Container Guide

The following guide covers the SUSE Linux Enterprise Server container ecosystem. Since containers are a constantly evolving technology, the guide is regularly updated, expanded and improved to reflect the latest technological developments.

Publication Date: 06 Mar 2025

Contents

1 Introduction to Linux containers

Linux containers offer a lightweight virtualization method to run multiple isolated virtual environments simultaneously on a single host. This technology is based on the Linux kernel's namespaces for process isolation and kernel control groups (cgroups) for resource (CPU, memory, disk I/O, network, etc.) management.

Unlike Xen and KVM, where a full guest operating system is executed through a virtualization layer, Linux containers share and directly use the host OS kernel.

Advantages of using containers

- **Size**: Container images should only include the content needed to run an application, whereas a virtual machine includes an entire operating system,
- **Performance**: Containers provide near-native performance, as the kernel overhead is lower compared to virtualization and emulation.
- **Security**: Containers make it possible to isolate applications into self-contained units, separated from the rest of the host system.
- **Resources management**: It is possible to granularly control CPU, memory, disk I/O, network interfaces, etc. inside containers (via cgroups).
- **Flexibility**: Container images hold all necessary libraries, dependencies, and files needed to run an application, thus can be easily developed and deployed on multiple hosts.

Limitations of containers

- Containers share the host system's kernel, so the containers have to use the specific kernel version provided by the host.
- Only Linux-based applications can be containerized on a Linux host.

- A container encapsulates binaries for a specific architecture (AMD64/Intel 64 or AArch64 for instance). So a container made for AMD64/Intel 64 only runs on an AMD64/Intel 64 system host without the use of emulation.
- Containers in themselves are no more secure than executing binaries outside of a container, and the overall security of containers depends on the host system. While containerized applications can be secured through AppArmor or SELinux profiles, container security requires putting in place tools and policies that ensure security of the container infrastructure and applications.

1.1 Key concepts and introduction to Podman

Although Docker Open Source Engine is a popular choice for working with images and containers, Podman provides a drop-in replacement for Docker that offers several advantages. While *Section 14, “Podman overview”* provides more information on Podman, this chapter offers a quick introduction to key concepts and a basic procedure for creating a container image and using Podman to run a container.

The basic Podman workflow is as follows:

Running a container, either on a local machine or in a cloud service, normally involves the following steps:

1. Fetch a base image by pulling it from a registry to your local machine.
2. Create a `Dockerfile` and use it to build a custom image on top of the base image.
3. Use the created image to start one or more containers.

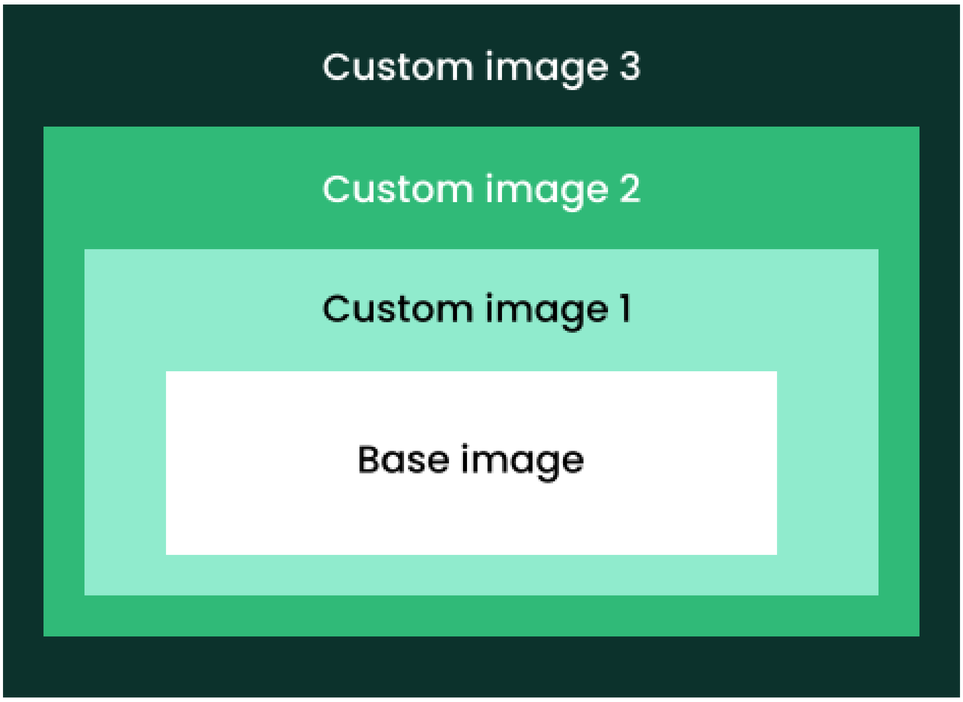
To run a container, you need an image. An image includes all dependencies needed to run the application. For example, the SLE BCI-Base image contains the SLE distribution with a minimal package selection.

While it is possible to create an image from scratch, few applications would work in such an empty environment. Thus, using an existing base image is more practical in most situations. A base image has no parent, meaning it is not based on another image.

Although you can use a base image for running containers, the main purpose of base images is to serve as foundations for creating custom images that can run containers with specific applications, servers, services, and so on.

Both base and custom images are available through a repository of images called a registry. Unless a registry is explicitly specified, Podman pulls images from the openSUSE (https://registry.opensuse.org) and Docker Hub (https://docker.io) registry. While you can fetch a base image manually, Podman can do that automatically when building a custom image.

To build a custom image, you must create a special file called a `Containerfile` or `Dockerfile` containing building instructions. For example, a `Dockerfile` can contain instructions to update the system software, install the desired application, open specific network ports, run commands, etc.



You can build images not only from base images, but also on top of custom images. So you can have an image consisting of multiple layers. Please refer to *Section 19, "Creating custom container images"* for more information.

2 Tools for building images and managing containers

All the tools described below are part of the **SUSE Linux Enterprise Server Containers Module**, except the Open Build Service (https://openbuildservice.org) ↗. You can see the full list of packages in the **Containers Module** in the SUSE Customer Center (https://scc.suse.com/packages) ↗.

2.1 SUSE Registry

https://registry.suse.com ↗ is the official source of SLE Base Container Images. It contains tested, updated and certified SLE Base Container Images. All images in the SUSE Registry are regularly rebuilt to include updates and fixes. The SUSE Registry's Web user interface lists a subset of the available images: Base Container Images, Development Stack Container Images, Application Container Images, SUSE Linux Enterprise Server Images, and Releases Out of General Support. The Web UI also provides additional information for each image, including release date, support level, size, digest of the images and packages inside the image.

2.2 Docker

Docker is a system for creating and managing containers. Its core is the Docker Open Source Engine— (https://github.com/moby/moby) ↗ a lightweight virtualization solution to run containers simultaneously on a single host. Docker containers can be built using Dockerfiles.

2.3 Podman

Podman (https://podman.io) ↗ stands for Pod Manager tool. It is a daemonless container engine for developing, managing and running Open Container Initiative (OCI) (https://opencontainers.org/about/overview/) ↗ containers on a Linux system, and it offers a drop-in alternative for Docker. Podman is the recommended container runtime for SLES. For a general introduction to Podman, refer to *Section 14, "Podman overview"*.

2.4 Buildah

Buildah (https://buildah.io) ↗ is a utility for building OCI container images. It is a complementary tool to Podman. In fact, the **`podman build`** command uses Buildah to perform container image builds. Buildah makes it possible to build images from scratch, from existing images, and using

Dockerfiles. OCI images built using the Buildah command-line tool and the underlying OCI-based technologies (for example, `containers/image` and `containers/storage`) are portable and can therefore run in a Docker Open Source Engine environment. For information on installing and using Buildah, refer to *Section 18, “Buildah overview”*.

