SUSE Linux Enterprise Server 12 SP5

Docker Open Source Engine Guide

Docker Open Source Engine Guide

SUSE Linux Enterprise Server 12 SP5

This guide introduces Docker Open Source Engine, a lightweight virtualization solution to run virtual units simultaneously on a single control host.

Publication Date: November 07, 2024

https://documentation.suse.com

Contents

1 Docker Open Source Engine Overview

Docker Open Source Engine is a lightweight virtualization solution to run multiple virtual units (containers) simultaneously on a single control host. Containers are isolated with Kernel Control Groups (*Control groups*) and *Namespace* .

Full virtualization solutions such as Xen, KVM, or `libvirt` are based on the processor simulating a complete hardware environment and controlling the virtual machines. However, Docker Open Source Engine only provides operating system-level virtualization where the Linux kernel controls isolated containers.

Before going into detail about Docker Open Source Engine, let's define some of the terms used:

Docker Open Source Engine
: Docker Open Source Engine is a server-client type application that performs all tasks related to virtual machines. Docker Open Source Engine comprises the following:

  - daemon - is the server side of Docker Open Source Engine that manages all Docker objects (images, containers, network used by containers, etc.)
  - REST API - applications can use this API to communicate directly with the daemon
  - a CLI client - that enables you to communicate with the daemon. If the daemon is running on a different machine than the CLI client, the CLI client can communicate by using network sockets or the REST API provided by Docker Open Source Engine.

Image
: An *image* is a read-only template used to create a *virtual machine* on the host server. A Docker image is made by a series of layers built one over the other. Each layer corresponds to a permanent change, for example an update of an application. The changes are stored in a file called a `Dockerfile`. For more details see the official Docker documentation (http://docs.docker.com/engine/reference/glossary#image).

Dockerfile
: A `Dockerfile` stores changes made on top of the base image. The *Docker Open Source Engine* reads instructions in the *Dockerfile* and builds a new image according to the instructions.

Container
: A *container* is a running instance based on a particular Docker *Image*. Each *container* can be distinguished by a unique container ID.

Registry

A *registry* is storage for already created images. It typically contains several *repositories* There are two types of registry:

- public registry - where everyone (usually registered) can download and use images. A typical public registry is Docker Hub (https://hub.docker.com/).
- private registry - these are accessible for particular users or from a particular private network.

Repository

A *repository* is storage in a *registry* that stores a different version of a particular image. You can pull or push images from or to a repository.

Control groups

Control groups, also called `cgroups`, is a Linux kernel feature that allows aggregating or partitioning tasks (processes) and all their children into hierarchically organized groups to isolate resources.

Namespace

Docker Open Source Engine uses *namespaces* for its containers that isolates resources reserved for particular containers.

Orchestration

In a production environment you typically need a cluster with many containers on each cluster node. The containers must cooperate and you need a framework that enables you to manage the containers automatically. The act of automatic container management is called container orchestration and is typically handled by Kubernetes.

Docker Open Source Engine is a platform that allows developers and system administrators to manage the complete life cycle of images. Docker Open Source Engine makes it easy to build, ship and run images containing applications.

Docker Open Source Engine provides you with the following advantages:

- Isolation of applications and operating systems through containers.
- Near native performance, as Docker Open Source Engine manages allocation of resources in real time.
- Controls network interfaces and resources available inside containers through cgroups.
- Versioning of images.

- Allows building new images based on existing ones.
- Provides you with container orchestration.

On the other hand, Docker Open Source Engine has the following limitations:

LIMITATIONS OF DOCKER OPEN SOURCE ENGINE

- Containers run inside the host system's kernel and cannot use a different kernel.
- Only allows Linux *guest* operating systems.
- Docker Open Source Engine is not a full virtualization stack like Xen, KVM, or `libvirt`.
- Security depends on the host system. Refer to the official security documentation (http://docs.docker.com/articles/security/) for more details.

1.1 Docker Open Source Engine Architecture

Docker Open Source Engine uses a client/server architecture. You can use the *CLI client* to communicate with the *daemon*. The *daemon* then performs operations with containers and manages images locally or in registry. The *CLI client* can run on the same server as the host daemon or on a different machine. The *CLI client* communicates with the *daemon* by using network sockets. The architecture is depicted in *Figure 1.1, "The Docker Open Source Engine architecture"*.

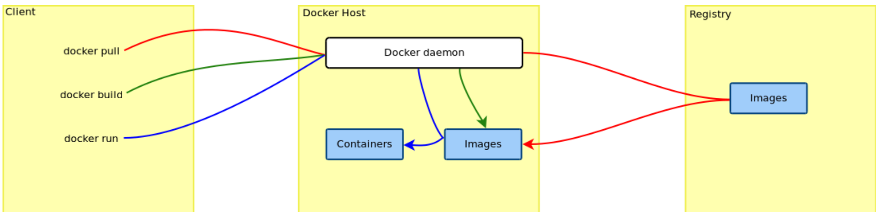

FIGURE 1.1: THE DOCKER OPEN SOURCE ENGINE ARCHITECTURE

1.2 Docker Open Source Engine Drivers

1.2.1 Container Drivers

Docker Open Source Engine uses libcontainer (https://github.com/docker/libcontainer) as the back-end driver to handle containers.

