable the `ol7_UEKR3` and `ol7_UEKR4` repositories and enable the `ol7_UEKR5` repository. You can do this easily using `yum-config-manager`:

```
sudo yum-config-manager --disable ol7_UEKR3 ol7_UEKR4
sudo yum-config-manager --enable ol7_UEKR5
```

3. Run the following command to upgrade the system to the selected UEK release:

   ```
   sudo yum update
   ```

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

   ```
   sudo systemctl reboot
   ```

Checking the Storage Driver

The Docker Engine uses `overlay2` as the default storage driver to manage Docker containers. The `overlay2` storage driver can run into issues on systems using an XFS formatted file system that is not created with the `-n ftype=1` option enabled. This is because overlay file systems depend on dtype support to handle metadata such as white outs for file deletion.

The root partition on Oracle Linux 7 is automatically formatted with `-n ftype=0` where XFS is selected as the file system, disabling dtype support. On new installations of Docker, the package installer checks the file system format options to ensure that dtype support is available. If dtype support is not enabled, the installer overrides the default storage driver to use `devicemapper` to ensure that Docker is ready-to-use on newly installed systems. However, upgraded versions of Docker continue to use the storage driver that was configured in the previous release. This means that if you have configured Docker to use `overlay2` on an underlying XFS-formatted file system, you may need to migrate the data to dedicated storage that has been formatted correctly.

Oracle recommends using Btrfs as a more stable and mature technology than overlayfs.

To check which storage driver and backing file system are configured on a running Docker Engine and to determine the path to the root Docker storage, run:

```
sudo docker info |grep 'Storage\|Filesystem\|Root'
```

If the storage driver is set to `overlay2` and the backing file system is set to `xfs`, check that the XFS file system is formatted correctly:

```
xfs_info /var/lib/docker |grep ftype
```

If necessary, replace */var/lib/docker* with the path to the root Docker storage returned in the previous command. If the information returned by this command includes `ftype=0`, you must migrate the data held in this directory to storage that is formatted with support for overlay filesystems.

To migrate the storage:

1. Attach a block storage device to the system where you are running Docker. Use the `lsblk` command to identify the device name and UUID. For example:

   ```
   lsblk -o 'NAME,TYPE,UUID,MOUNTPOINT'
   ```

   If necessary, you may need to partition the device using a partitioning tool such as `fdisk` or `parted`.

2. Format the block device with the XFS file system, for example to format a partition `/dev/sdb1`:

```
sudo mkfs -t xfs -n ftype=1 /dev/sdb1
```

It is essential that you use the `-n ftype=1` option when you create the file system or you will not be able to use overlayfs.

3. Temporarily mount the new file system, so that you can copy the contents from the existing Docker root directory:

```
sudo mount -t xfs /dev/sdb1 /mnt
```

4. Stop the Docker Engine, if it is running:

```
sudo systemctl stop docker
```

5. Move the existing Docker data to the new file system:

```
sudo mv /var/lib/docker/* /mnt
```

6. Unmount the new file system and remount it onto the Docker root directory:

```
sudo umount /mnt
sudo mount -t xfs /dev/sdb1 /var/lib/docker
```

7. Create an entry in your fstab to ensure that the file system is mounted at boot. Open `/etc/fstab` in an editor and add a line similar to the following:

```
UUID=UUID_value /var/lib/docker   xfs     defaults        0 0
```

Replace *UUID_value* with the UUID value for the partition that you created. Use the `lsblk` or `blkid` command if you need to check the value.

> **Tip:**
>
> If you do not have additional storage available for this purpose, it is possible to create an XFS file system image and loopback mount this. For example, to create a 25 GB image file in the root directory, you could use the following command:
>
> ```
> sudo mkfs.xfs -d file=1,name=/DockerStorage,size=25g -n ftype=1
> ```
>
> To temporarily mount this file, you can enter:
>
> ```
> sudo mount -o loop -t xfs /DockerStorage /mnt
> ```
>
> An entry in `/etc/fstab`, to make a permanent mount for Docker storage, may look similar to the following:
>
> ```
> /DockerStorage    /var/lib/docker        xfs     loop            0 0
> ```
>
> This configuration can help as a temporary solution to solve upgrade issues. However, using a loopback mounted file system image as a form of permanent storage for Docker is not recommended for production environments.

See Configuring Docker Storage for more information on setting up and configuring storage for Docker.

Upgrading the Docker Engine

To upgrade the Docker Engine:

1. Stop the `docker` service if it is running:

   ```
   sudo systemctl stop docker
   ```

2. Update the `docker-engine` and `docker-cli` packages:

   ```
   sudo yum update docker-engine docker-cli
   ```

3. Start the `docker` service:

   ```
   sudo systemctl start docker
   ```

4
Managing the Docker Engine Service

This chapter describes common Docker Engine administration and configuration tasks with specific focus on usage on Oracle Linux 7.

Configuring the Docker Engine Service

It is possible to configure the Docker Engine runtime options in a variety of ways. Where possible, Oracle recommends using the `/etc/docker/daemon.json` file to configure these options. For more information on the format and options for this configuration file, see https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file.

In rare instances, some runtime configuration options may not have an equivalent option that can be set in `/etc/docker/daemon.json`. Oracle previously allowed users to set these runtime options by editing variables in `/etc/sysconfig/docker`, `/etc/sysconfig/docker-network` and `/etc/sysconfig/docker-storage`. While these files can still be used for this purpose, they may be deprecated in future releases. Oracle recommends creating an alternate drop-in unit for the Docker Systemd service where you may need to specify alternate runtime options when loading the Docker Engine.

For example, you can create `/etc/docker/daemon.json` to contain the following content:

```
{
  "selinux-enabled": true
}
```

When you have finished editing the configuration file, reload to scan for new or changed units:

```
sudo systemctl daemon-reload
```

Finally, restart the Docker Engine service:

```
sudo systemctl restart docker
```

Reloading or Restarting the Docker Engine

If you change the Docker Engine configuration while the `docker` service is running, you must reload the service configuration to make the changes take effect.

To reload the `docker` service configuration, enter the following command:

```
sudo systemctl daemon-reload
```

If you do not reload the service configuration, `systemd` continues to use the original, cached configuration.

If you need to restart the `docker` service itself, enter the following command:

```
sudo systemctl restart docker
```

Enabling Non-root Users to Run Docker Commands

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

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

1. Create the `docker` group, if it does not already exist:

   ```
   sudo groupadd docker
   ```

2. Restart the `docker` service:

   ```
   sudo systemctl restart docker
   ```

   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:

   ```
   sudo usermod -a -G docker user1
   ```

Configuring User Namespace Remapping

To force processes running in Docker containers to run with an alternate user namespace mapping on the host system,use the `userns-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/engine/security/userns-remap/.

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.

   ```
   user:start_uid:uid_count
   ```

   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. Create or edit `/etc/docker/daemon.json`.

   If you are creating this file from scratch, it should look like this:

   ```
   {
     "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.

   If this file already exists and contains other entries, be careful that adding a line for the `storage-driver` configuration variable conforms with typical JSON formatting.

   For more information on the format and options for this configuration file, see https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file.

4. Check that the `--userns-remap` option is not being invoked as a command line switch when starting the Docker Engine daemon.

   You should check that this option does not appear in the `/etc/sysconfig/docker` file. This file is deprecated and may be removed in future releases. If this file contains any other configuration parameters, consider whether you could move these into `/etc/docker/daemon.json` to future-proof your configuration.

   Also check that this option does not appear in any systemd drop-in files in `/etc/systemd/system/docker.service.d/`. While this is a supported configuration option, it is preferable to keep all Docker Engine configuration in the same place, where possible.

5. Reload the `docker` service in systemd to activate changes to the service configuration:

   ```
   sudo systemctl daemon-reload
   ```

   If you need to restart the `docker` service itself, enter the following command:

   ```
   sudo systemctl restart docker
   ```

   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.

Enabling Live Restore for Containers

Docker has a `live-restore` option that can be used to keep containers running even if the Docker Engine daemon becomes unavailable. This option can help reduce container downtime due to crashes, planned outages and upgrades. To enable this facility you must edit `/etc/docker/daemon.json` and set the `"live-restore"` parameter to `true`. For more information on this facility, see https://docs.docker.com/config/containers/live-restore/.

Setting Container Registry Options

Oracle Container Runtime for Docker contains a number of configuration options that can be applied to the Docker Engine to control and customize the handling of commands to access a Docker registry.

Adding Registries

Oracle Container Runtime for Docker provides the option to connect to multiple registries to pull container images by configuring a registry list. By default, the Docker Engine is configured to pull images directly from the Docker Hub if no additional registries have been defined. You can configure a registry list to specify multiple registries that can be queried sequentially to pull an image. This can be used to configure the Docker Engine to first attempt to pull an image from a local registry and then fall back to an alternate registry, such as the Oracle Container Registry, before finally using the configured default registry. This is achieved by setting the `add-registry` option in `/etc/docker/daemon.json`.