2.5 skopeo

skopeo (https://github.com/containers/skopeo) ↗ is a command-line utility for managing, inspecting and signing container images and image repositories. skopeo can be used to inspect containers and repositories on remote and local container registries. skopeo can copy container images between different storage back-ends. skopeo is part of the `Basesystem Module` of SUSE Linux Enterprise Server.

2.6 Helm

Helm (https://helm.sh) ↗ is the Kubernetes package manager, and it is the de-facto standard for deploying containerized applications on Kubernetes clusters using charts. Helm can be used to install, update and remove containerized applications in Kubernetes environments. It can also handle the associated resources, such as configuration, storage volumes, etc. For instance, it is used for instance to deploy the RMT server (see RMT documentation (https://documentation.suse.com/sles/15-SP4/single-html/SLES-rmt/index.html#sec-rmt-deploy-kubernetes) ↗ for more information).

2.7 Distribution

Distribution (https://github.com/distribution/distribution) ↗ is an open-source registry implementation for storing and distributing container images using the OCI Distribution Specification. It provides a simple, secure and scalable base for building a large scale registry solution or running a simple private registry. Distribution can also mirror Docker Hub but not any other private registry (https://docs.docker.com/registry/recipes/mirror/#gotcha) ↗.

2.8 Open Build Service

The Open Build Service (OBS) (https://openbuildservice.org) provides free infrastructure for building and storing RPM packages including different container formats. The OBS Container Registry (https://registry.opensuse.org) provides a detailed listing of all container images built by the OBS, complete with commands for pulling the images into your local environment. The OBS openSUSE container image templates (https://build.opensuse.org/image_templates) can be modified to specific needs, which offers the easiest way to create your own container branch. Container images can be built with Docker tools from an existing image using a `Dockerfile`. Alternatively, images can be built from scratch using the KIWI NG image-building solution.

Instructions on how to build images on OBS can be found in the following blog post (https://openbuildservice.org/2018/05/09/container-building-and-distribution/).

SUSE Container Images, called SLE Base Container Images (SLE BCIs) are the only official container images. They are not available at https://build.opensuse.org, and the RPMs available on the OBS are not identical to the RPMs distributed as part of SUSE Linux Enterprise Server. This means that it is not possible to build supported images at https://build.opensuse.org.

For more information about SLE BCIs, refer to *Section 4, "General-purpose SLE Base Container Images"*.

2.9 KIWI NG

KIWI (https://github.com/OSInside/kiwi) Next Generation (KIWI NG) is a multi-purpose tool for building images. In addition to container images, regular installation ISO images, and images for virtual machines, KIWI NG can build images that boot via PXE or Vagrant boxes. The main building block in KIWI NG is an image XML description, a directory that includes the `config.xml` or `config.kiwi` file along with scripts or configuration data. The process of creating images with KIWI NG is fully automated and does not require any user interaction. Any information required for the image creation process is provided by the primary configuration file `config.xml`. The image can be customized using the `config.sh` and `images.sh` scripts.

Note

It is important to distinguish between KIWI NG (currently version 9.20.9) and its unmaintained legacy versions (7.x.x or older), now called KIWI Legacy (https://documentation.suse.com/kiwi/).

For information on how to install KIWI NG and use it to build images, see the KIWI NG documentation (https://osinside.github.io/kiwi/) ↗. A collection of example image descriptions can be found on the KIWI NG GitHub repository (https://github.com/OSInside/kiwi-descriptions) ↗.

KIWI NG's man pages provide information on using the tool. To access the man pages, install the `kiwi-man-pages` package.

3 Introduction to SLE Base Container Images

SLE Base Container Images (SLE BCIs) are minimal SLES 15-based images that you can use to develop, deploy and share applications. There are two types of SLE BCIs:

- General-purpose SLE BCIs can be used for building custom container images and for deploying applications.
- Development Stack SLE BCIs provide minimal environments for developing and deploying applications in specific programming languages.

SLE Base Container Images are available from the SUSE Registry (https://registry.suse.com) ↗. It contains tested and updated SLE Base Container Images. All images in the SUSE Registry undergo a maintenance process. The images are built to contain the latest available updates and fixes. The SUSE Registry's Web user interface lists a subset of the available images. For information about the SUSE Registry, see *Section 2.1, “SUSE Registry”*.

SLE base images in the SUSE Registry receive security updates and are covered by the SUSE support plans. For more information about these support plans, see *Section 22, “Compatibility and support conditions”*.

3.1 Why SLE Base Container Images

SLE BCIs offer a platform for creating SLES-based custom container images and containerized applications that can be distributed freely. SLE BCIs feature the same predictable enterprise lifecycle as SLES. The SLE_BCI 15 SP3 and SP4 repository (which is a subset of the SLE repository) gives SLE BCIs access to 4000 packages available for the AMD64/Intel 64, AArch64, POWER and IBM Z architectures. The packages in the repository have undergone quality assurance and security audits by SUSE. The container images are FIPS-compliant when running on a host in FIPS mode. In addition to that, SUSE can provide support for SLE BCIs through SUSE subscription plans.

Security

Each package in the SLE_BCI repository undergoes security audits, and SLE BCIs benefit from the same mechanism of dealing with CVEs as SUSE Linux Enterprise Server. All discovered and fixed vulnerabilities are announced via e-mail, the dedicated CVE pages (https://www.suse.com/security/cve/), and as OVAL and CVRF data. To ensure a secure supply chain, all container images are signed with Notary v1, Podman's GPG signatures, and Sigstore Cosign.

Stability

Since SLE BCIs are based on SLES, they feature the same level of stability and quality assurance. Similar to SLES, SLE BCIs receive maintenance updates that provide bug fixes, improvements and security patches.

Tooling and integration

SLE BCIs are designed to provide drop-in replacements for popular container images available on hub.docker.com. You can use the general-purpose SLE BCIs and the tools they put at your disposal to create custom container images, while the Development Stack SLE BCIs provide a foundation and the required tooling for building containerized applications.

Redistribution

SLE Base Container Images are covered by a permissive EULA (https://www.suse.com/de-de/licensing/eula/#bci) that allows you to redistribute custom container images based on a SLE Base Container Image.

3.2 Highlights

- SLE BCIs are fully compatible with SLES, but **they do not require a subscription to run and distribute them.**
- SLE BCIs automatically run in FIPS-compatible mode when the host operating system is running in FIPS mode.

- Each SLE BCI includes the RPM database, which makes it possible to audit the contents of the container image. You can use the RPM database to determine the specific version of the RPM package any given file belongs to. This allows you to ensure that a container image is not susceptible to known and already fixed vulnerabilities.
- All SLE BCIs (except for those without Zypper) come with the `container-suseconnect` service. This gives containers that run on a registered SLES host access to the full SLES repositories. `container-suseconnect` is invoked automatically when you run Zypper for the first time, and the service adds the correct SLES repositories into the running container. On an unregistered SLES host or on a non-SLES host, the service does nothing. See *Section 11.2, “Using container-suseconnect with SLE BCIs”* for more information.

Note: SLE_BCI repository

There is a SLE_BCI repository for each SLE service pack. This means that SLE BCIs based on SP4 have access to the SLE_BCI repository for SP4, all SLE BCIs based on SP5 use the SLE_BCI repository for SP5, and so on. Each SLE_BCI repository contains all SLE packages except kernels, boot loaders, installers (including YaST), desktop environments and hypervisors (such as KVM and Xen).