1.2.2 Storage Drivers

Docker Open Source Engine supports different storage drivers:

- `vfs`: this driver is automatically used when the Docker host file system does not support copy-on-write. This is a simple driver which does not offer some advantages of Docker Open Source Engine (like sharing layers, more on that in the next sections). It is highly reliable but also slow.
- `devicemapper`: this driver relies on the device-mapper thin provisioning module. It supports copy-on-write, hence it offers all the advantages of Docker Open Source Engine.
- `btrfs`: this driver relies on Btrfs to provide all the features required by Docker Open Source Engine. To use this driver the `/var/lib/docker` directory must be on a Btrfs file system.
- `AUFS`: this driver relies on the AUFS union file system. Neither the upstream kernel nor the SUSE one supports this file system. Hence the AUFS driver is not built into the SUSE `docker` package.

SLE 12 uses the Btrfs file system by default, which leads Docker Open Source Engine to use the `btrfs` driver.

It is possible to specify which driver to use by changing the value of the `DOCKER_OPTS` variable defined inside of the `/etc/sysconfig/docker` file. This can be done either manually or using YaST by browsing to *System* › */etc/sysconfig Editor* › *System* › *Management* › *DOCKER_OPTS* menu and entering the `-s storage_driver` string.

For example, to force the usage of the `devicemapper` driver enter the following text:

```
DOCKER_OPTS="-s devicemapper"
```

Important: Mounting `/var/lib/docker`

It is recommended to have `/var/lib/docker` mounted on a separate partition or volume to not affect the Docker Open Source Engine host operating system in case of a file system corruption.

In case you choose the Btrfs file system for `/var/lib/docker`, it is strongly recommended to create a subvolume for it. This ensures that the directory is excluded from file system snapshots. If not excluding `/var/lib/docker` from snapshots, the file system will likely run out of disk space soon after you start deploying containers. What's more, a rollback to a previous snapshot will also reset the Docker Open Source Engine database and images. Refer to *Creating and Mounting New Subvolumes* in *Book “Administration Guide”, Chapter 7 “System Recovery and Snapshot Management with Snapper”, Section 7.1 “Default Setup”* for details.

2 Docker Open Source Engine Installation

2.1 General Preparation

Prepare the host as described below. Before installing any Docker-related packages, you need to enable the container module:

Note: Built-in Docker Orchestration Support

Starting with Docker Open Source Engine 1.12, the container orchestration is now an integral part of the Docker Open Source Engine. Even though this feature is available in SLESSP1 and in SLESSP2, it is not supported and is only a technical preview. Use Kubernetes for Docker container orchestration, for details refer to the Kubernetes documentation (http://kubernetes.io/docs/getting-started-guides/kubeadm/).

PROCEDURE 2.1: ENABLING THE CONTAINER MODULE USING YAST

1. Start YaST, and select *Software* › *Software Repositories*.
2. Click *Add* to open the add-on dialog.
3. Select *Extensions and Modules from Registration Server* and click *Next*.
4. From the list of available extensions and modules, select *Container Module 12 x86_64* and click *Next*.
   The containers module and its repositories will be added to your system.
5. If you use Subscription Management Tool, update the list of repositories on the SMT server.

PROCEDURE 2.2: ENABLING THE CONTAINER MODULE USING SUSECONNECT

- The Container Module can be added also with the following command:

```
$ sudo SUSEConnect -p sle-module-containers/12/x86_64 -r ''
```

Note: Note about the SUSEConnect syntax

The `-r ''` flag is required to avoid a known limitation of SUSEConnect.

PROCEDURE 2.3: INSTALLING AND SETTING UP DOCKER OPEN SOURCE ENGINE

1. Install the `docker` package:

```
sudo zypper install docker
```

2. To automatically start the Docker service at boot time:

```
sudo systemctl enable docker.service
```

   This will automatically enable docker.socket in consequence.

3. In case you will use Portus and an SSL secured registry, open the `/etc/sysconfig/docker` file. Search for the parameter `DOCKER_OPTS` and add `--insecure-registry ADDRESS_OF_YOUR_REGISTRY`.

4. In the production environment when using the SSL secured registry with Portus, add CA certificates to the directory `/etc/docker/certs.d/<registry address>` and copy the CA certificates to your system:

```
    sudo cp CA /etc/pki/trust/anchors/ && update-ca-certificates
```

5. Start the Docker service:

```
sudo systemctl start docker.service
```

   This will automatically start docker.socket in consequence.