```
...
  "add-registry": [
    "container-registry.oracle.com"
  ],
...
```

If you are creating this file from scratch with just the `add-registry` option, it would look like this:

```
{
  "add-registry": [
    "container-registry.oracle.com"
  ]
}
```

You can add multiple registries by appending the domain or domains you would like to add to the same list:

```
...
  "add-registry": [
    "container-registry.oracle.com",
    "registry.example.com"
  ],
...
```

Restart the Docker Engine service to apply your change:

```
sudo systemctl restart docker
```

Blocking Registries

Oracle Container Runtime for Docker provides the option to prevent access to specified registries when attempting to pull container images. This can be used to prevent users from pulling images from specific external registries. This is achieved by setting the `block-registry` option in `/etc/docker/daemon.json`.

```
...
  "block-registry": [
    "docker.io"
  ],
...
```

You can disable multiple registries by appending the domain or domains you would like to block to the same line:

```
...
  "block-registry": [
    "docker.io",
    "registry.example.com"
  ],
...
```

When you have finished editing `/etc/docker/daemon.json`, restart the Docker Engine service:

```
sudo systemctl restart docker
```

Setting the Default Registry

By default, the Docker Engine is configured to pull images directly from the Docker Hub if no additional registries have been defined.

It is possible to change the default registry by setting the `default-registry` option in `/etc/docker/daemon.json`.

```
...
  "default-registry": "test.registry.com",
...
```

Finally, restart the Docker Engine service:

```
sudo systemctl restart docker
```

When the default registry is changed, image references within the Docker Engine for images that have been pulled from the Docker Hub are updated to correctly display the docker.io prefix. For example `nginx:latest` is updated to reflect `docker.io/nginx:latest`. Images from the new default registry are displayed without a prefix.

The default registry determines the last possible registry that Docker Engine checks when you search for or pull an image. If you have configured multiple registries using the `add-registry` option then those registries are checked in sequential order, and if an image is not found in any of the other registries that you have been configured then the default registry is always used as the final option.

Adding Insecure Registries

Oracle Container Runtime for Docker provides the option to enable a registry that delivers containers over HTTPS but without any certificate validation, such as when using self-signed certificates for testing purposes, or to enable the use of registry that only uses HTTP. This is achieved using the `insecure-registry` option in `/etc/docker/daemon.json`.

```
...
  "insecure-registries" : ["insecure-registry.example.com"],
...
```

The `insecure-registry` option allows Docker to attempt an HTTPS connection to the registry, without any validation of the certificates presented by the registry. If the registry is not accessible via HTTPS, Docker falls back to attempt the connection using HTTP.

Restart the Docker Engine service to apply your changes:

```
sudo systemctl restart docker
```

5
Working With Containers and Images

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 7.

Pulling Oracle Linux Images From a Container Registry

You can get Oracle Linux images to run on the Docker Engine from the `oraclelinux` repository at the Docker Hub. For a list of the Oracle Linux images that are available, see https://hub.docker.com/_/oraclelinux/).

An Internet connection is required to pull images from the Docker Hub or the Oracle Container Registry. If you make use of a proxy server to access the Internet, see Configuring a Proxy Server.

Oracle Linux images, along with many other Oracle product images, are also hosted on the Oracle Container Registry at https://container-registry.oracle.com and on the Docker Hub at https://hub.docker.com. More information on using the Oracle Container Registry to pull images is covered in Pulling Images From the Oracle Container Registry. See Using Third-Party Registries for more information on using the Docker Hub.

To download an Oracle Linux image, use the `docker pull` command. For example, to pull an Oracle Linux image from the Docker Hub:

```
docker pull oraclelinux:7-slim

Trying to pull repository docker.io/library/oraclelinux ...
7-slim: Pulling from docker.io/library/oraclelinux
977461c90301: Pull complete
Digest: sha256:0743f72832d8744a89b7be31b38b9fb2e5390044cbb153cd97b3e797723e4704
Status: Downloaded newer image for oraclelinux:7-slim
```

To display a list of the images that you have downloaded to a system, use the `docker images` command, for example:

```
docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
oraclelinux         7-slim              c2b5cb5bcd9d        7 days ago          118MB
oraclelinux         7                   31f4bed1dc33        7 days ago          232MB
oraclelinux         latest              31f4bed1dc33        7 days ago          232MB
oraclelinux         8                   8988c7081e1f        5 weeks ago         411MB
```

Each image in the repository is distinguished by its `TAG` value and its unique `IMAGE ID`. In the example, the tags `7` and `latest` refer to the same image ID for Oracle Linux 7.

When new images are made available for Oracle Linux updates, the tags `7`, `8`, and `latest` are updated in the `oraclelinux` repository to refer to the appropriate newest version.

If an image is downloaded from an alternate registry to the default registry, the `REPOSITORY` value also indicates the registry from which the image was pulled. For example:

```
docker images

REPOSITORY                                      TAG         IMAGE ID
CREATED       SIZE
container-registry.oracle.com/os/oraclelinux    latest      31f4bed1dc33    7 days
ago    232MB
```

See Setting Container Registry Options for more information on adding registries and configuring a default registry.

Enabling or Disabling Docker Content Trust

Content Trust 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://docs.docker.com/engine/security/trust/content_trust/.

Enabling FIPS Mode in Containers

To run containers in FIPS mode, you must first enable FIPS mode on your Oracle Linux host system.

For more information about Oracle Linux 7 releases that have FIPS validated cryptographic modules available for installation, see Oracle Linux 7: Security Guide.

> **Note:**
>
> Oracle provides FIPS compliant container images by using the `slim-fips` tag. Container images tagged as FIPS compliant include compliant cryptographic package versions and initial image setup required for container FIPS mode. See The slim Tag for more information.

For Oracle Linux 7 Containers:

To enable FIPS mode in an Oracle Linux 7 container, install the `dracut-fips` package or mount `/etc/system-fips` from the host. For more information about mounting host files and directories from inside a container, see Accessing External Files From Docker Containers.

For Oracle Linux 8 Containers:

To enable FIPS mode in an Oracle Linux 8 container, mount `/etc/system-fips` from the host. For more information about mounting host files and directories from inside a container, see Accessing External Files From Docker Containers.

Additionally, bind mount FIPS cryptographic policies within the container from `/usr/share/crypto-policies/back-ends/FIPS` to `/etc/crypto-policies/back-ends`:

```
mount --bind /usr/share/crypto-policies/back-ends/FIPS /etc/crypto-policies/back-ends
```

Creating and Running Docker Containers

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

```
docker run -i -t --name guest oraclelinux:7-slim

bash-4.2# cat /etc/oracle-release
Oracle Linux Server release 7.7
bash-4.2# exit
```

This example runs an interactive `bash` shell using the Oracle Linux 7 image named `oraclelinux:7-slim` to provide the container. The `/bin/bash` command is the default command run for all `oraclelinux` base images. The `-t` and `-i` options allow you to use a pseudo-terminal to run the container interactively.

The following examples may use the prompt `[root@host ~]` and `[root@guest ~]` (or similar) to represent the prompts shown by the host and by the container respectively. The actual prompt displayed by the container may 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, for example:

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

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 --rm container-registry.oracle.com/os/oraclelinux:7-slim

Unable to find image 'container-registry.oracle.com/os/oraclelinux:7-slim' locally
Trying to pull repository container-registry.oracle.com/os/oraclelinux ...
7-slim: Pulling from container-registry.oracle.com/os/oraclelinux
Digest: sha256:267f37439471f1c5eae586394c85e743b887c7f97e4733e10e466158083c021e
```

```
Status: Downloaded newer image for container-registry.oracle.com/os/
oraclelinux:7-slim
[root@guest /]# cat /etc/oracle-release
Oracle Linux Server release 7.7
[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
68359521c0b7  oraclelinux:7-slim  "/bin/bash" 2 hours ago   Up 8 minutes
guest
```

The container named `guest` with the ID `68359521c0b7` 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   31252  31235  0   05:59   pts/0   00:00:00   /bin/bash
```

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 ~]#
```

You can also 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:7-slim
b4c224f83e35927f67b973febb006b0af4d037f41c30e1f4bdcc4b822e12fd0f
[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
b4c224f83e35  oraclelinux:7-slim ...      ...      Exited (0) About a minute
```

```
ago     newguest
68359521c0b7  oraclelinux:7-slim ...      ...      Exited (137) 45 seconds ago
guest
```

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 Nov 26 06:27 /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.

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

```
[root@host ~]# docker logs -f guest
bash-4.2# exit
exit
bash-4.2# ls -l /tmp/foobar
-rw-r--r-- 1 root root 0 Nov 26 06:33 /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
```

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.
>
> 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`.

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` and `docker create`:

`--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.

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 to restrict the capabilities of a container, for example:

```
[root@host ~]# docker run --cap-add=ALL --cap-drop=NET_ADMIN -i -t --rm
oraclelinux:7
[root@guest /]# ip route del default
RTNETLINK answers: Operation not permitted
```

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` and `docker create`:

`--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.

For example, `--device=/dev/sdd:/dev/xvdd:r` would make the host device `/dev/sdd` available to the container as the device `/dev/xvdd` with read-only permission.

NOT_SUPPORTED:

Do not make block devices that can easily be removed from the system available to untrusted containers.

Accessing the Host's Process ID Namespace

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.

NOT_SUPPORTED:

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

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

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.

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 a container based on the `oraclelinux:7-slim` image so that it can run an Apache HTTP server. After stopping the container, the image `mymod/httpd:v1` is created from it.

Tip:

The `oraclelinux:7-slim` and `oraclelinux:8-slim` images provide the bare minimum operating system required for Oracle Linux 7 and Oracle Linux 8. Using these images can help to reduce resource usage when running containers based on them. You can also ensure that the image that you create is limited to the base requirements for your application.

To create an Apache server image from an `oraclelinux:7-slim` container:

1. Run the `bash` shell inside a container named `httpd1`:

```
docker run -i -t --name httpd1 oraclelinux:7-slim /bin/bash

[root@httpd1 ~]#
```

2. If you use a web proxy, edit the yum configuration on the guest as described in Oracle Linux 7: Managing Software.
3. Install the `httpd` package:

```
[root@httpd1 ~]# yum -y 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 simply using the `exit` command from within the interactive guest session:

```
[root@httpd1 ~]# exit
exit
```

   Or by using the `docker stop` command on the host:

```
docker stop httpd1
```

6. Create the image `mymod/httpd` with the tag `v1` using the ID of the container that you stopped:

```
docker commit -m "ol7-slim + httpd" -a "A N Other" \
  `docker ps -l -q` mymod/httpd:v1

sha256:b03fbc3216882a25e32c92caa2e797469a1ac98e5fc90affa07263b8cb0aa799
```

   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.

> **Tip:**
>
> The `docker ps -l -q` command returns the ID of the last created container. We used this command in the example to obtain the ID of the container that we wanted to use to generate the image. You may, alternatively, specify the ID directly or use an alternate variation on this command to obtain the correct ID.

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

```
docker images

REPOSITORY          TAG                 IMAGE ID
CREATED             SIZE
mymod/httpd         v1                  b03fbc321688        2 minutes
ago       426MB
oraclelinux         7-slim              c2b5cb5bcd9d        7 days
ago          118MB
```

7. Remove the container named `httpd1`.

```
docker rm httpd1
```

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

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

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`, for example:

```
docker ps

CONTAINER ID  IMAGE           COMMAND                  CREATED        STATUS
PORTS                 NAMES
154f05ea464e  mymod/httpd:v1  "/usr/sbin/httpd -D …"  2 minutes ago  Up 2 minutes
0.0.0.0:8080->80/tcp  newguest
```

Alternately, use the `docker port` command, for example:

```
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:

```
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:

```
firewall-cmd --zone=public --permanent --add-port=8080/tcp
```

If you need to remove an image, use the `docker rmi` command:

```
docker rmi mymod/httpd:v1

Untagged: mymod/httpd:v1
Deleted: sha256:b03fbc3216882a25e32c92caa2e797469a1ac98e5fc90affa07263b8cb0aa799
Deleted: sha256:f10c5b69ca9c3df53412238eefac72522720bc7c1a6a8eb6d21801c23a81c126
```

> **Note:**
>
> 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 Creating a Docker Image From a Dockerfile.

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:7-slim` 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:

   ```
   # Dockerfile that modifies oraclelinux:7-slim to include an Apache HTTP
   server
   FROM oraclelinux:7-slim
   MAINTAINER A N Other <another@example.com>
   RUN sed -i -e '/^\[main\]/aproxy=http://proxy.example.com:80' /etc/yum.conf
   RUN yum -y install httpd
   RUN echo "HTTP server running on guest" > /var/www/html/index.html
   EXPOSE 80
   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:

   `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.

   `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 to access the port over Docker's internal network (see Communicating Between Docker Containers).

   `FROM`

   Defines the image that Docker uses as a basis for the new image.

MAINTAINER

Defines who is responsible for the Dockerfile.

RUN

Defines the commands that Docker runs to modify the new image. In the example, the `RUN` lines set up the web proxy, install the `httpd` package, and create a simple home page for the server.

For more information about other instructions that you can use in a Dockerfile, see https://docs.docker.com/engine/reference/builder/.

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

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

Sending build context to Docker daemon  2.048kB
Step 1/6 : FROM oraclelinux:7-slim
Trying to pull repository docker.io/library/oraclelinux ...
7-slim: Pulling from docker.io/library/oraclelinux
a8d84c1f755a: Pull complete
Digest: sha256:d574213fa96c19ae00269730510c4d81a9979ce2a432ede7a62b62d594cc5f0b
Status: Downloaded newer image for oraclelinux:7-slim
 ---> c3d869388183
Step 2/6 : MAINTAINER A N Other <another@example.com>
 ---> Running in 26b0ba9f45e8
Removing intermediate container 26b0ba9f45e8
 ---> f399f426b849
Step 3/6 : RUN yum -y install httpd
 ---> Running in d75a9f312202
Loaded plugins: ovl
Resolving Dependencies
--> Running transaction check
---> Package httpd.x86_64 0:2.4.6-88.0.1.el7 will be installed
...
Complete!
Removing intermediate container d75a9f312202
 ---> aa3ab87bcae3
Step 4/6 : RUN echo "HTTP server running on guest" > /var/www/html/index.html
 ---> Running in dddedfc56849
Removing intermediate container dddedfc56849
 ---> 8fedc8516013
Step 5/6 : EXPOSE 80
 ---> Running in 6775d6e3996f
Removing intermediate container 6775d6e3996f
 ---> 74a960cf0ae9
Step 6/6 : ENTRYPOINT /usr/sbin/httpd -D FOREGROUND
 ---> Running in 8b6e6f61a2c7
Removing intermediate container 8b6e6f61a2c7
 ---> b29dea525f0a
Successfully built b29dea525f0a
Successfully tagged mymod/httpd:v2
```

Having built the image, you can test it by creating a container instance named `httpd2`:

```
docker run -d --name httpd2 -P mymod/httpd:v2
```

> **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 a random available high-order port (higher than 30000) on the host.

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

```
docker inspect --format='{{ .NetworkSettings.Ports }}' httpd2

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.

You can open the port by updating the firewall:

```
firewall-cmd --add-port=49153/tcp
firewall-cmd --permanent --add-port=49153/tcp
```

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

```
curl http://localhost:49153

HTTP server running on guest
```

Creating Multi-Stage Docker Image Builds

From Oracle Container Runtime for Docker 17.06, it is possible to perform multi-stage builds from a single Dockerfile. This allows you to perform interim build or compilation steps during the creation of the final image, without including all of the build tools and artifacts in the final image. This helps to reduce image sizes, and improves performance. It also allows you to deliver an image containing only the required binary and not all of the layers that were required to produce the binary.

In this section, we provide a very simple example scenario, where the source of a program is built in an interim compiler image and the resulting binary is copied into a separate image to produce the final target image. This entire build is handled by a single Dockerfile.

Create a simple "hello world" style program in C, by pasting the following text into a file named `hello.c`:

```
#include <stdio.h>

int
main (void)
{
  printf ("Hello, world!\n");
```

```
  return 0;
}
```

Create a Dockerfile that contains the following text:

```
FROM gcc AS BUILD
COPY . /usr/src/hello
WORKDIR /usr/src/hello
RUN gcc -Wall hello.c -o hello

FROM oraclelinux:7-slim
COPY --from=BUILD /usr/src/hello/hello hello
CMD ["./hello"]
```

Note that there are two `FROM` lines in this Dockerfile. The first `FROM` statement pulls the latest `gcc` image from the Docker hub and uses the `AS` syntax to assign it a name that we can refer to later when copying elements from this temporary build environment to our target image.

In the build environment, the source file is copied into the image and the `gcc` compiler is run against the source file to produce a `hello` binary.

The second `FROM` statement pulls the `oraclelinux:7-slim` image. This image is used to host the `hello` binary, which is copied into it directly from the build environment. By doing this, the source, the compiler and any other build artifacts can be excluded from the final image.

To build the new image and run it, try running the following:

```
docker build -t hello-world ./

Sending build context to Docker daemon  35.38MB
Step 1/7 : FROM gcc AS BUILD
 ---> 7d9419e269c3
Step 2/7 : COPY . /usr/src/hello
 ---> ee7310cc4464
Removing intermediate container 1d51e6f16833
Step 3/7 : WORKDIR /usr/src/hello
 ---> 2c0298733ba0
Removing intermediate container 46a09ccc06d6
Step 4/7 : RUN gcc -Wall hello.c -o hello
 ---> Running in f003deeebc20
 ---> 67c85367cac1
Removing intermediate container f003deeebc20
Step 5/7 : FROM oraclelinux:7-slim
 ---> da5e55a16f7a
Step 6/7 : COPY --from=BUILD /usr/src/hello/hello hello
 ---> 8bd284b0d7eb
Removing intermediate container d71eee578325
Step 7/7 : CMD ./hello
 ---> Running in d6051d9e0a9d
 ---> dac5aa2d651d
Removing intermediate container d6051d9e0a9d
Successfully built dac5aa2d651d
Successfully tagged hello-world:latest

docker run hello-world

Hello, world!
```

The `hello-world` image is generated to contain and run the `hello` binary, but doesn't contain any of the components that were required to build the binary. The final image has less layers, is smaller and excludes any of the build steps in its history.

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. Containers on the default bridge network can communicate with each other on this network directly, although there is domain name resolution within this network to make containers specifically aware of each other.
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. More information on user-defined networking is provided in Communicating Between Docker Containers.

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 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/.

Communicating Between Docker Containers

All containers are automatically added to the default bridge network and assigned IP addresses by the Docker Engine. This means that containers are effectively able to communicate directly using the bridge network. However there is no automatic service discovery on the default bridge network. If containers need to be able to resolve IP addresses by container name, you should use a user-defined network instead.

You can use the `--link` option with `docker run` to make network connection information about a server container available to a client container. For example to link a client container, client1, to a server container, httpd_server, you could run:

```
docker run --rm -t -i --name client1 --link http-server:server oraclelinux /bin/
bash
```

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 server container name and IP address are also set in `/etc/hosts` in the client container, to facilitate easy access.

The `--link` option is considered a legacy feature and may be deprecated in future releases. It is not recommended in most cases.

The preferred approach to setting up communications between containers is to create user-defined networks. These networks provide better isolation and can perform DNS resolution of container names to IP addresses. A variety of network drivers are available, but the most commonly used is the bridged network which behaves similarly to the default bridge network but which provides additional features.

The following example shows how to create a simple user-defined network bridge and how to connect containers to it, to allow them to communicate easily with each other.

1. Create a network using the bridge driver.

   ```
   docker network create --driver bridge http_network
   ```

   In the example, the network is named *http_network*.

   You can check that the network has been created and which driver it is using:

   ```
   docker network ls

   NETWORK ID          NAME                DRIVER              SCOPE
   094c50739e14        bridge              bridge              local
   7eff8115af9a        host                host                local
   4a03450bf054        http_network        bridge              local
   457c4070f5a2        none                null                local
   ```

   You can also inspect the network object to discover more information:

   ```
   docker network inspect http_network

   [
       {
   ```

```
        "Name": "http_network",
        "Id":
"4a03450bf054a6d4d4db52da36eab8d934d35bf961b3b3adb4fe20be54c0fdac",
        "Created": "2019-02-06T04:40:47.177691733-08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

2. Connect existing containers to the user-defined network.

```
docker network connect http_network http-server
docker network connect http_network client1
```

In this example, *http-server* and *client1* are existing containers that are connected to the newly created *http_network* bridge network.

3. Connect a new container to the user-defined network, using the `--network` option.

```
docker run --rm -t -i --name client2 --network http_network
oraclelinux:7 /bin/bash
```

You can check that domain name resolution is working from within the container by pinging any other container on the network by its container name:

```
[root@client1 ~]# ping -c 1 http-server

PING http-server (172.18.0.2) 56(84) bytes of data.
64 bytes from http-server.http_network (172.18.0.2): icmp_seq=1 ttl=64
time=0.162 ms

--- http-server ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.162/0.162/0.162/0.000 ms
```

You can access services on containers within the network using their container names. For example:

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

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

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:

```
echo "This text was created in a file on the host" > /var/www/html/index.html
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 ~]# systemctl status httpd
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:

```
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:

```
echo "Change the file on the host" > /var/www/html/index.html

[root@host ~]# curl http://localhost:49153
Change the file on the host
```

Even if you delete the file on the host, it is still visible in the container:

```
rm -f /var/www/html/index.html

[root@host ~]# ls -l /var/www/html/index.html
ls: cannot access /var/www/html/index.html: No such file or directory
[root@host ~]# curl http://localhost:49153
Change the file on the host
```

It is not possible to use a Dockerfile to define how to mount a file or file system from a host. Docker applications are intended to be portable and it is unlikely that a file or file system that exists on the original host would be available on another system. If you want external file data to be portable, you can encapsulate it in a *data volume container*. See Creating and Using Data Volume Containers.

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 named `Dockerfile` that defines the image for a data volume container:

   ```
   # Dockerfile that modifies oraclelinux:7-slim to create a data volume
   container
   FROM oraclelinux:7-slim
   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 `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:

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

```
Sending build context to Docker daemon  2.048kB
Step 1/8 : FROM oraclelinux:7-slim
 ---> c2b5cb5bcd9d
Step 2/8 : MAINTAINER A N Other <another@example.com>
 ---> Running in 56c7b79c246e
Removing intermediate container 56c7b79c246e
 ---> 620ff82e21cb
Step 3/8 : RUN mkdir -p /var/www/html
 ---> Running in ac91306f3d74
Removing intermediate container ac91306f3d74
 ---> 379c58d9eab9
Step 4/8 : RUN echo "This is the content for file1.html" > /var/www/html/file1.html
 ---> Running in 981773ba0210
Removing intermediate container 981773ba0210
 ---> 2ee97d83b582
Step 5/8 : RUN echo "This is the content for file2.html" > /var/www/html/file2.html
 ---> Running in 36e8550c9a8b
Removing intermediate container 36e8550c9a8b
 ---> 4ba8d28df981
Step 6/8 : RUN echo "This is the content for index.html" > /var/www/html/index.html
 ---> Running in 6f15a403b4f6
Removing intermediate container 6f15a403b4f6
 ---> 550bb92c154b
Step 7/8 : VOLUME /var/www/html
 ---> Running in 1806e5d6e643
Removing intermediate container 1806e5d6e643
 ---> 0e3de4ac4c9c
Step 8/8 : ENTRYPOINT /usr/bin/tail -f /dev/null
 ---> Running in 6cde4f965504
Removing intermediate container 6cde4f965504
 ---> 5e4e2780503b
Successfully built 5e4e2780503b
Successfully tagged mymod/dvc:v1
```

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

```
docker run -d --name dvc1 mymod/dvc:v1 tail -f /dev/null
```

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`.

```
docker run -d --volumes-from dvc1 --name websvr -P mymod/httpd:v2
```

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. For example:

```
[root@host ~]# docker port websvr 80
0.0.0.0:32769
[root@host ~]# curl http://localhost:32769
This is the content for index.html
[root@host ~]# curl http://localhost:32769/file1.html
This is the content for file1.html
[root@host ~]# curl http://localhost:32769/file2.html
This is the content for file2.html
```

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 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:

```
docker run --rm --volumes-from dvc1 -v /var/tmp:/host:rw oraclelinux:7-slim \
   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:

```
docker run --rm --volumes-from dvc2 -v /var/tmp:/host:ro \
  oraclelinux:7-slim 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` (`/var/tmp/dvc1_files` on the host) 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:

```
docker run --rm --volumes-from dvc1 -v /var/tmp:/host:rw \
  oraclelinux:7-slim tar -cPvf /host/dvc1_files.tar /var/www/html

ls -l /var/tmp/dvc1_files.tar

-rw-r--r--. 1 root root 10240 Aug 31 14:37 /var/tmp/dvc1_files.tar

docker run --rm --volumes-from dvc2 -i -t --name guest -v /var/tmp:/host:ro \
  oraclelinux:7-slim /bin/bash
```

On the guest, run the following commands:

```
[root@guest ~]# rm /var/www/html/*.html
[root@guest ~]# ls -l /var/www/html/*.html
total 0
[root@guest ~]# tar -xPvf /host/dvc1_files.tar
var/www/html/
var/www/html/file1.html
var/www/html/file2.html
var/www/html/index.html
[root@guest ~]# 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
[root@guest ~]# exit
exit
```

This example uses a transient, interactive container named `guest` to extract the contents of the archive to `dvc2`.

Using Labels to Define Metadata

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\ 7"
```

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
7"
```

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'"
```

Alternately, you can append these options to a list in the `/etc/docker/daemon.json` file, for example:

```
{
  "labels": ["com.mydom.dept='DEVGROUP'", "com.mydom.version='1.0.0-ga'"]
}
```

> **Note:**
>
> After adding or modifying a configuration file while the `docker` service is running, run the command `systemctl daemon-reload` to tell `systemd` to reload the configuration for the service.

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.

Defining the Logging Driver

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`.

About Image Digests

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.

Specifying Control Groups for Containers

You can use the `--cgroup-parent` option with the `docker create` command to specify the control group (*cgroup*) in which a container should run.