Note: Requesting a missing package

If the SLE_BCI repository does not have a package you need, you have two options. As an existing SUSE customer, you can file a feature request. As a regular user, you can request a package to be created by creating an issue in Bugzilla (https://bugzilla.suse.com/enter_bug.cgi?product=SUSE%20Linux%20Enterprise%20Base%20Container%20Images).

4 General-purpose SLE Base Container Images

There are four general-purpose SLE BCIs, and each container image comes with a minimum set of packages to keep its size small. You can use a general-purpose SLE BCI either as a starting point for building custom container images, or as a platform for deploying specific software.

SUSE offers several general-purpose SLE BCIs that are intended as deployment targets or as foundations for creating customized images: SLE BCI-Base, SLE BCI-Minimal, SLE BCI-Micro and SLE BCI-BusyBox. These images share the common SLES base, and none of them ship with a specific language or an application stack. All images feature the RPM database (even if the specific image does not include the RPM package manager) that can be used to verify the provenance of every file in the image. Each image includes the SLES certificate bundle, which allows the deployed applications to use the system's certificates to verify TLS connections.

The table below provides a quick overview of the differences between SLE BCI-Base, SLE BCI-Minimal, SLE BCI-Micro and SLE BCI-BusyBox.

TABLE 1: SUPPORT MATRIX

Features	SLE BCI-Base	SLE BCI-Minimal	SLE BCI-Micro	SLE BCI-BusyBox
glibc	✓	✓	✓	✓
CA certificates	✓	✓	✓	✓
rpm database	✓	✓	✓	✓
coreutils	✓	✓	✓	busybox
bash	✓	✓	✓	╳
rpm (binary)	✓	✓	╳	╳
zypper	✓	╳	╳	╳

4.1 SLE BCI-Base and SLE BCI-Init: When you need flexibility

SLE BCI-Base comes with the Zypper package manager and the free SLE_BCI repository. This allows you to install software available in the repository and customize the image during the build. The downside is the size of the image. It is the largest of the general-purpose SLE BCIs, so it is not always the best choice for a deployment image.

A variant of SLE BCI-Base called SLE BCI-Init comes with systemd preinstalled. The SLE BCI-Init container image can be useful in scenarios requiring systemd for managing services in a single container.

Important: Using SLE BCI-init with Docker

When using SLE BCI-init container with Docker, you must use the following arguments for `systemd` to work correctly in the container:

```
> docker run -ti --tmpfs /run -v /sys/fs/cgroup:/sys/fs/cgroup:rw --cgroupns=host
 registry.suse.com/bci/bci-init:latest
```

To correctly shut down the container, use the following command:

```
> docker kill -s SIGRTMIN+3 CONTAINER_ID
```

4.2 SLE BCI-Minimal: When you do not need Zypper

This is a stripped-down version of the SLE BCI-Base image. SLE BCI-Minimal comes without Zypper, but it does have the RPM package manager installed. This significantly reduces the size of the image. However, while RPM can install and remove packages, it lacks support for repositories and automated dependency resolution. The SLE BCI-Minimal image is therefore intended for creating deployment containers, and then installing the desired RPM packages inside the containers. Although you can install the required dependencies, you need to download and resolve them manually. However, this approach is not recommended as it is prone to errors.

4.3 SLE BCI-Micro: When you need to deploy static binaries

This image is similar to SLE BCI-Minimal but without the RPM package manager. The primary use case for the image is deploying static binaries produced externally or during multi-stage builds. As there is no straightforward way to install additional dependencies inside the container image, we recommend deploying a project using the SLE BCI-Micro image only when the final build artifact bundles all dependencies and has no external runtime requirements (like Python or Ruby).

4.4 SLE BCI-BusyBox: When you need the smallest and GPLv3-free image

Similar to SLE BCI-Micro, the SLE BCI-BusyBox image comes with the most basic tools only. However, these tools are provided by the BusyBox project. This has the benefit of further size reduction. Furthermore, the image contains no GPLv3 licensed software. When using the image, keep in mind that there are certain differences between the BusyBox tools and the GNU Coreutils. So scripts written for a system that uses GNU Coreutils may require modification to work with BusyBox.

4.5 Approximate sizes

For your reference, the list below provides an approximate size of each SLE BCI. Keep in mind that the provided numbers are rough estimations.

- `SLE BCI-Base` ~94 MB
- `SLE BCI-Minimal` ~42 MB
- `SLE BCI-Micro` ~26 MB
- `SLE BCI-BusyBox` ~14 MB

5 Using Long Term Service Pack Support container images from the SUSE Registry

Long Term Service Pack Support (LTSS) container images are available at registry.suse.com/suse/ltss/ ↗. To access and use the container images, you must have a valid LTSS subscription.

Before you can pull or download LTSS container images, you must log in to the SUSE Registry as a user. There are three ways to do that.

Use the system registration of your host system

If the host system you are using to build or run a container is already registered with the correct subscription required for accessing the LTSS container images, you can use the registration information from the host to log in to the registry.

The file /etc/zypp/credentials.d/SCCcredentials contains a username and a password. These credentials allow you to access any container that is available under the subscription of the respective host system. You can use these credentials to log in to SUSE Registry using the following commands (use the leading space before the **echo** command to avoid storing the credentials in the shell history):

```
> set +o history
> echo PASSWORD | podman login -u USERNAME --password-stdin registry.suse.com
> set -o history
```

Use a separate SUSE Customer Center registration code

If the host system is not registered with SUSE Customer Center, you can use a valid SUSE Customer Center registration code to log in to the registry:

```
> set +o history
> echo SCC_REGISTRATION_CODE | podman login -u "regcode" --password-stdin
 registry.suse.com
> set -o history
```

The user parameter in this case is the verbatim string regcode, and *SCC_REGISTRATION_CODE* is the actual registration code obtained from SUSE.

Use the organization mirroring credentials

You can also use the organization mirroring credentials to log in to the SUSE Registry:

```
> set +o history
> echo SCC_MIRRORING_PASSWORD | podman login -u "SCC_MIRRORING_USER" --password-
stdin registry.suse.com
> set -o history
```

These credentials give you access to all subscriptions the organization owns, including those related to container images in the SUSE Registry. The credentials are highly privileged and should be preferably used for a private mirroring registry only.

6 Development Stack SLE Base Container Images

Development Stack SLE BCIs are built on top of the SLE BCI-Base. Each container image comes with the Zypper stack and the free SLE_BCI repository. Additionally, each image includes most common tools for building and deploying applications in the specific language environment. This includes tools like a compiler or interpreter as well as the language-specific package manager.

Below is an overview of the Development Stack SLE BCIs available in the SUSE Registry (https://registry.suse.com).

python
: Ships with the python3 version from the tag and pip3, curl, git tools.

node
: Comes with nodejs version from the tag, npm and git. The yarn package manager can be installed with the `npm install -g yarn` command.

openjdk
: Ships with the OpenJDK runtime. Designed for deploying Java applications.

openjdk-devel
: Includes the development part of OpenJDK in addition to the OpenJDK runtime. Instead of Bash, the default entry point is the jshell shell.

ruby
: A standard development environment based on Ruby 2.5, featuring ruby, gem and bundler as well as git and curl.

rust
: Ships with the Rust compiler and the Cargo package manager.

golang
: Ships with the go compiler version specified in the tag.

dotnet-runtime
: Includes the .NET runtime from Microsoft and the Microsoft .NET repository.

dotnet-aspnet
: Ships with the ASP.NET runtime from Microsoft and the Microsoft .NET repository.

dotnet-sdk
: Comes with the .NET and ASP.NET SDK from Microsoft as well as the Microsoft .NET repository.

php
: Ships with the PHP version specified in the tag.