The Docker daemon listens on a local socket which is accessible only by the `root` user and by the members of the `docker` group. The `docker` group is automatically created at package installation time. To allow a certain user to connect to the local Docker daemon, use the following command:

```
sudo /usr/sbin/usermod -aG docker USERNAME
```

The user can communicate with the local Docker daemon upon his next login.

2.2 Networking

If you want your containers to be able to access the external network, you must enable the `ipv4 ip_forward` rule. This can be done using YaST by browsing to *System* › *Network Settings* › *Routing* menu and ensuring `Enable IPv4 Forwarding` is checked.

This option cannot be changed when networking is handled by the Network Manager. In such cases the `/etc/sysconfig/SuSEfirewall2` file needs to be edited manually to ensure the `FW_ROUTE` flag is set to `yes`:

```
FW_ROUTE="yes"
```

2.2.1 Networking Limitations on Power Architecture

Currently Docker networking has two limitations on the POWER architecture.

The first limitation is about iptables. SLE 12 machines cannot run Docker Open Source Engine with the iptables support enabled. An update of the kernel is going to solve this issue. In the meantime the `docker` package for POWER has iptables support disabled via a dedicated directive inside of `/etc/sysconfig/docker`.

As a result of this limitation Docker containers will not have access to the outer network. A possible workaround is to share the same network namespace between the host and the containers. This however reduces the isolation of the containers.

The network namespace of the host can be shared on a per-container basis by adding `--net=host` to the `docker run` command.

Note: iptables support on SLE 12 SP1

SLE 12 SP1 hosts are not affected by this limitation but, given they use the same SLE 12 package, they will have iptables support disabled. This can be changed by removing the `-iptables=false` setting inside of `/etc/sysconfig/docker`.

The second limitation is about network isolation between the containers and the host. Currently it is not possible to prevent containers from probing or accessing arbitrary ports of each other.

3 Installing `sle2docker`

The `sle2docker` is used to import pre-built SUSE Linux Enterprise images. The imported pre-built images can then be used to create base Docker images.

The tool is part of the official container module. You can install it by using `zypper`. But prior to installing `sle2docker`, verify that the following prerequisites are fulfilled:

- Ruby is installed on the host machine.
- The docker daemon is running on the system.
- The user invoking `sle2docker` must have proper rights to invoke Docker commands.

If the conditions above are fulfilled, you can install the `sle2docker` tool by running:

```
sudo zypper in sle2docker
```

4 Storing Images

Prior to creating your own images, you should decide where you will store the images. The easiest solution would be to push these images to the Docker Hub (https://hub.docker.com). By default all images pushed to the Docker Hub are public. This is probably fine as long as this does not violate your company's policy and your images do not contain sensitive data or proprietary software.

If you need to restrict access to your Docker images, there are two options:

- Get a subscription on Docker Hub that unlocks the feature to create private repositories.
- Run an on-site Docker Registry where to store all the Docker images used by your organization or company and combine them with Portus to secure the registry.

This chapter describes how to set up an on-site Docker Registry and how to combine it with Portus.

4.1 What is a Docker Registry?

The Docker Registry is an open source project created by Docker Inc. It allows the storage and retrieval of Docker images. By running a local instance of the Docker Registry it is possible to completely avoid usage of the Docker Hub.

The Docker Registry is also used by the Docker Hub. However, the Docker Hub, as seen from the user perspective, is made of the following parts at least:

- The user interface (UI): The part that is accessed by users with their browser. The UI provides a nice and intuitive way to browse the contents of the Docker Hub either manually or by using a search feature. It also allows to create organizations made by different users.
  This component is closed source.
- The authentication component: This is used to protect the images stored inside of the Docker Hub. It validates all push, pull and search requests.
  This component is closed source.
- The storage back-end: This is where the Docker images are sent and downloaded from. It is provided by the Docker Registry.
  This component is open source.

4.2 Installing and Setting Up Docker Registry

1. Install the `docker-distribution-registry` package:

```
sudo zypper install docker-distribution-registry
```

2. To automatically start the Docker Registry at boot time:

```
sudo systemctl enable registry
```

3. Start the Docker Registry:

```
sudo systemctl start registry
```

The Docker Registry configuration is defined inside of `/etc/registry/config.yml`.

With the default configuration the registry listens on ports `5000` and stores the Docker images under `/var/lib/docker-registry`.

Note: Incompatible Versions of Docker Open Source Engine and Docker Registry

Docker Registry version 2.3 is not compatible with Docker Registry versions older than 1.10, because v2 manifests were only introduced with Docker Open Source Engine 1.10. As Docker Open Source Engine and Docker Registry can be installed on different boxes, the versions might be incompatible. If you experience communication errors between between Docker Open Source Engine and Docker Registry, update both to the latest versions.

For more details about Docker Registry and its configuration, see the official documentation at: https://docs.docker.com/registry/ ↗.