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.

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.

Enabling a Container to 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. This setting allows the container to track the UTS namespace of the host or to set the host name and domain from the container.

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

Setting ulimit Values on Containers

The `--ulimit` option to `docker run` allows you to specify `ulimit` values for a container, for example:

```
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.

You can set default `ulimit` values for all containers by specifying `default-ulimits` options in a `/etc/docker/daemon.json` configuration file, for example:

```
"default-ulimits": {
  "nofile": {
      "Name": "nofile",
      "Hard": 128,
      "Soft": 256
    },
    "nproc" : {
      "Name": "nproc",
      "Hard": 32,
      "Soft": 64
    }
},
```

> **Note:**
>
> After adding or modifying the configuration file while the `docker` service is running, run the command `systemctl daemon-reload` to tell `systemd` to reload the configuration for the service.

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

Building Images With Resource Constraints

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 ID     NAME          CPU %      MEM USAGE/LIMIT     MEM %     NET
I/O            BLOCK I/O     PIDS
1ab12958b915     cntr1         0.05%      504 KiB/128 MiB     0.39%     2.033
KiB/40 B     13.7MB/1MB    1
3cf41296a324     cntr2         0.08%      1.756 MiB/128 MiB   1.37%     5.002
KiB/92 B     15.8MB/3MB    1
```

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 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
```

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'" 7ac15076dcc1 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`.

6
Using Docker Registries

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 the Docker Hub and is available at:

https://hub.docker.com

Oracle makes open source software available on the GitHub Container registry. See https://github.com/oracle/docker-images and https://docs.github.com/en/packages/guides/about-github-container-registry for more information.

Oracle also hosts its own Docker registry, the Oracle Container Registry, which contains both licensed and open source Oracle software. The Oracle Container Registry is located at:

https://container-registry.oracle.com

You can configure multiple registries when pulling images. See Setting Container Registry Options for more information on using multiple registries.

The Oracle Container Registry provides a web interface that allows an administrator to select the images for the software that your organization wants to use.

If you want to use licensed Oracle software images, you must first log into the Oracle Container Registry web interface and accept the Oracle Standard Terms and Restrictions for the software images.

Open source software images, and all of the software an image contains, is licensed under one or more open source license, provided in the container image. Your use of the container image is subject to the terms of those licenses.

You can use one of the Oracle Container Registry mirrors for faster download in your geographical region.

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.

Users of Oracle Cloud Infrastructure can use the Registry service for an Oracle-managed Docker registry that can serve images to your internal compute instances and can be exposed as a public registry on the Internet if required. The Oracle Cloud Infrastructure Registry service includes fine grained policy controls to allow you to control registry access. For complete documentation around using this service, see https://docs.oracle.com/iaas/Content/Registry/Concepts/registryoverview.htm

Pulling Images From the Oracle Container Registry

This section discusses pulling an image from the Oracle Container Registry.

If you are pulling a licensed Oracle software image, you must first log into the Oracle Container Registry and accept the Oracle Standard Terms and Restrictions. For information on pulling licensed Oracle software from the Oracle Container Registry, see Pulling Licensed Software From the Oracle Container Registry.

To pull an image from the Oracle Container Registry:

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

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

```
docker pull container-registry.oracle.com/os/oraclelinux:7-slim
```

The *area* and *image* are nearly always specified in lower case. The command to pull an image is usually provided on the repository information page in the Oracle Container Registry web interface. Other useful information about the image and how it should be run may also be available on the same page.

Using Oracle Container Registry Notary for Content Trust

The Oracle Container Registry also includes a Notary service that can be used to validate signed images that are pulled from the registry. This service helps to improve security and can mitigate against inadvertently running a compromised image on your infrastructure.

Using the Notary service is straightforward and only requires that you set two environment variables for the user that runs any Docker commands.

1. Set the `DOCKER_CONTENT_TRUST` environment variable to enable content trust within Docker:

   ```
   export DOCKER_CONTENT_TRUST=1
   ```

   See Enabling or Disabling Docker Content Trust for more information.

2. Set the `DOCKER_CONTENT_TRUST_SERVER` to point to the Notary service. It is important that you specify the port number in the value that you provide for this variable:

   ```
   export DOCKER_CONTENT_TRUST_SERVER="https://container-
   trust.oci.oraclecloud.com:443"
   ```

3. Pull an image from the container registry, as described in Pulling Images From the Oracle Container Registry to update the Notary metadata cache. If content trust is working correctly, the image pull will behave normally. If the Notary is unavailable, Docker falls back to its local cache of signature metadata and issues a warning:

   ```
   WARN[0015] Error while downloading remote metadata, using cached timestamp
       - this might not be the latest version available remotely
   ```

   If you try to pull an image that is not trusted or not hosted on the Oracle Container Registry an error is displayed and the image cannot be pulled. For example, when pulling an image from docker.io with content trust enabled and the content trust server configured for the Oracle Container Registry Notary service, the following error is returned:

```
Error: remote trust data does not exist for nginx: container-
trust.oci.oraclecloud.com:443
does not have trust data for docker.io/library/nginx
```

A user can explicitly disable content trust when running a Docker command by specifying the `--disable-content-trust` option or can simply unset the environment variable. Equally, content trust can be forced, regardless of the environment variable setting by running a command with `--disable-content-trust=false` as an option.

Pulling Licensed Software From the Oracle Container Registry

The Oracle Container Registry contains images for licensed commercial Oracle software products. To pull images for licensed software on the Oracle Container Registry, you must have an Oracle Account. You can create an Oracle Account using:

https://profile.oracle.com/myprofile/account/create-account.jspx

> **Note:**
>
> You do not need to log into the Oracle Container Registry or accept the Oracle Standard Terms and Restrictions to pull open source Oracle software images.

To pull a licensed software image from the Oracle Container Registry:

1. In a web browser, log into the Oracle Container Registry using your Oracle Account:

   https://container-registry.oracle.com

2. Use the web interface to accept the Oracle Standard Terms and Restrictions for the Oracle software images you want to pull. Your acceptance of these terms are stored in a database that links the software images to your Oracle Account. Your acceptance of the Oracle Standard Terms and Restrictions is valid only for the repositories for which you accept the terms. You may need to repeat this process if you attempt to pull software from alternate or newer repositories in the registry. This is subject to change without notice.
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 Oracle Account you used to log into the web interface:

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

   You are prompted for the username and password for the Oracle Account.
5. Pull the images your require using the `docker pull` command. For example:

   ```
   docker pull container-registry.oracle.com/java/serverjre
   ```

   For more detailed information on pulling images from the Oracle Container Registry, see Pulling Images From the Oracle Container Registry.

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

6. After you have pulled images from the Oracle Container Registry, it is good practice to log out of the registry to prevent unauthorized access, and to remove any record of your credentials that Docker may store for future operations:

```
docker logout container-registry.oracle.com
```

Using the Oracle Container Registry Mirrors

The Oracle Container Registry has many mirror servers located around the world. You can use a registry mirror in your global region to improve download performance of container images.

To get a list of the available mirrors, and the command to pull the image from the mirror, see the information page for an image using the Oracle Container Registry web interface. The list of registry mirrors is available towards the end of the image information page, in the `Tags` table. The table heading includes a `Download Mirror` drop down to select a registry mirror. When you select a mirror, the `Pull Command` column changes to show the command to pull the image from the selected mirror.

Pull an image from an Oracle Container Registry mirror using the URL for that mirror. For example, to pull the Oracle Linux 7 image from the Sydney mirror, use:

```
docker pull container-registry-sydney.oracle.com/os/oraclelinux:7-
slim
```

To download licensed Oracle software images from a registry mirror, you must first accept the Oracle Standard Terms and Restrictions in the Oracle Container Registry web interface.

https://container-registry.oracle.com

To pull licensed Oracle software images, log in to the Oracle Container Registry mirror before you pull the image. For example:

```
docker login container-registry-sydney.oracle.com
docker pull container-registry-sydney.oracle.com/java/serverjre
docker logout container-registry-sydney.oracle.com.oracle.com
```

Using Third-Party Registries

There are several third-party registries that are available for use with Oracle Container Runtime for Docker, such as the Docker Hub and the GitHub Container Registry. Oracle makes images for some licensed commercial Oracle software products on some of these third-party registries.

GitHub Container Registry

GitHub is a popular open-source development platform and is one of Oracle's preferred repositories for open source software. GitHub Container Registry provides an industry standard registry that you can use to pull images for the containers that you need to run in your environment.

No authentication is required to pull an image from GitHub Container Registry. For example, to pull the slim Oracle Linux 8 image from the GitHub Container Registry, use:

```
docker pull ghcr.io/oracle/oraclelinux:8-slim
```

For more information on GitHub Container Registry, see https://docs.github.com/en/packages/guides/about-github-container-registry

Docker Hub

The Docker Hub is available at:

https://hub.docker.com

You are able to browse the Docker Hub, but to access many of the images hosted there, you must log in with a valid Docker ID. If you do not have a Docker ID, you can register at:

https://hub.docker.com/signup

The Docker Hub provides a web interface that allows you to select the Docker Certified images that you want to install, and to agree to any terms and conditions that may apply, or to make payment if required. When you have agreed to the terms and conditions that apply to an image, the image is stored in the `My Content` area, so that you can revisit it later.

The Docker Hub may require that you are logged in before you can pull any Docker Certified images hosted in this registry. This makes sure the terms and conditions that apply to the image have been accepted, and that any payments have been settled.

Since the Docker Hub is the default registry, you can pull an image by simply specifying its name and tag. For example, to pull the `oraclelinux:7-slim` image:

```
docker pull oraclelinux:7-slim

Trying to pull repository docker.io/library/oraclelinux ...
7-slim: Pulling from docker.io/library/oraclelinux
a61503a3b32e: Pull complete
Digest: sha256:bb7c3969d33b3c2695b11dd705e18ed604ce0f1e3317ef293e8f0d9d125dc90a
Status: Downloaded newer image for oraclelinux:7-slim
oraclelinux:7-slim
```

Setting Up a Local Docker Registry

This section contains information about setting up a local Docker registry server, which can be used to host your own images, and can also be used as a mirror for the Oracle Container Registry.

Users of Oracle Cloud Infrastructure should consider using the Oracle-managed Registry service to cater to local Docker registry requirements. See https://docs.oracle.com/iaas/Content/Registry/Concepts/registryoverview.htm.

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

Creating a Registry File System

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 file system for this. It is recommended you create a Btrfs formatted file system to allow you to easily scale your registry file system, and to leverage Btrfs features such as snapshotting. The instructions in this section provide details for setting up a Btrfs file system. The device could be a disk partition, an LVM volume, a loopback device, a multipath device, or a LUN.

If you want dedicated storage for the registry file system, create a file system and mount it at `/var/lib/registry`. This example uses Btrfs to format the file system.

To create a Btrfs file system for the registry:

1. 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 on the partition `/dev/sdc1`:

   ```
   sudo 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 on a logical volume named `docker-registry` in the `ol` volume group:

   ```
   sudo 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 using `mkfs.btrfs`, see Oracle Linux 7: Managing File Systems.

2. 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:

   ```
   sudo blkid /dev/sdc1

   /dev/sdc1: LABEL="var-lib-registry" UUID="50041443-b7c7-4675-95a3-
   bf3a30b96c17" \
   UUID_SUB="09de3cb1-2f9b-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.

3. Edit the `/etc/fstab` file and add an entry to make sure the file system is mounted when the system boots.

   ```
   UUID=UUID_value /var/lib/registry  btrfs  defaults  0 0
   ```

   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:

   ```
   LABEL=label /var/lib/registry  btrfs  defaults  0 0
   ```

4. Create the `/var/lib/registry` directory.

   ```
   sudo mkdir /var/lib/registry
   ```

5. Mount all the file systems listed in `/etc/fstab`.

```
sudo mount -a
```

6. Verify that the file system is mounted.

```
df

Filesystem     1K-blocks    Used Available Use% Mounted on
...
/dev/sdc1            ...    ...  ...       1%   /var/lib/registry
```

Setting Up Transport Layer Security for the Docker Registry

The registry host requires a valid X.509 certificate and private key to enable Transport Layer Security (TLS) with the registry, similar to using TLS for a web server. This section discusses adding the host's X.509 certificate and private key to Docker.

If the host already has an X.509 certificate, you can use that with Docker.

If the host does not have an X.509 certificate, you can create a self-signed, private certificate for testing purposes. For information on creating a self-signed certificate and private key, see Oracle Linux: Managing Certificates and Public Key Infrastructure.

If you want to disable X.509 certificate validation for testing purposes, see Setting Container Registry Options.

To use the X.509 Certificate with Docker:

1. If the host's X.509 certificate was issued by an intermediate Certificate Authority (CA), you must combine the host's certificate with the intermediate CA's certificate to create a chained certificate so that Docker can verify the host's X.509 certificate. For example:

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

2. Create the `/var/lib/registry/conf.d` directory, into which you need to copy the certificate and private key.

```
sudo mkdir -p /var/lib/registry/conf.d
```

3. Copy the certificate and private key to the `/var/lib/registry/conf.d` directory.

```
sudo cp certfile /var/lib/registry/conf.d/domain.crt
sudo cp keyfile /var/lib/registry/conf.d/domain.key
```

   where *certfile* is the full path to the host's X.509 certificate, and *keyfile* is the full path to the host's private key. For example:

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

4. Make sure the file permissions are correct for the private key:

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

Creating the Registry

This section discusses creating the registry server as a Docker container application. Perform these steps on the registry host.

Create the Docker registry container. For example:

```
docker run -d -p 5000:5000 --name registry --restart=always \
    -v /var/lib/registry:/registry_data \
    -e REGISTRY_STORAGE_FILESYSTEM_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 \
    -e REGISTRY_AUTH="" \
    container-registry.oracle.com/os/registry:latest
```

The registry image is pulled from the Oracle Container Registry and the Docker registry container is started.

The option not processed within `--restart=always` option starts the registry container when Docker is started.

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.

If you do not have an Oracle Account and if you do not require support, you can alternately use the publicly available Docker registry image at `library/registry:latest`.

Setting Up the Registry Port

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.

If you are running a firewall, make sure the TCP port that you want the Docker registry to listen on is accessible. If you are running `firewalld`, add the default rule for the `docker-registry` service:

```
sudo firewall-cmd --zone=public --permanent --add-service=docker-
registry
```

If you do not run the registry on the default port you can specify the port directly:

```
sudo firewall-cmd --zone=public --permanent --add-port=5001/tcp
```

Distributing X.509 Certificates

If the registry host uses a self-signed X.509 certificate, you must distribute the certificate to **all** hosts in your deployment that you intend to use the local Docker registry.

Perform the following steps **on each host** that needs to access the local registry. Substitute *registry_hostname* with the name of the registry host, and *port* with the port number you selected for your Docker registry server (5000 by default).

To distribute a self signed X.509 certificate:

1. Create the `/etc/docker/certs.d/`*`registry_hostname`*`:`*`port`* directory.

   ```
   sudo mkdir -p /etc/docker/certs.d/registry_hostname:port
   ```

2. Copy the X.509 certificate from the registry host using:

```
scp root@registry_hostname:/var/lib/registry/conf.d/domain.crt \
/etc/docker/certs.d/registry_hostname:port/ca.crt
```

3. Restart the `docker` service.

```
sudo systemctl restart docker.service
```

Importing Images Into a Registry

When you have set up a Docker registry server, you can import images into the registry so that they can be used to deploy containers. You may either pull images from a 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 images into a local Docker registry:

1. Pull an image from a registry. For example, 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 Creating a Docker Image From an Existing Container and Creating a Docker Image From a Dockerfile for information on how you can create your own images. When you have committed a customized image, you can tag it and push it to your local registry as indicated in the steps above.

7

Security Recommendations

Ensure that your infrastructure and containerized applications remain secure by following security recommendations and guidelines. Oracle recommends that in addition to the information provided here, you review upstream security guidelines such as those provided at https://docs.docker.com/engine/security/.

Best Practices for Docker Components

It is important to follow security guidelines at all levels within the infrastructure for an environment to best mitigate against exploitation. You should follow the best practice guidelines for each component at play within the environment.

Note that while containerization provides resource separation between applications running on the same host, the separation is not complete and it is possible to break out of a container or to exploit a container in such a way that it could affect other containers running on the same host. If you have different tenancies within your organization or if you have different customers that are using the same infrastructure, it is imperative that their containers run on different hosts or on different virtual machines to achieve more complete separation and to prevent the likelihood of a serious data breach.

Host

- **Regularly update the host kernel and operating system software**

  Oracle regularly releases security patches and bug fixes for the kernel and operating system software as issues are resolved. It is highly recommended that you keep the operating system current with the most recent software updates. Subscribe the system to the latest software channels or repositories; run regular yum update operations; and consider using Ksplice to keep your system software up to date.

- **Use a minimal operating system and ensure that it is following security best practices**

  Where possible, use a minimal operating system installation and ensure that it follows the security best practices described in Oracle Linux 7: Security Guide. Most importantly, you should reduce the number of services running on the same system. Ideally, move all other services to reside within containers controlled by Docker or move them to other systems entirely. This helps to contain damage in the event of container breakout.

- **Regularly scrutinize the operating system and kernel for safety**

  Be vigilant that the operating system is regularly scrutinized for safety and potential vulnerabilities.