7 Application SLE Base Container Images

Application SLE BCIs are SLE BCI-Base container images that include specific applications, such as the PostgreSQL database and the Performance Co-Pilot a system-level performance analysis toolkit. Application SLE BCIs are available in the dedicated section of the SUSE Registry (https://registry.suse.com/#apps).

8 Important note about the support status of SLE Base Container Images

All container images, except for base, are currently classified as tech preview, and SUSE does not provide support for them. This information is visible on the web on registry.suse.com (https://registry.suse.com). It is also indicated via the `com.suse.supportlevel` label whether a container image still has the tech preview status. You can use the skopeo and jq utilities to check the status of the desired SLE BCI as follows:

```
> skopeo inspect docker://registry.suse.com/bci/bci-micro:15.4 | jq
 '.Labels["com.suse.supportlevel"]'
    "techpreview"

    > skopeo inspect docker://registry.suse.com/bci/bci-base:15.4 | jq
 '.Labels["com.suse.supportlevel"]'
    "l3"
```

In the example above, the `com.suse.supportlevel` label is set to `techpreview` in the `bci-micro` container image, indicating that the image still has the tech preview status. The `bci-base` container image, on the other hand, has full L3 support. Unlike the general purpose SLE BCIs, the Development Stack SLE BCIs may not follow the lifecycle of the SLES distribution: they are supported as long as the respective language stack receives support. In other words, new versions of SLE BCIs (indicated by the OCI tags) may be released during the lifecycle of a SLES Service Pack, while older versions may become unsupported. Refer to https://suse.com/lifecycle to find out whether the container in question is still under support.

Important

A SLE Base Container Image is no longer updated after its support period ends. You will not receive any notification when that happens.

9 SLE Base Container Image labels

SLE BCIs feature the following labels.

com.suse.eula

Marks which section of the SUSE EULA applies to the container image.

com.suse.release-stage

Indicates the current release stage of the image.

- `prototype` Indicates that the container image is in a prototype phase.
- `alpha` Prevents the container image from appearing in the registry.suse.com Web interface even if it is available there. The value also indicates the alpha quality of the container image.
- `beta` Lists the container image in the Beta Container Images section of the registry.suse.com Web interface and adds the Beta label to the image. The value also indicates the beta quality of the container image.
- `released` Indicates that the container image is released and suitable for production use.

com.suse.supportlevel

Shows the support level for the container.

- `l2` Problem isolation, which means technical support designed to analyze data, reproduce customer problems, isolate problem areas, and provide a resolution for problems not resolved by Level 1, or prepare for Level 3.
- `l3` Problem resolution, which means technical support designed to resolve problems by engaging engineering to resolve product defects which have been identified by Level 2 Support.
- `acc` Software delivered with the SLE Base Container Image may require an external contract.
- `techpreview` The image is unsupported and intended for use in proof-of-concept scenarios.
- `unsupported` No support is provided for the image.

com.suse.lifecycle-url

Points to the https://www.suse.com/lifecycle/ ↗ page that offers information about the lifecycle of the product an image is based on.

9.1 Working with SLE BCI labels

All SLE 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 therefore available for derived images and containers.

Here is an example of the labels information shown by `podman inspect`:

```
podman inspect registry.suse.com/suse/sle15
    [...]
    "Labels": {
                "com.suse.bci.base.created": "2023-01-26T22:15:08.381030307Z",
                "com.suse.bci.base.description": "Image for containers based on SUSE
 Linux Enterprise Server 15 SP4.",
                "com.suse.bci.base.disturl": "obs://build.suse.de/SUSE:SLE-15-
SP4:Update:CR/images/1477b070ae019f95b0f2c3c0dce13daf-sles15-image",
                "com.suse.bci.base.eula": "sle-bci",
                "com.suse.bci.base.image-type": "sle-bci",
                "com.suse.bci.base.lifecycle-url": "https://www.suse.com/lifecycle",
                "com.suse.bci.base.reference": "registry.suse.com/suse/
sle15:15.4.27.14.31",
                "com.suse.bci.base.release-stage": "released",
                "com.suse.bci.base.source": "https://sources.suse.com/SUSE:SLE-15-
SP4:Update:CR/sles15-image/1477b070ae019f95b0f2c3c0dce13daf/",
                "com.suse.bci.base.supportlevel": "l3",
                "com.suse.bci.base.title": "SLE 15 SP4 Base Container Image",
                "com.suse.bci.base.url": "https://www.suse.com/products/server/",
                "com.suse.bci.base.vendor": "SUSE LLC",
                "com.suse.bci.base.version": "15.4.27.14.31",
                "com.suse.eula": "sle-bci",
                "com.suse.image-type": "sle-bci",
                "com.suse.lifecycle-url": "https://www.suse.com/lifecycle",
                "com.suse.release-stage": "released",
                "com.suse.sle.base.created": "2023-01-26T22:15:08.381030307Z",
                "com.suse.sle.base.description": "Image for containers based on SUSE
 Linux Enterprise Server 15 SP4.",
                "com.suse.sle.base.disturl": "obs://build.suse.de/SUSE:SLE-15-
SP4:Update:CR/images/1477b070ae019f95b0f2c3c0dce13daf-sles15-image",
                "com.suse.sle.base.eula": "sle-bci",
                "com.suse.sle.base.image-type": "sle-bci",
                "com.suse.sle.base.lifecycle-url": "https://www.suse.com/lifecycle",
```

```
                "com.suse.sle.base.reference": "registry.suse.com/suse/
sle15:15.4.27.14.31",
                "com.suse.sle.base.release-stage": "released",
                "com.suse.sle.base.source": "https://sources.suse.com/SUSE:SLE-15-
SP4:Update:CR/sles15-image/1477b070ae019f95b0f2c3c0dce13daf/",
                "com.suse.sle.base.supportlevel": "l3",
                "com.suse.sle.base.title": "SLE 15 SP4 Base Container Image",
                "com.suse.sle.base.url": "https://www.suse.com/products/server/",
                "com.suse.sle.base.vendor": "SUSE LLC",
                "com.suse.sle.base.version": "15.4.27.14.31",
                "com.suse.supportlevel": "l3",
                "org.openbuildservice.disturl": "obs://build.suse.de/SUSE:SLE-15-
SP4:Update:CR/images/1477b070ae019f95b0f2c3c0dce13daf-sles15-image",
                "org.opencontainers.image.created": "2023-01-26T22:15:08.381030307Z",
                "org.opencontainers.image.description": "Image for containers based on
 SUSE Linux Enterprise Server 15 SP4.",
                "org.opencontainers.image.source": "https://sources.suse.com/SUSE:SLE-15-
SP4:Update:CR/sles15-image/1477b070ae019f95b0f2c3c0dce13daf/",
                "org.opencontainers.image.title": "SLE 15 SP4 Base Container Image",
                "org.opencontainers.image.url": "https://www.suse.com/products/server/",
                "org.opencontainers.image.vendor": "SUSE LLC",
                "org.opencontainers.image.version": "15.4.27.14.31",
                "org.opensuse.reference": "registry.suse.com/suse/sle15:15.4.27.14.31"
            },
    [...]
```

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

Use Podman to retrieve labels of a local image. The following command lists all labels and only the labels information of the `bci-base:15.5` image:

```
podman inspect -f {{.Labels | json}} registry.suse.com/bci/bci-base:15.5
```

It is also possible to retrieve the value of a specific label:

```
podman inspect -f {{ index .Labels \"com.suse.sle.base.supportlevel\" }}
 registry.suse.com/bci/bci-base:15.5
```

The preceding command retrieves the value of the `com.suse.sle.base.supportlevel` label.