4.3 Limitations

The Docker Registry has two major limitations:

- It lacks any form of authentication. That means everybody with access to the Docker Registry can push and pull images to it. That also includes the possibility to overwrite already existing images.
- There is no way to see which images have been pushed to the Docker Registry. You can manually take notes of what is being stored inside of it. There is also no search functionality, which makes collaboration harder.

The next section is going to introduce Portus, the solution to all of the problems above.

4.4 Portus

Portus is an authentication service and user interface for the Docker Registry. It is an open source project created by SUSE to address all the limitations faced by the local instances of Docker Registry. By combining Portus and Docker Registry, it is possible to have a secure and enterprise ready on-premise version of the Docker Hub.

Portus is available for SLES customers as a Docker image from SUSE Container Registry. For example, to pull the `2.4.0` tag, run the following command:

```
tux > docker pull registry.suse.com/sles12/portus:2.4.0
```

In addition to the official version of the Portus image from SUSE Container Registry, there is a community version that can be found on Docker Hub. However, as a customer of SLES, we strongly suggest you use the official Portus image instead. The Portus image for SLES customers has the same code as the one from the community. Therefore, the setup instructions from http://port.us.org/docs/deploy.html apply for both images.

Portus was previously also available as an RPM package. To migrate from the RPM package to the Docker image, follow the instructions at http://port.us.org/docs/migrate-from-rpm.html.

5 Creating Custom Images

For creating your custom image you need a base Docker image of SLES. You can use any of the pre-built SLES images that you can obtain as described in *Section 5.2, "Customizing SLES Docker Images"*.

Note: No SLES Images in Docker Hub

Usually you can pull a variety of base Docker images from the docker hub (https://hub.docker.com/) but that does not apply for SLES. Currently we cannot distribute SLES images for Docker Open Source Engine because there is no way to associate an End-User License Agreement (EULA) to a Docker image. `sle2docker` enables you to import pre-built SLES images that you can use for creating base SLES images.

After you obtain your base docker image, you can modify the image by using a `Dockerfile` (usually placed in the build directory). Then use the standard building tool to create your custom image:

```
docker build PATH_TO_BUILD_DIRECTORY
```

For more `docker build` options, refer to the official Docker documentation (https://docs.docker.com/engine/reference/commandline/build/).

Note: Dockerizing Your Applications

You may want to write a dockerfile for your own application that should be run inside a docker container. For a procedure refer to *Chapter 6, Creating Docker Images of Applications*.

5.1 Obtaining Base SLES Images

How to obtain a pre-built base image depends on the SUSE Linux Enterprise Server version:

Up to SLES 12 SP2

```
tux > sudo zypper in sles11sp4-docker-image
```

SLES 12 SP3 and higher

```
docker pull registry.suse.com/suse/sles12sp4
```

Pre-built images do not have repositories configured. But when the Docker host has an SLE subscription that provides access to the product used in the image, Zypper will automatically have access to the right repositories.

Important: Docker Image Packages are no Longer supported

Pre-built base images like `sles11sp4-docker-image` and `suse-sles12sp3-image` that ship with the SUSE Linux Enterprise12 Containers module do not receive updates. We recommend to use SUSE Linux Enterprise12SP3 and newer images that can be obtained through the `registry.suse.com` Docker registry.

If you have obtained the image with Zypper, you need to activate it. Proceed as follows:

PROCEDURE 5.1: ACTIVATING THE BASE IMAGE FOR UP TO SLES 12 SP2

1. Get the proper image name with `sle2docker` by running

```
sle2docker list
```

2. Activate the image by using the image name from the previous step:

```
sle2docker activate PRE-BUILT_IMAGE_NAME
```

3. Check if the image was successfully activated by running

```
sle2docker
```

You can customize the docker image as described in *Section 5.2, "Customizing SLES Docker Images"*.

The `latest` tag refers to the most recently built and published image for the newest Service Pack release, while builds for a specific Service Pack can be referenced by *`MAJOR.SP`*. To refer to a specific image build, the build identification numbers need to be appended, e.g. `15.0.3.2.1` or `15.1.2.3`. The full reference including the tag to a specific image is part of the meta information, see *Section 5.2.3, "Meta Information in SLE Container Images"*.

5.2 Customizing SLES Docker Images

The pre-built images do not have any repository configured and do not include any modules or extensions. They contain a zypper service (https://github.com/SUSE/container-suseconnect) that contacts either the SUSE Customer Center (SCC) or your Subscription Management Tool

(SMT) server, according to the configuration of the SLE host that runs the Docker container. The service obtains the list of repositories available for the product used by the Docker image. You can also directly declare extensions in your `Dockerfile` (for details refer to *Section 5.2.4, "Adding SLE Extensions and Modules to Images"*.