- **Use a mature kernel that provides the best possible security feature set**

  Oracle requires that you use UEK R5 or later with Oracle Container Runtime for Docker. This helps to ensure that kernel-based features such as kernel namespaces, private networking and control groups are mature, reliable and heavily tested. Since the kernel capabilities that are required for Docker are equally required to support other tools and

features such as LXC (Linux Containers), the required kernel capabilities are all tested regularly within the supported UEK releases.

Docker Engine

- **Restrict who can create and control containers using the Docker engine**

  The Docker engine is typically run with root privileges on the host system. You must, therefore, restrict who can create a new container on a system, or who has access to the Docker engine to start a container. Some operations, such as loading or pulling images could be potentially exploited through other inputs. Make sure that the users that have access to perform actions using the Docker engine are all trusted users.

- **Regularly update the Docker engine software**

  As with all software on the operating system, regular updates are critical to ensure that you are running the latest security patched binaries.

- **Use current recommended drivers**

  Do not use the legacy LXC driver. Oracle strongly recommends using btrfs-based storage in production environments. OverlayFS is acceptable for testing purposes, but developers are cautioned that there are known issues due to its lack of maturity. The default device-mapper option is not recommended as it is very slow and prone to running out of space inside the container.

- **Minimize Docker engine client ports**

  Run the Docker Daemon with only the single default Unix socket port if possible. If other sockets must be used, configure them for TLS.

- **Ensure sufficient storage capacity for Docker containers and images**

  Docker stores container and image data in a directory (`/var/lib/docker` by default). This space may fill up fast, so to prevent denial-of-service of Docker, the containers, and the Docker host, ensure sufficient storage capacity.

  Create a separate partition (logical volume) for storing Docker files.

- **Protect Docker service and configuration files**

  Set appropriate permissions and ownership for service and configuration files to prevent unauthorized access. The default values are usually appropriate, but the following guidelines may help when performing and audit.

  Ensure that the following Docker engine system files have secure permissions set (owner/group is `root:root` and permissions are 644 or more restrictive):

  - `/usr/lib/systemd/system/docker.service`
  - `/usr/lib/systemd/system/docker-registry.service`
  - `/usr/lib/systemd/system/docker.socket`
  - `/etc/sysconfig/docker`
  - `/etc/default/docker`
  - `/etc/sysconfig/docker-network`
  - `/etc/sysconfig/docker-registry`
  - `/etc/sysconfig/docker-storage`

Ensure that the following Docker engine system files have secure permissions set (owner/group is `root:root` and permissions are 755 or more restrictive):

- `/etc/docker`

Ensure that the following Docker engine system files have secure permissions set (owner/group is `root:root` and permissions are 444 or more restrictive):

- `/etc/docker/certs.d/<registry-name>/*`
- TLS CA certificate file (the file given with the `--tlscacert` parameter)
- Docker server certificate file (the file given with the `--tlscert` parameter)

Ensure that the following Docker engine system files have secure permissions set (owner/group is `root:root` and permissions are 400 or more restrictive):

- Docker server certificate key file (the file given with the `--tlskey` parameter)

Ensure that the following Docker engine system files have secure permissions set (owner/group is `root:docker` and permissions are 460 or more restrictive):

- `/var/run/docker.sock`

- **Monitor and audit Docker system files**

  Audit all Docker engine activities using a system logging facility like auditd. The service logging level should be "info" (default). Ensure enough storage space is available for audit logs to grow. Monitor the following files and directories:

  - `/var/lib/docker`
  - `/etc/docker`
  - `/usr/lib/systemd/system/docker-registry.service`
  - `/usr/lib/systemd/system/docker.service`
  - `/var/run/docker.sock`
  - `/etc/sysconfig/docker`
  - `/etc/sysconfig/docker-network`
  - `/etc/sysconfig/docker-registry`
  - `/etc/sysconfig/docker-storage`
  - `/etc/default/docker`

Docker Images

- **Ensure that images come from verified and trusted sources**

  Verify that Docker images are received and deployed unchanged from a source with a trusted reputation and which has been authenticated.

  When pulling images from remote sources, ensure that the connection is protected and that you are using HTTPS for the pull request. Do not use insecure image registries that are not protected by TLS.

  Ideally, you should pull images by pre-verified hash rather than by tag and if possble, export these images and host them on more secure media servers under your own control.

Where possible, Docker images should come from and be based on a curated, trusted collection of image suppliers.

- **Create reliably reproducible images**

  When using Dockerfiles to build new images, review base images and installed software for security. To help ensure that new images use base images and software that you have properly reviewed for security vulnerabilities:

  - Specify a fixed version in the base image in an image Dockerfile.
  - Specify fixed versions in package pulls in the build steps of an image Dockerfile (note that dependencies of dependencies can still be a reliability problem).
  - Ensure the that package pulls in the build steps are using trusted and verified sources

- **Minimize packages installed on images**

  Do not install unnecessary packages into new image builds. Review Dockerfiles to remove unnecessary installation steps so that images remain limited to their function.

- **Use Linux security modules**

  Within your images, use the appropriate security modules where possible:

  - Run SELinux on Red Hat distributions
  - Run AppArmor on Debian and Ubuntu distributions

- **Regularly update images**

  Containers must be regularly scanned to detect out of date or unpatched software. Since containers must be immutable, you cannot patch the software, you must instead replace it with a newly built image. Consider rolling your own new patched image if you are relying on a third party image to be updated and you can't wait.

  Vendors tend to focus on the current stream of development. Instead of patching your containers and images, rebuild the images from scratch and instantiate new containers from the newer builds.

Docker Containers

- **Run containers as a non-root user**

  Unless otherwise specified, Docker runs each container as root. Since the UIDs are shared across the host, the root user in a container is the root user on the host. When possible ensure that containers are started as a non-root user by using the `--user` flag. You can start a container to run as the current user by taking advantage of the `id` command. For example, use `--user $(id -u):$(id -g)` when starting a container.

- **Limit container memory and CPU usage**

  Create and launch containers with limited container memory and CPU boundaries using the `-m` and `--memory-swap` options for memory and swap memory; and the `-c` option for CPU.

- **Limit container restarts**

To prevent potential denial-of-service resulting from a container that spins out of control, limit container restarts using the `--restart=on-failure:N` option when creating or launching a container.

- **Monitor container resource usage**

  Docker provides facilities to monitor container resource usage, such as memory consumption, CPU time, I/O and network usage. Review container resource usage for performance, error detection and anomalous behavior. Consider using tools to monitor real-time resource usage for anomalous activity such as utilization of resources, suspicious traffic and unexpected user activity.

- **Limit container file access**

  When creating and launching containers, limit container file access using the `--read-only` flag or the `-v <host dir>:<container dir>:ro` option. Explicitly create volume(s) for container applications to write in and monitor changes to files in these volumes. Ensure that volumes that are dedicated for container write access are reviewed for sprawl and are cleaned up regularly.

  Do not mount sensitive host system directories at container runtime:

  - `/`
  - `/boot`
  - `/dev`
  - `/etc`
  - `/lib`
  - `/proc`
  - `/sys`
  - `/usr`

- **Regularly review containers for safety**

  Consider using tools that help to automate container safety checks and to monitor for changes within containers. For example, Docker Bench for Security (CIS) and Docker Diff can be helpful for this purpose. See https://github.com/docker/docker-bench-security and https://docs.docker.com/engine/reference/commandline/diff/ for more information.

  Systematically remove images and containers that are not needed from the host system to avoid image and container sprawl and to help prevent the accidental usage of an old, unused image or container that has potentially avoided security scrutiny.

- **Limit kernel capabilities in containers**

  When creating and launching containers, limit kernel capabilities using the `--cap-add` and `--cap-drop` options. Note that you can set the value for either of these options to `all`. Try to apply the minimal set of kernel capabilities required by the containerized application.

  By default, the following kernel capabilities are *granted* to a container:

  - `CHOWN`
  - `DAC_OVERRIDE`
  - `FSETID`
  - `FOWNER`

  - MKNOD
  - NET_RAW
  - SETGID
  - SETUID
  - SETFCAP
  - SETPCAP
  - NET_BIND_SERVICE
  - SYS_CHROOT
  - KILL
  - AUDIT_WRITE.

  By default, the following notable kernel capabilities are *removed* from a container:

  - SYS_TIME
  - NET_ADMIN
  - SYS_MODULE
  - SYS_NICE
  - SYS_ADMIN

  Do not use the `--privileged` option when starting containers.

- **Limit kernel file handle and process resources in containers**

  When creating and launching containers, limit kernel resources by using the `--ulimit` option or set container defaults using the `--default-ulimit` when starting the Docker service.

- **Limit container networking**

  Limit container networking completely using the `--icc=false` option when starting the Docker engine if you do not need your containers to communicate at all. By disabling inter-container communication, no network traffic is allowed between containers, but they are still able to publish ports on the host.

  When publishing ports to the host, specify the IP address of the interface that you wish the port to bind to so that the attack surface is reduced to the network interface where the container should be listening. Docker publishes to all interfaces (0.0.0.0) by default if an IP address is not specified when using the `-p` or `--publish` option.