The skopeo tool makes it possible to examine labels of an image without pulling it first. For example:

```
skopeo inspect -f {{.Labels | json}} docker://registry.suse.com/bci/bci-base:15.5
skopeo inspect -f {{ index .Labels \"com.suse.sle.base.supportlevel\" }} docker://
registry.suse.com/bci/bci-base:15.5
```

10 SLE BCI tags

Tags are used to refer to images. A tag forms a part of the image's name. Unlike labels, tags can be freely defined, and they are usually used to indicate a version number.

If a tag exists in multiple images, the newest image is used. The image maintainer decides which tags to assign to the container image.

The conventional tag format is `repository name`: `image version specification` (usually version number). For example, the tag for the latest published image of SLE 15 SP2 would be `suse/sle15:15.2`.

11 Understanding SLE BCIs

There are certain features that set SLE BCIs apart from similar offerings, like images based on Debian or Alpine Linux. Understanding the specifics can help you to get the most out of SLE BCIs in the shortest time possible.

11.1 Package manager

The default package manager in SLES is Zypper. Similar to APT in Debian and APK in Alpine Linux, Zypper offers a command-line interface for all package management tasks. Below is a brief overview of commonly used container-related Zypper commands.

Install packages

`zypper --non-interactive install PACKAGE_NAME`

Add a repository

`zypper --non-interactive addrepo REPOSITORY_URL; zypper --non-interactive refresh`

Update all packages

`zypper --non-interactive update`

Remove a package

`zypper --non-interactive remove --clean-deps PACKAGE_NAME` (the `--clean-deps` flag ensures that no longer required dependencies are removed as well)

Clean up temporary files

`zypper clean`

For more information on using Zypper, refer to https://documentation.suse.com/sles/html/SLES-all/cha-sw-cl.html#sec-zypper.

All the described commands use the `--non-interactive` flag to skip confirmations, since you cannot approve these manually during container builds. Keep in mind that you must use the flag with any command that modifies the system. Also note that `--non-interactive` is not a "yes to all" flag. Instead, `--non-interactive` confirms what is considered to be the intention of the user. For example, an installation command with the `--non-interactive` option fails if it needs to import new repository signing keys, as that is something that the user must verify themselves.

11.2 Using container-suseconnect with SLE BCIs

`container-suseconnect` (https://github.com/SUSE/container-suseconnect) is a plugin available in all SLE BCIs that ship with Zypper. When the plugin detects the host's SUSE Linux Enterprise Server registration credentials, it uses them to give the container access to the SUSE Linux Enterprise repositories. This includes additional modules and previous package versions that are not part of the free SLE_BCI repository. Refer to *Section 13.3, "container-suseconnect"* for more information on how to use the repository for SLES, openSUSE and non-SLES hosts.

11.3 Common patterns

The following examples demonstrate how to accomplish certain tasks in a SLE BCI compared to Debian.

Remove orphaned packages

- Debian: **`apt-get autoremove -y`**
- SLE BCI: Not required if you remove installed packages using the **`zypper --non-interactive remove --clean-deps`** *`PACKAGE_NAME`*

Obtain container's architecture

- Debian: **`dpkgArch="$(dpkg --print-architecture | awk -F- '{ print $NF }')"`**
- SLE BCI: **`arch="$(uname -p)"`**

Install packages required for compilation

- Debian: `apt-get install -y build-essential`
- SLE BCI: `zypper -n in gcc gcc-c++ make`

Verify GnuPG signatures

- Debian: `gpg --batch --verify SIGNATURE_URL FILE_TO_VERIFY`
- SLE BCI: `zypper -n in dirmngr; gpg --batch --verify SIGNATURE_URL FILE_TO_VERIFY; zypper -n remove --clean-deps dirmngr; zypper -n clean`

11.4 Package naming conventions

SLE package naming conventions differ from Debian, Ubuntu and Alpine, and they are closer to those of Red Hat Enterprise Linux. The main difference is that development packages of libraries (that is, packages containing headers and build description files) are named `PACKAGE-devel` in SLE, as opposed to `PACKAGE-dev` in Debian and Ubuntu. When in doubt, search for the package using the following command: `docker run --rm registry.suse.com/bci/bci-base:OS_VERSION zypper search PACKAGE_NAME` (replace `OS_VERSION` with the appropriate service version number, for example: `15.3` or `15.4`).

11.5 Adding GPG signing keys

The SUSE keys are already in the RPM database of the SLE Base Container Image. This means that you do not have to import them.

However, adding external repositories to a container or container image normally requires importing the GPG key used for signing the packages. This can be done with the `rpm --import KEY_URL` command. This adds the key to the RPM database, and all packages from the repository can be installed afterwards.

12 Getting started with SLE Base Container Images

The SLE BCIs are available as OCI-compatible container images directly from the SUSE Registry (https://registry.suse.com), and they can be used like any other container image, for example:

```
> podman run --rm -it registry.suse.com/bci/bci-base:15.4 grep '^NAME' /etc/os-release
 NAME="{sles}"
```

Alternatively, you can use a SLE BCI in `Dockerfile` as follows:

```
FROM registry.suse.com/bci/bci-base:15.4
    RUN zypper --non-interactive in python3 && \
        echo "Hello Green World!" > index.html
    ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
    EXPOSE 8000
```

You can then build container images using the **`docker build .`** or **`buildah bud .`** commands:

```
> docker build .
    Sending build context to Docker daemon  2.048kB
    Step 1/4 : FROM registry.suse.com/bci/bci-base:15.4
     ---> e34487b4c4e1
    Step 2/4 : RUN zypper --non-interactive in python3 &&     echo "Hello Green World!" >
 index.html
     ---> Using cache
     ---> 9b527dfa45e8
    Step 3/4 : ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
     ---> Using cache
     ---> 953080e91e1e
    Step 4/4 : EXPOSE 8000
     ---> Using cache
     ---> 48b33ec590a6
    Successfully built 48b33ec590a6

    > docker run -p 8000:8000 --rm -d 48b33ec590a6
    575ad7edf43e11c2c9474055f7f6b7a221078739fc8ce5765b0e34a0c899b46a

    > curl localhost:8000
    Hello Green World!
```

13 Registration and online repositories

As a pre-requisite to work with containers on a SUSE Linux Enterprise Server, you have to enable the SLE Containers Module. This consists of container-related packages, including container engine and core container tools like on-premise registry. For more information about SLE Modules, refer to https://documentation.suse.com/sles/html/SLES-all/article-modules.html.

The regular SLE subscription includes SLE Containers Module free of charge.

13.1 Enabling the Containers Module using the YaST graphical interface

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 `Containers Module 15 SP4 x86_64` and click *Next*. This adds the `Containers Module` and its repositories to the system.
5. If you use Repository Mirroring Tool, update the list of repositories on the RMT server.

13.2 Enabling the Containers Module from the command line using SUSEConnect

The Containers Module can also be added with the following command:

```
> sudo SUSEConnect -p sle-module-containers/15.4/x86_64
```

13.3 container-suseconnect

`container-suseconnect` (https://github.com/SUSE/container-suseconnect) is a plugin available in all SLE Base Container Images that ship with Zypper. When the plugin detects the host's SUSE Linux Enterprise Server registration credentials, it uses them to give the container access the SUSE Linux Enterprise repositories. This includes additional modules providing access to all packages included in SLES.

13.3.1 Using container-suseconnect on SLES and openSUSE

If you are running a registered SLES system with Docker, `container-suseconnect` automatically detects and uses the subscription, without requiring any action on your part.