You do not need to add any credentials to the Docker image because the machine credentials are automatically injected into the container by the Docker daemon. They are injected inside of the `/run/secrets` directory. The same applies to the `/etc/SUSEConnect` file of the host system, which is automatically injected into the `/run/secrets` directory.

Note: Credentials and Security

The contents of the `/run/secrets` directory are never committed to a Docker image, hence there is no risk of your credentials leaking.

Note: Building Images on Systems Registered with RMT

When the host system used for building Docker images is registered against RMT, the default behavior allows only building containers of the same code base as the host. For example, if your Docker host is a SLE 15 system you can only build SLE 15-based images on that host by default. To build images for a different SLE version, for example SLE 12 on a SLE 15 host, the host machine credentials for the target release can be injected into the container as outlined below.

When the host system is registered again SUSE Customer Center this restriction does not apply.

Note: Building Container Images in On-Demand SLE Instances in the Public Cloud

When building container images on SLE instances that were launched as so-called "on-demand" or "pay as you go" instances on a Public Cloud (AWS, GCE, or Azure), some additional steps have to be performed. For installing packages and updates, the "on-demand" public cloud instances are connected to a public cloud-specific update infrastructure, which is based on RMT servers operated by SUSE on the various Public Cloud Providers. Some additional steps are required to locate the required services and authenticate with them.

A new service was introduced to enable this, called `containerbuild-regionsrv`. This service is available in the public cloud images provided through the Marketplaces of the various Public Cloud Providers. So before building an image, this service has to be started on the public cloud instance by running the following command:

```
tux > sudo systemctl start containerbuild-regionsrv
```

To start it automatically after system startup, enable it with **`systemctl`**:

```
tux > sudo systemctl enable containerbuild-regionsrv
```

The Zypper plugins provided by the SLE base images will then connect to this service for retrieving authentication details and information about which update server to talk to. In order for that to work the container has to be built with host networking enabled, like the following example:

```
tux > docker build --network host build-directory/
```

Since update infrastructure in the Public Clouds is based upon RMT, the same restrictions with regard to building SLE images for SLE versions differing from the SLE version of the host apply here as well (see *Note: Building Images on Systems Registered with RMT*).

To obtain the list of repositories use the following command:

```
zypper ref -s
```

It will automatically add all the repositories to your container. For each repository added to the system a new file will be created under **`/etc/zypp/repos.d`**. The URLs of these repositories include an access token that automatically expires after 12 hours. To renew the token call the **`zypper ref -s`** command. It is secure to commit these files to a Docker image.

If you want to use a different set of credentials, place a custom `/etc/zypp/credentials.d/SCCcredentials` file inside of the Docker image. It contains the machine credentials that have the subscription you want to use. The same applies to the `SUSEConnect` file: to override the file available on the host system that is running the Docker container, add a custom `/etc/SUSEConnect` file inside of the Docker image.

Now you can create a custom Docker image by using a `Dockerfile`. If you want to create a custom image, refer to `Dockerfile` as described in *Section 5.2.2*. In case you would like to move your application to a Docker container, refer to *Chapter 6, Creating Docker Images of Applications*. After you have edited the `Dockerfile`, build the image by running the following command in the same directory in which the `Dockerfile` resides:

- *Section 5.2.1, "Creating a Custom SLE 11 SP4 Image"* for SLE 11 SP4
- *Section 5.2.2, "Creating a Custom SLE 12 Image"* for SLE 12

In case you would like to move your application to a Docker container, refer to *Chapter 6, Creating Docker Images of Applications*.

5.2.1 Creating a Custom SLE 11 SP4 Image

The following `Dockerfile` creates a simple Docker image based on SLE 11 SP4:

```
FROM suse/sles11sp4:latest

RUN zypper ref -s
RUN zypper -n in vim
```

When the Docker host machine is registered against an internal SMT server, the Docker image requires the SSL certificate used by SMT:

```
FROM suse/sles11sp4:latest

# Import the crt file of our private SMT server
ADD http://smt.test.lan/smt.crt /etc/ssl/certs/smt.pem
RUN c_rehash /etc/ssl/certs

RUN zypper ref -s
RUN zypper -n in vim
```

5.2.2 Creating a Custom SLE 12 Image

The following `Dockerfile` creates a simple Docker image based on SLE 12 SP4:

```
FROM registry.suse.com/suse/sles12sp4:latest

RUN zypper ref -s
```

```
RUN zypper -n in vim
```

When the Docker host machine is registered against an internal SMT server, the Docker image requires the SSL certificate used by SMT:

```
FROM registry.suse.com/suse/sles12sp4:latest

# Import the crt file of our private SMT server
ADD http://smt.test.lan/smt.crt /etc/pki/trust/anchors/smt.crt
RUN update-ca-certificates

RUN zypper ref -s
RUN zypper -n in vim
```