  Do not run SSH inside of containers.

  Do not map privileged ports (< 1024) inside of containers.

  Do not use the `--net=host` mode option for containers when they are started or run.

- **Do not share host namespaces with your containers**

  Do not share host namespaces such as the PID or IPC namespaces when starting or running containers.

- **Do not expose host devices into containers**

  Do not expose host devices into containers when you start or run them.

Containerized Applications

- **Minimize kernel calls in containerized applications**

  Since the kernel is shared between containers, kernel calls increase risk to other containers running on the host system. Avoid kernel calls within containerized appliactions wherever possible.

- **Run Container applications as a non-root user**

  Unless otherwise specified, Docker runs each container as root. Ensure that containerized applications run as a non-root user. Since the UIDs are shared across the host, the root user in a container is the root user on the host.

  If you ever need to change the user, consider using gosu instead of sudo because gosu creates a single process, instead of the two processes that sudo creates. This can avoid issues where container signals are forwarded. See https://github.com/tianon/gosu for more information.

- **Remove or minimize the use of setuid and setgid in containerized applications**

  Most applications don't need any setuid or setgid binaries. If you can, disable or remove such binaries. By doing so, you remove the chance of them being used for privilege escalation attacks. If you discover binaries that have setuid or setgid permission flags, remove them altogether or try to remove the permission flags to remove the risks that are associated with these permissions on a binary.

- **Design containerized applications to be impermanent**

  As much as is possible, design applications to be stateless, rollable, instantly migrateable microservices container apps if possible. If using applications outside of your own design, take this approach into consideration when selecting software that you intend to run within your containers. This quality can be helpful in maintaining service during and in the time following a breach or accident in the system.

Additional Deployment and Development Tools

- **Avoid deploying or using development tools in production environments**

  There are many development tools available that can aid in the use of Docker, including boot2docker, Kitematic, VMware Fusion, and Vagrant. Avoid deploying these to production environments to reduce the attack surface.

  Development tools are often less security hardened so avoid employing them in production environments.

8
Known Issues

The following sections describe known issues in the current release of Oracle Container Runtime for Docker.

WARNING: bridge-nf-call-iptables Is Disabled

Warning messages may be displayed by Docker Engine when a user performs some actions, such as running `docker info` if the system kernel on a host system is configured to disable the `net.bridge.bridge-nf-call-iptables` and `net.bridge.bridge-nf-call-ip6tables` options. For example, the user may see an error similar to:

```
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
```

This is expected behavior. These settings control whether packets traversing a network bridge are processed by iptables rules on the host system. Typically, enabling these options is not desirable as this can cause guest container traffic to be blocked by iptables rules that are intended for the host. This could cause unpredictable behavior for containers that do not expect traffic to be firewalled at the host level.

If you accept and understand the implications of enabling these options or you have no iptables rules set on the host, you can enable these options to remove the warning messages. To temporarily enable these options:

```
sysctl net.bridge.bridge-nf-call-iptables=1
sysctl net.bridge.bridge-nf-call-ip6tables=1
```

To make these options permanent, edit `/etc/sysctl.conf` and add the lines:

```
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

Starting the Docker Engine With User Namespace Remapping Set To Default Can Fail

Starting the Docker Engine with User Namespace Remapping set to default can fail with an error during the creation of the `dockremap` user. For example:

```
dockerd --userns-remap default

Error during "dockremap" user creation: Couldn't create subordinate ID
ranges: Unable to add subuid range to user: "dockremap"; output: usermod:
invalid option -- 'v'
Usage: usermod [options] LOGIN
```

Creating a manual map file is unaffected by this issue.

Issue Pulling aarch64 Images From Oracle Container Registry

There is an issue pulling images for the Arm (aarch64) platform from Oracle Container Registry. The issue is under investigation.

Images for aarch64 are available on Docker Hub and work as expected.

9

Oracle Linux Container Image Tagging Conventions

Oracle follows several conventions when tagging container images for Oracle Linux. Users should be aware of these conventions to ensure that the best image is used for the purpose at hand to avoid unnecessary breakages in functionality and to help ensure that images continue to use the most recently patched software.

The slim Tag

Oracle releases minimal compressed versions of each Oracle Linux release. These images contain just enough operating system to run within a container and to perform installations of additional packages. These images are the recommended images for general use within builds and where scripted installation is likely to be used. The images that use this tag are maintained at the most current update level.

For example, to use the most recent version of an Oracle Linux 7 slim image, use the `7-slim` tag. To use the most recent version of an Oracle Linux 8 slim image, use the `8-slim` tag.

```
docker pull oraclelinux:7-slim
```

FIPS compliant versions of images are tagged with the `slim-fips` tag. These images include compliant cryptographic package versions and most of the initial image setup required for container FIPS compliance. To use these images, you must enable FIPS mode on the host system.

The following `slim-fips` images are available:

- `oraclelinux:7-slim-fips`:
  - The latest FIPS compliant versions Oracle Linux 7 cryptographic packages at the time of the release of the image are already installed;
  - The Oracle Linux 7.8 security validation repository is already enabled in the image yum configuration file, so that the container can retrieve system updates that include FIPS compliant cryptographic package versions;
  - The `dracut-fips` package required for container FIPS mode is already installed.
- `oraclelinux:8-slim-fips`:
  - The latest FIPS compliant versions Oracle Linux 8 cryptographic packages at the time of the release of the image are already installed;
  - The Oracle Linux 8.4 security validation repository is already enabled in the image yum configuration file, so that the container can retrieve system updates that include FIPS compliant cryptographic package versions;
  - The `/etc/system-fips` file required for container FIPS mode in docker is already created.

- Note that Oracle Linux 8 docker containers still require that you mount FIPS cryptographic policies from `/usr/share/crypto-policies/back-ends/FIPS` to `/etc/crypto-policies/back-ends` in the container. See Enabling FIPS Mode in Containers.

General Oracle Linux release Tags

Oracle Linux images are tagged at their release level and are maintained to always map to the latest corresponding update level. If you need a more complete operating system than the version provided in a slim image, you should use a release tag to obtain the latest image for that Oracle Linux image.

For example, to get the latest update release image for Oracle Linux 8, use the `8` tag:

```
docker pull oraclelinux:8
```

Oracle Linux Update Level Tags

Oracle Linux images are tagged at their update level. The other tags described map onto the latest or most current update level for an Oracle Linux image.

> **NOT_SUPPORTED:**
>
> Do not directly use update level tags within your Dockerfile or within any of your builds unless you have a specific use case that requires a particular update level. Typical use cases involve trying to resolve an issue or bug that is only present at a particular update level of Oracle Linux.
>
> Using an update level tag can result in your containers running unpatched software that may expose you to security issues and software bugs.

Update level tags use dot notation to indicate the update level. For example, Oracle Linux 8.2 is indicated using the `8.2` tag:

```
docker pull oraclelinux:8.2
```

The latest Tag

> **Important:**
>
> Oracle does not provide this tag for Oracle Linux images. Use a slim image or a release tag instead. Oracle also recommends that users avoid dependency on this tag when working with other distribution or software images.

The use of a default often results in significant confusion and regularly breaks builds and scripted functionality for end users. For this reason, and to help encourage best

practice when working with image tags, Oracle does not provide a `latest` tag for Oracle Linux images.

The following reasons for Oracle's decision on this help to explain why this tag is not available:

- When the `latest` tag is used, it can result in significant jumps between distribution releases rather than simple update levels. This is usually not what a user intends when selecting the `latest` tag, or depending on tools to fall back to this tag by not specifying a tag at all. Expected functionality can change dramatically between releases resulting in changes to commands, options, configurations and available software.
- There is no easy way to identify which `latest` image was used for a particular build, making it difficult to see the differences between two final build images. This problem tracking changes also makes it difficult to roll back to a known functioning base image if a new build fails.
- Tagging an image with the `latest` tag is not automatic and it is possible for a more recent image to be available while the image tagged as `latest` has not been updated. This can lead to unexpected consequences.
- There is no guarantee that all tools treat the `latest` tag the same. While some tools may default to always pulling an image tagged as `latest` from an upstream registry, other tools may default to a locally stored image also tagged as `latest`, even if it has fallen out of date.

This decision may result in errors in some tools that fall back to the `latest` tag when no tag is specified for an image. For example:

```
docker pull docker.io/library/oraclelinux

Trying to pull docker.io/library/oraclelinux...
  manifest unknown: manifest unknown
Error: error pulling image "docker.io/library/oraclelinux": unable to pull docker.io/
library/oraclelinux:
unable to pull image: Error initializing source docker://oraclelinux:latest: Error
reading manifest latest
in docker.io/library/oraclelinux: manifest unknown: manifest unknown
```

Always specify the appropriate tag for the image that you intend to use! For example:

```
docker pull oraclelinux:8
```