On openSUSE systems with Docker, you must copy the files `/etc/SUSEConnect` and `/etc/zypp/credentials.d/SCCcredentials` from a registered SLES machine to your local machine. Note that the `/etc/SUSEConnect` file is required only if you are using RMT for managing your registration credentials.

13.3.2 Using container-suseconnect on non-SLES hosts or with Podman and Buildah

You need a registered SLES system to use `container-suseconnect` on non-SLE hosts or with Podman and Buildah. This can be a physical machine, a virtual machine, or the SLE BCI-Base container with `SUSEConnect` installed and registered.

If you do not use RMT, copy `/etc/zypp/credentials.d/SCCcredentials` to the development machine. Otherwise, copy both the `/etc/zypp/credentials.d/SCCcredentials` and `/etc/SUSEConnect` files.

You can use the following command to obtain `SCCcredentials` (replace *`REGISTRATION_CODE`* with your SCC registration code)

```
podman run --rm registry.suse.com/suse/sle15:latest bash -c \
        "zypper -n in SUSEConnect; SUSEConnect --regcode REGISTRATION_CODE; \
         cat /etc/zypp/credentials.d/SCCcredentials"
```

If you are running a container based on a SLE BCI, mount `SCCcredentials` (and optionally `/etc/SUSEConnect`) in the correct destination. The following example shows how to mount `SCCcredentials` in the current working directory:

```
podman run -v /path/to/SCCcredentials:/etc/zypp/credentials.d/SCCcredentials \
        -it --pull=always registry.suse.com/bci/bci-base:latest
```

Do not copy the `SCCcredentials` and `SUSEConnect` files into the container image to avoid inadvertently adding them to the final image. Use secrets instead, as they are only available to a single layer and are not part of the built image. To do this, put a copy of `SCCcredentials` (and optionally `SUSEConnect`) somewhere on the file system and modify the **`RUN`** instructions that invoke Zypper as follows:

```
FROM registry.suse.com/bci/bci-base:latest
```

```
RUN --mount=type=secret,id=SUSEConnect \
    --mount=type=secret,id=SCCcredentials \
    zypper -n in fluxbox
```

Buildah support mounting secrets via the `--secret` flag as follows:

```
buildah bud --layers --secret=id=SCCcredentials,src=/path/to/SCCcredentials \
        --secret=id=SUSEConnect,src=/path/to/SUSEConnect .
```

Note: known issue

container-suseconnect runs automatically every time you invoke Zypper. If you are not using a registered SLES host, you may see the following error message:

```
> zypper ref
    Refreshing service 'container-suseconnect-zypp'.
    Problem retrieving the repository index file for service 'container-
suseconnect-zypp':
    [container-suseconnect-zypp|file:/usr/lib/zypp/plugins/services/container-
suseconnect-zypp]
    Warning: Skipping service 'container-suseconnect-zypp' because of the above
 error.
```

Ignore the message, as it simply indicates that container-suseconnect was not able to retrieve your SUSE Customer Center credentials, and thus could not add the full SLE repositories. You still have full access to the SLE_BCI repository, and can continue using the container as intended.

13.3.3 Adding modules into the container or container Image

`container-suseconnect` allows you to automatically add SLE Modules into a container or container image. What modules are added is determined by the environment variable `ADDITIONAL_MODULES` that includes a comma-separated list of the module names. In a `Dockerfile`, this is done using the `ENV` directive as follows:

```
FROM registry.suse.com/bci/bci-base:latest
    ENV ADDITIONAL_MODULES sle-module-desktop-applications,sle-module-development-tools

    RUN --mount=type=secret,id=SCCcredentials zypper -n in fluxbox && zypper -n clean
```

14 Podman overview

Podman (https://podman.io/) is short for Pod Manager Tool. It is a daemonless container engine for managing Open Container Initiative (OCI) containers on a Linux system. By default, Podman supports rootless containers, which reduces attack surface when running containers. Podman can be used to create OCI-compliant container images using a `Dockerfile` and a range of commands identical to Docker Open Source Engine. For example, the `podman build` command performs the same task as `docker build`. In other words, Podman provides a drop-in replacement for Docker Open Source Engine.

Moving from Docker Open Source Engine to Podman does not require any changes in the established workflow. There is no need to rebuild images, and you can use the exact same commands to build and manage images as well as run and control containers.

Podman differs from Docker Open Source Engine in the following ways:

- Podman does not use a daemon, so the container engine interacts directly with an image registry, containers and image storage when needed.
- Podman features native systemd integration that allows for the use of systemd to run containers. Generating the required systemd unit files is supported by Podman using the **`podman generate systemd`** command. Moreover, Podman can run systemd inside containers.
- Podman does not require root privileges to create and run containers. This means that Podman can run under the `root` user as well as in an unprivileged environment. Moreover, a container created by an unprivileged user cannot get higher privileges on the host than the container's creator.
- Podman can be configured to search multiple registries by reading `/etc/containers/registries.conf` file.
- Podman can deploy applications from Kubernetes manifests
- Podman supports launching systemd inside a container and requires no potentially dangerous workarounds.

Podman makes it possible to group containers into pods. Pods share the same network interface. A typical scenario for grouping containers into a pod is a container that runs a database and a container with a client that accesses the database. For further information about pods, refer to *Section 21.1, "Single container host with Podman"*.

14.1 Podman installation

To install Podman, make sure you have the SLE Containers Module enabled (see *Section 13, “Registration and online repositories”*), run the command **`sudo zypper in podman`**. Then run **`podman info`** to check whether Podman has been installed successfully.

By default, Podman launches containers as the current user. For unprivileged users, this means launching containers in rootless mode. Support for rootless containers is enabled for all newly created users in SLE by default, and no additional steps are necessary.

In case Podman fails to launch containers in rootless mode, check whether an entry for the current user is present in `/etc/subuid`:

```
> grep $(id -nu) /etc/subuid
    user:10000:65536
```

When no entry is found, add the required sub-UID and sub-GID entries via the following command:

```
> sudo usermod --add-subuids 100000-165535 --add-subgids 100000-165535 $(id -nu)
```

To enable the change, reboot the machine or stop the session of the current user. To do the latter, run **`loginctl list-sessions | grep`** *`USER`* and note the session ID. Then run **`loginctl kill-session`** *`SESSION_ID`* to stop the session.

The **`usermod`** above defines a range of local UIDs to which the UIDs allocated to users inside the container are mapped on the host. Note that the ranges defined for different users must not overlap. It is also important that the ranges do not reuse the UID of an existing local user or group. By default, adding a user with the **`useradd`** command on SUSE Linux Enterprise automatically allocates sub-UID and sub-GID ranges.

When using rootless containers with Podman, it is recommended to use cgroups v2. cgroups v1 are limited in terms of functionality compared to v2. For example, cgroups v1 do not allow proper hierarchical delegation to the user's subtrees. Additionally, Podman is unable to read container logs properly with cgroups v1 and the systemd log driver. To enable cgroups v2, add the following to the kernel cmdline: `systemd.unified_cgroup_hierarchy=1`

Running a container with Podman in rootless mode on SUSE Linux Enterprise Server may fail, because the container needs read access to the SUSE Customer Center credentials. For example, running a container with the command **`podman run -it --rm registry.suse.com/suse/sle15 bash`** and then executing **`zypper ref`** results in the following error message:

```
Refreshing service 'container-suseconnect-zypp'.
```

```
    Problem retrieving the repository index file for service 'container-suseconnect-
zypp':
    [container-suseconnect-zypp|file:/usr/lib/zypp/plugins/services/container-
suseconnect-zypp]
    Warning: Skipping service 'container-suseconnect-zypp' because of the above error.
    Warning: There are no enabled repositories defined.
    Use 'zypper addrepo' or 'zypper modifyrepo' commands to add or enable repositories
```

To solve the problem, grant the current user the required access rights by running the following command on the host:

```
> sudo setfacl -m u:$(id -nu):r /etc/zypp/credentials.d/*
```

Log out and log in again to apply the changes.