5.2.3 Meta Information in SLE Container Images

Starting from SUSE Linux Enterprise 12 SP3, all base container images include information such as a build time stamp and description. This information is provided in the form of labels attached to the base images and is thus available for derived images and containers as well. It can be displayed with **`docker inspect`**:

```
tux > docker inspect registry.suse.com/suse/sle15
[...]
            "Labels": {
                "com.suse.sle.base.created": "2019-06-20T18:21:37.729383880Z",
                "com.suse.sle.base.description": "Image containing a minimal environment
 for containers based on SUSE Linux Enterprise Server 15 SP1.",
                "com.suse.sle.base.disturl": "obs://build.suse.de/SUSE:SLE-15-
SP1:Update:CR/images/20efed47827dc48da9537c1aeed4dbe2-sles15-image",
                "com.suse.sle.base.reference": "registry.suse.com/suse/
sle15:15.1.6.2.31",
                "com.suse.sle.base.title": "SUSE Linux Enterprise Server 15 SP1 Base
 Container",
                "com.suse.sle.base.url": "https://www.suse.com/products/server/",
                "com.suse.sle.base.vendor": "SUSE LLC",
                "com.suse.sle.base.version": "15.1.6.2.31",
                "org.openbuildservice.disturl": "obs://build.suse.de/SUSE:SLE-15-
SP1:Update:CR/images/20efed47827dc48da9537c1aeed4dbe2-sles15-image",
                "org.opencontainers.image.created": "2019-06-20T18:21:37.729383880Z",
                "org.opencontainers.image.description": "Image containing a minimal
 environment for containers based on SUSE Linux Enterprise Server 15 SP1.",
                "org.opencontainers.image.title": "SUSE Linux Enterprise Server 15 SP1
 Base Container",
                "org.opencontainers.image.url": "https://www.suse.com/products/server/",
```

```
                "org.opencontainers.image.vendor": "SUSE LLC",
                "org.opencontainers.image.version": "15.1.6.2.31",
                "org.opensuse.reference": "registry.suse.com/suse/sle15:15.1.6.2.31"
            }
[...]
```

All labels are shown twice. This is necessary to ensure that in derived images the information about the original base image is still visible and not overwritten.

5.2.4 Adding SLE Extensions and Modules to Images

You may have subscriptions to SLE extensions or modules that you would like to use in your custom image. To add them to the Docker image, proceed as follows:

PROCEDURE 5.2: ADDING EXTENSION AND MODULES

1. Add the following into your `Dockerfile`:

   ```
   ADD *.repo /etc/zypp/repos.d/
   ADD *.service /etc/zypp/services.d
   RUN zypper refs && zypper refresh
   ```

2. Copy all `.service` and `.repo` files that you will use into the directory where you will build the Docker image from the `Dockerfile`.

6 Creating Docker Images of Applications

Docker Open Source Engine is a technology that can help you to minimize resources used to run or build your applications. There are several types of applications that are suitable to run inside a Docker container like daemons, Web pages or applications that expose ports for communication. You can use Docker Open Source Engine to automate building and deployment processes by adding the build process into a Docker image, then building the image and then running containers based on that image.

Running your application inside a Docker container provides you with the following advantages:

- You can minimize the runtime environment of the application as you can add to the Docker image of the application just the required processes and applications.
- The image with your application is portable across machines also with different Linux host systems.
- You can share the image of your application by using a repository.
- You can use different versions of required packages in the container than the host system uses without having problems with dependencies.
- You can run several instances of the same application that are completely independent from each other.

Using Docker Open Source Engine for building of applications provides the following features:

- You can prepare a complete building image.
- Your build always runs in the same environment.
- Your developers can test their code in the same environment as used in production.
- You can set up an automated building process.

The following section provides you with examples and tips on how to create Docker images of your applications. Prior to reading further, make sure that you have activated your SLES base Docker image as described in *Section 5.1, "Obtaining Base SLES Images"*.

6.1 Running an Application with Specific Package Versions

You may face a problem that your application uses a specific version of a package that is different from the package installed on the system that should run your application. You can modify your application to work with another version or you may create a Docker image with that particular package version. The following example of a `Dockerfile` shows an image based on a current version of SLES but with an older version of the `example` package

```
FROM registry.suse.com/suse/sles12sp4:latest
MAINTAINER Tux

RUN zypper ref && zypper in -f example-1.0.0-0
COPY application.rpm /tmp/

RUN zypper --non-interactive in /tmp/application.rpm

ENTRYPOINT ["/etc/bin/application"]

CMD ["-i"]
```

Now you can build the image by running in the same directory as the `Dockerfile` resides:

```
docker build --tag tux_application:latest .
```

The `Dockerfile` example shown above performs the following operations during the **`docker build`**:

1. Updates the SLES repositories.
2. Installs the desired version of the `example` package.
3. Copies your application package to the image. The source RPM must be placed in the build context.
4. Unpacks your application.
5. The last two steps run your application after a container is started.