To give multiple users the required access, create a dedicated group using the **`groupadd`** *`GROUP-NAME`* command. Then use the following command to change the group ownership and rights of files in the `/etc/zypp/credentials.d/` directory.

```
> sudo chgrp GROUPNAME /etc/zypp/credentials.d/*
      > sudo chmod g+r /etc/zypp/credentials.d/*
```

You can then grant a specific user write access by adding them to the created group.

14.1.1 Tips and tricks for rootless containers

Podman remaps user IDs with rootless containers. In the following example, Podman remaps the current user to the default user in the container:

```
> podman run --rm -it registry.suse.com/bci/bci-base id
    uid=0(root) gid=0(root) groups=0(root)
```

Note that even if you are root in the container, you cannot gain superuser privileges outside of it.

This user remapping can have undesired side effects when sharing data with the host, where the shared files belong to different user IDs in the container and on the host. The issue can be solved using the command-line flag `--userns=keep-id` that makes it possible to keep the current user id in the container:

```
> podman run --userns=keep-id --rm -it registry.suse.com/bci/bci-base id
    uid=1000(user) gid=1000(users) groups=1000(users)
```

The flag `--userns=keep-id` has a similar effect when used with bind mounts:

```
> podman run --rm -it -v $(pwd):/share/ registry.suse.com/bci/bci-base stat /share/
      File: /share/
```

```
      Size: 318             Blocks: 0          IO Block: 4096   directory
    Device: 2ch/44d Inode: 3506170     Links: 1
    Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
    Access: 2023-05-03 12:52:18.636392618 +0000
    Modify: 2023-05-03 12:52:17.178380923 +0000
    Change: 2023-05-03 12:52:17.178380923 +0000
     Birth: 2023-05-03 12:52:15.852370288 +0000

     > podman run --userns=keep-id --rm -it -v $(pwd):/share/ registry.suse.com/bci/bci-
base stat /share/
      File: /share/
      Size: 318             Blocks: 0          IO Block: 4096   directory
    Device: 2ch/44d Inode: 3506170     Links: 1
    Access: (0755/drwxr-xr-x)  Uid: ( 1000/    user)   Gid: ( 1000/    users)
    Access: 2023-05-03 12:52:18.636392618 +0000
    Modify: 2023-05-03 12:52:17.178380923 +0000
    Change: 2023-05-03 12:52:17.178380923 +0000
     Birth: 2023-05-03 12:52:15.852370288 +0000
```

Podman stores the containers' data in the storage graph root (default is `~/.local/share/containers/storage`). Because of the way Podman remaps user IDs in rootless containers, the graph root may contain files that are not owned by your current user but by a user ID in the sub-UID region assigned to your user. As these files do not belong to your current user, they can be inaccessible to you.

To read or modify any file in the graph root, enter a shell as follows:

```
> podman unshare bash

    > id
    uid=0(root) gid=0(root) groups=0(root),65534(nobody)
```

Note that **`podman unshare`** performs the same user remapping as **`podman run`** does when launching a rootless container. You cannot gain elevated privileges via **`podman unshare`**.

Do not modify files in the graph root as this can corrupt Podman's internal state and render your containers, images and volumes inoperable.

14.1.2 Caveats of rootless containers

Because unprivileged users cannot configure network namespaces on Linux, Podman relies on a userspace network implementation called `slirp4netns`. It emulates the full TCP-IP stack and can cause a heavy performance degradation for workloads relying on high network transfer rates. This means that rootless containers suffer from slow network transfers.

On Linux, unprivileged users cannot open ports below port number 1024. This limitation also applies to Podman, so by default, rootless containers cannot expose ports below port number 1024. You can remove this limitation using the following command: **`sysctl net.ipv4.ip_unprivileged_port_start=0`**.

To remove the limitation permanently, run **`sysctl -w net.ipv4.ip_unprivileged_port_start=0`**.

Note that this allows **all** unprivileged applications to bind to ports below 1024.

14.1.3 podman-docker

Because Podman is compatible with Docker Open Source Engine, it features the same command-line interface. You can also install the package `podman-docker` that allows you to use an emulated Docker CLI with Podman. For example, the **`docker pull`** command, that fetches a container image from a registry, executes **`podman pull`** instead. The **`docker build`** command executes **`podman build`**, etc.

Podman also features a Docker Open Source Engine compatible socket that can be launched using the following command:

```
> sudo systemctl start podman.socket
```

The Podman socket can be used by applications designed to communicate with Docker Open Source Engine to launch containers transparently via Podman. The Podman socket can be used to launch containers using **`docker compose`**, without running Docker Open Source Engine.

14.2 Obtaining container images

14.2.1 Configuring container registries

By default, Podman is configured to use SUSE Registry only. To make Podman search the SUSE Registry first and use Docker Hub as a fallback, make sure that the `/etc/containers/registries.conf` file contains the following configuration:

```
unqualified-search-registries = ["registry.suse.com", "docker.io"]
```

14.2.2 Searching images in registries

Using the **`podman search`** command allow you to list available containers in the registries defined in `/etc/containers/registries.conf`.

To search in all registries:

```
podman search go
```

To search in a specific registry:

```
podman search registry.suse.com/go
```

14.2.3 Downloading (pulling) images

The **`podman pull`** command pulls an image from an image registry:

```
> podman pull REGISTRY:PORT/NAMESPACE/NAME:TAG
```

For example:

```
> podman pull registry.suse.com/bci/bci-base
```

Note that if you do not specify a tag, Podman pulls the `latest` tag.

14.3 Renaming images and image tags

Tags are used to assign descriptive names to container images, thus making it easier to identify individual images.

Pull the SLE BCI-Base image from SUSE Registry:

```
> podman pull registry.suse.com/bci/bci-base
   Trying to pull registry.suse.com/bci/bci-base:latest...
   Getting image source signatures
   Copying blob bf6ca87723f2 done
   Copying config 34578a383c done
   Writing manifest to image destination
   Storing signatures
   34578a383c7b6fdcb85f90fbad59b7e7a16071cf47843688e90fe20ff64a684
```

List the pulled images:

```
> podman images
   REPOSITORY                      TAG         IMAGE ID      CREATED        SIZE
```

```
    registry.suse.com/bci/bci-base  latest      34578a383c7b  22 hours ago   122 MB
```

Rename the SLE BCI-Base image to `my-base`:

```
podman tag 34578a383c7b my-base
```

```
podman images
    REPOSITORY                      TAG         IMAGE ID      CREATED        SIZE
    registry.suse.com/bci/bci-base  latest      34578a383c7b  22 hours ago   122 MB
    localhost/my-base               latest      34578a383c7b  22 hours ago   122 MB
```

Add a custom tag `1` (indicating that this version 1 of the image) to `my-base`:

```
> podman tag 34578a383c7b my-base:1
```

```
> podman images
    REPOSITORY                      TAG         IMAGE ID      CREATED        SIZE
    registry.suse.com/bci/bci-base  latest      34578a383c7b  22 hours ago   122 MB
    localhost/my-base               latest      34578a383c7b  22 hours ago   122 MB
    localhost/my-base               1           34578a383c7b  22 hours ago   122 MB
```

Note that the default tag `latest` is still present.