After a successful build of the `tux_application` image, you can start a container based on your new image:

```
docker run -it --name application_instance tux_application:latest
```

You have created a container that runs a single instance of your application. Bear in mind that after closing the application, the Docker container exits as well.

6.2 Running Applications with Specific Configuration

You may need to run an application that is delivered in a standard package accessible through SLES repositories but you may need to use a different configuration or use specific environment variables. In case you would like to run several instances of the application with non-standard configuration, you can create your own image that will pass the custom configuration to the application.

An example with the *example* application follows:

```
FROM registry.suse.com/suse/sles12sp4:latest

RUN zypper ref && zypper --non-interactive in example

ENV BACKUP=/backup

RUN mkdir -p $BACKUP
COPY configuration_example /etc/example/

ENTRYPOINT ["/etc/bin/example"]
```

The above example `Dockerfile` results in the following operations:

1. Refreshing of repositories and installation of the *example*.
2. Sets a `BACKUP` environment variable (the variable persists to containers started from the image). You can always overwrite the value of the variable with a new one while running the container by specifying a new value.
3. Creates the directory `/backup`.
4. Copies the `configuration_example` to the image.
5. Runs the *example* application.

Now you can build the image and after a successful build, you can run a container based on your image.

6.3 Sharing Data between an Application and the Host System

You may run an application that needs to share data between the application's container and the host file system. Docker Open Source Engine enables you to do data sharing by using volumes. You can declare a mount point directly in the `Dockerfile`. But you cannot specify a directory on the host system in the `Dockerfile` as the directory may not be accessible at the build time. You can find the mounted directory in the `/var/lib/docker/volumes/` directory on the host system.

Note: Discarding Changes to the Directory to Be Shared

After you declare a mount point by using the `VOLUME` instruction, all your changes performed (by using the `RUN` instruction) to the directory will be discarded. After the declaration, the volume is part of a temporary container that is then removed after a successful build. In case you need to e.g. change permissions, perform the change before you declare the directory as a mount point in the `Dockerfile`.

You can specify a particular mount point on the host system when running a container by using the `-v` option:

```
docker run -it --name testing -v /home/tux/data:/data sles12sp4:latest /bin/bash
```

Note

Using the `-v` option overwrites the `VOLUME` instruction if you specify the same mount point in the container.

Now let's create an example image with a Web server that will read Web content from the host's file system. The `Dockerfile` could look as follows:

```
FROM registry.suse.com/suse/sles12sp4:latest

RUN zypper ref && zypper --non-interactive in apache2

COPY apache2 /etc/sysconfig/

RUN chown -R admin /data
```

```
EXPOSE 80

VOLUME /data

ENTRYPOINT ["apache2ctl"]
```

The example above installs the Apache Web server to the image and copies all your configuration to the image. The `data` directory will be owned by the *admin* user and will be used as a mount point to store your web pages.

6.4 Applications Running in the Background

Your application may need to run in the background as a daemon or as an application exposing ports for communication. In that case a typical Docker Open Source Engine container may be run in background. An example `Dockerfile` for an Apache2 server exposing a port looks as follows:

EXAMPLE 6.1: `Dockerfile` FOR AN APACHE2 WEB SERVER

```
FROM registry.suse.com/suse/sles12sp4:latest ❶
MAINTAINER tux ❷
RUN zypper ref -s && zypper --non-interactive in apache2

RUN echo "The Web Server is running" > /srv/www/htdocs/test.html ❸
# COPY data/* /srv/www/htdocs/ ❹

EXPOSE 80 ❺

ENTRYPOINT ["/usr/sbin/httpd"]
CMD ["-D", "FOREGROUND"]
```

❶ Base image, taken from *Section 5.1, “Obtaining Base SLES Images”*.

❷ Optional maintainer of the image.

❸ The test line for debugging purposes; can be removed if everything works as expected.

❹ The copy instruction to copy your own data to the server's directory. Currently, this line is disabled due to the hash mark in the first column.

❺ The exposed port for the Apache Web server.

Note: Check for Running Apache2 Instances on Your Host

Make sure that you do not have any Apache2 server instances running on your host. The Docker container would not serve any data if you have a host Apache2 server running. Remove or stop any Apache2 servers on your host.

To use the container, proceed as follows:

PROCEDURE 6.1: TESTING THE APACHE2 WEB SERVER

1. Prepare for the build process:

   a. Make sure you have installed the SUSE Linux Enterprise images as described in *Section 5.1, "Obtaining Base SLES Images"*.

   b. Save the `Dockerfile` from *Example 6.1, "`Dockerfile` for an Apache2 Web Server"* to a directory `docker`.

   c. Create your HTML files inside `docker/data`. Anything you put in this directory is copied to the Docker image and as such exposed on your Web server.

2. Build the container. Set a tag for your image with the `-t` option (here `tux/apache2`, but you can use any name you want):

```
sudo docker build -t tux/apache2 .
```

   Docker Open Source Engine refreshes repositories and installs the Apache2 server as it is not installed by default in the SLES Docker image.

3. Run the image in "detached" mode:

```
docker run --detach --interactive --tty tux/apache2
```

   Docker Open Source Engine responds with the container ID, for example:

```
7bd674eb196d330d50f8a3cfc2bc61a243a4a535390767250b11a7886134ab93
```

4. Open a Web browser and enter in the text field `http://localhost:80/test.html`. You should see the output `The Web Server is running`.

With the last procedure, you have built an image which was used to test the build process and the Apache2 Web server. If everything was successful, use the following steps to serve your data through the Apache2 Web server:

PROCEDURE 6.2: CREATING A DOCKER CONTAINER WITH YOUR OWN DATA

1. Stop and remove the previous container with:

```
docker ps --latest
CONTAINER ID        IMAGE               COMMAND                  [...]
afee0124a0c7        tux/apache2         "/usr/sbin/httpd -..."   [...]
docker rm --force afee0124a0c7
```

2. Remove or disable the echo line ❸ in *Example 6.1, "`Dockerfile` for an Apache2 Web Server"* and remove the hash character in the first column of the copy line ❹.

3. Rebuild the image as described in *Step 2* of *Procedure 6.1*.

4. Run the image in "detached" mode:

```
docker run --detach --interactive --tty tux/apache2
```

Docker Open Source Engine responds with the container ID, for example:

```
e43fff4ae9832ecdb7677c058a73039d7610c32145a1d9b6ad0a4ed52b5c4dc7
```

You can now access your data at `http://localhost:80`. If you do not want to copy your data into the Docker container, share a specific directory on your host. Refer to https://docs.docker.com/storage/volumes/ for more information.

7 Working with Containers

After you have created your images, you can start your containers based on that image. You can run an instance of the image by using the **`docker run`** command. The Docker Open Source Engine then creates and starts the container. The command **`docker run`** takes several arguments:

- A container name - it is recommended to name your container.
- Specify a user to use in your container.
- Define a mount point.
- Specify a particular host name, etc.

The container typically exits if its main process finishes. For example, if your container starts a particular application, as soon as you quit the application, the container exits. You can start the container again by running:

```
docker start -ai <container name>
```

You may need to remove unused containers, you can achieve this by using:

```
docker rm <container name>
```

7.1 Linking Containers

Docker Open Source Engine enables you to link containers together which allows for communication between containers on the same host server. If you use the standard networking model, you can link containers by using the `--link` option when running containers:

First create a container to link to:

```
docker run -d --name sles sles12sp4 /bin/bash
```

Then create a container that will link to the *sles* container:

```
docker run --link sles:sles sles12sp4 /bin/bash
```

The container that links to *sles* has defined environment variables that enable connecting to the linked container.

8 Troubleshooting

8.1 Analyze container images with container-diff

In case a custom Docker Open Source Engine container image built on top of the SLE base container image is not working as expected, the `container-diff` tool can help you analyze the image and collect information relevant for troubleshooting.

`container-diff` makes it possible to analyze image changes by computing differences between images and presenting the diff in a human-readable and actionable format. The tool can find differences in system packages, language-level packages, and files in a container image.

`container-diff` can handle local container images (using the prefix `daemon://`), images in a remote registry (using the prefix `remote://`), and images saved as `.tar` archives. You can use `container-diff` to compute the diff between a local version of an image and a remote version.

To install `container-diff` , run the **`sudo zypper in container-diff`** command.

8.1.1 Basic container-diff commands

The command **`container-diff analyze`** *`IMAGE`* runs a standard analysis on a single image. By default, it returns a hash and size of the container image. For more information that can help you to identify and fix problems, use the specific analyzers. Use the `--type` parameter to specify the desired analyzer. Two of the most useful analyzers are `history` (returns a list of descriptions of how an image layer was created) and `file` (returns a list of file system contents, including names, paths, and sizes):

```
tux > sudo container-diff analyze --type=history daemon://
IMAGE
tux > sudo container-diff analyze --type=file daemon://
IMAGE
```

To view all available parameters and their brief descriptions, run the **`container-diff analyze --help`** command.

Using the **`container-diff diff`** command, you can compare two container images and examine differences between them. Similar to the **`container-diff analyze`** command, **`container-diff diff`** supports several parameters. The example command below compares two images and returns a list of descriptions of how IMAGE_2 was created from IMAGE_1.

```
tux > sudo container-diff diff daemon://
IMAGE_1
daemon://
IMAGE_2
--type=history
```

To view all available parameters and their brief descriptions, run the **`container-diff diff --help`** command.

A GNU licenses

This appendix contains the GNU Free Documentation License version 1.2.

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/copyleft/ ↗.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled “GNU
Free Documentation License”.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.