14.4 Deploying container images

Similar to Docker Open Source Engine, Podman can run containers in an interactive mode, allowing you to inspect and work with an image. To run the `suse/sle15` image in interactive mode, use the following command:

```
> podman run --rm -ti suse/sle15
```

Note: Fixing the lost network access to rootful containers

Rootful containers may occasionally become inaccessible from the outside. The issue is caused by firewalld reloading its permanent rules and discarding any temporary rules created by Podman's networking back-end (either CNI or Netavark). A temporary workaround is to reload the Podman network using the **`podman network reload --all`** command. If you use Netavark 1.9.0 or higher as the network back-end, a permanent fix to the problem is to use the `netavark-firewalld-reload.service` service. Enable and start the service as follows:

```
# systemctl enable netavark-firewalld-reload.service
```

```
# systemctl restart netavark-firewalld-reload.service
```

You can check which back-end and version you are using by running `podman info --format "{{.Host.NetworkBackend}}"` and `podman info --format "{{.Host.NetworkBackendInfo.Version}}"`, respectively.

We recommend adding permanent firewall rules for containers you want to be accessible from outside of the host. This ensures that the rules persist on firewall reloads and system reboots. This approach also offers greater flexibility (for example, it allows you to assign the rules to a certain firewalld zone).

14.5 Building images with Podman

Podman can build images from a `Dockerfile`. The **`podman build`** command behaves as **`docker build`**, and it accepts the same options.

Podman's companion tool Buildah provides an alternative way to build images. For further information about Buildah, refer to *Section 18, "Buildah overview"*.

15 Setting up Docker Open Source Engine

15.1 Preparing the host

Before installing any Docker-related packages, you need to enable the Containers Module:

Note: Built-in Docker orchestration support

Container orchestration is a part of Docker Open Source Engine. Even though this feature is available in SUSE Linux Enterprise, it is not supported by SUSE, and is only provided as a technology preview. Use Kubernetes for container orchestration. For details, refer to the Kubernetes documentation (https://kubernetes.io/docs/getting-started-guides/kubeadm/).

15.1.1 Installing and configuring Docker Open Source Engine

1. Install the `docker` package:

```
> sudo zypper install docker
```

2. Enable the Docker service, so it starts automatically at boot time:

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

   This also enables `docker.socket`.

3. Open the `/etc/sysconfig/docker` file. Search for the parameter `DOCKER_OPTS` and add `--insecure-registry ADDRESS_OF_YOUR_REGISTRY`.

   a. Add CA certificates to the directory `/etc/docker/certs.d/REGISTRY_ADDRESS`:

```
> sudo cp CA /etc/pki/trust/anchors/
```

   b. Copy the CA certificates to your system:

```
> sudo update-ca-certificates
```

4. Start the Docker service:

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

   This also starts `docker.socket`.

The Docker daemon listens on a local socket accessible only by the root user and by the members of the docker group. The docker group is automatically created during package installation.

To allow a certain user to connect to the local Docker daemon, use the following command:

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

This allows the user to communicate with the local Docker daemon.

15.2 Configuring the network

To give the containers access to the external network, enable the `ipv4 ip_forward` rule.

15.2.1 How Docker Open Source Engine interacts with iptables

To learn more about how containers interact with each other and the system firewall, see the Docker documentation (https://docs.docker.com/v17.09/engine/userguide/networking/default_network/container-communication/) ↗.

It is also possible to prevent Docker Open Source Engine from manipulating `iptables`. See the Docker documentation (https://docs.docker.com/network/iptables/#prevent-docker-from-manipulating-iptables) ↗.

15.3 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 driver is simpler than the others listed and does not offer certain advantages of Docker Open Source Engine such as shared layers. It is a slow but reliable driver.
- devicemapper: This driver relies on the device-mapper thin provisioning module. It supports copy-on-write, so it provides all the advantages of Docker Open Source Engine.
- btrfs: This driver relies on Btrfs to offer 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.

SUSE Linux Enterprise uses the Btrfs file system by default. This forces Docker Open Source Engine to use the btrfs driver.

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

For example, to enable the devicemapper driver, enter the following text:

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

Important: Mounting `/var/lib/docker`

It is recommended to mount `/var/lib/docker` on a separate partition or volume. In case of file system corruption, this allows the operating system to run Docker Open Source Engine unaffected.

If 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 you do not exclude `/var/lib/docker` from snapshots, there is a risk of the file system running out of disk space soon after you start deploying containers. Moreover, a rollback to a previous snapshot will also reset the Docker database and images. For more information, see https://documentation.suse.com/sles/html/SLES-all/cha-snapper.html#sec-snapper-setup-customizing-new-subvolume.

15.4 Updates

All updates to the `docker` package are marked as interactive (that is, no automatic updates) to avoid accidental updates that can break running container workloads. Stop all running containers before applying a Docker Open Source Engine update.

To prevent data loss, avoid workloads that rely on containers that automatically start after Docker Open Source Engine update. Although it is technically possible to keep containers running during an update via the `--live-restore` option, such updates can introduce regressions. SUSE does not support this feature.

16 Configuring image storage

Before creating custom images, decide where you want to store images. A simple solution is to push images to Docker Hub (https://hub.docker.com). By default, all images pushed to Docker Hub are public. Make sure not to publish sensitive data or software not licensed for public use.

You can restrict access to custom container images with the following:

- Docker Hub allows creating private repositories for paid subscribers.
- An on-site Docker Registry allows storing all the container images used by your organization.

Instead of using Docker Hub, you can run a local instance of Docker Registry, an open source platform for storing and retrieving container images.

16.1 Running a Docker Registry

The SUSE Registry provides a container image that makes it possible to run a local Docker Registry as a container. It stores the pushed images in a container volume corresponding to the directory `/var/lib/docker-registry`. It is recommended to either create a named volume for the registry or to bind mount a persistent directory on the host to `/var/lib/docker-registry` in the container. This ensures that pushed images persist deleting the container.

Run the following command to pull the registry container image from the SUSE Registry and start a container that can be accessed on port 5000 with the container storage bind mounted locally to */PATH/DIR/*:

```
> podman run -d --restart=always --name registry -p 5000:5000 \
    -v /PATH/DIR:/var/lib/docker-registry registry.suse.com/suse/registry
```

Alternatively, create a named volume *registry* for the SUSE Registry container as follows:

```
> podman run -d --restart=always --name registry -p 5000:5000 \
     -v registry:/var/lib/docker-registry registry.suse.com/suse/registry
```

To make it easier to manage the registry, create a corresponding system unit:

```
>  podman generate systemd registry | \
     sudo tee /etc/systemd/system/suse_registry.service
```

Enable and start the registry service, then verify its status:

```
> sudo systemctl enable suse_registry.service
      > sudo systemctl start suse_registry.service
      > sudo systemctl status suse_registry.service
```

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

16.2 Limitations

Docker Registry has two major limitations:

- It lacks any form of authentication. That means everybody with access to Docker Registry can push and pull images to it. That includes overwriting existing images. It is recommended to setup some form of access restriction as described in the upstream documentation https://distribution.github.io/distribution/about/deploying/#restricting-access ↗.
- It is not possible to see which images have been pushed to Docker Registry. You need to keep a record of what is being stored on it. There is also no search functionality.

17 Verifying container images

Verifying container images allows you to confirm their provenance, thus ensuring the supply chain security. This chapter provides information on signing and verifying container images.

17.1 Verifying SLE Base Container Images with Docker

Signatures for images available through SUSE Registry are stored in the Notary. You can verify the signature of a specific image using the following command:

```
> docker trust inspect --pretty registry.suse.com/suse/IMAGE:TAG
```

For example, the command **`docker trust inspect --pretty registry.suse.com/suse/sle15:latest`** verifies the signature of the latest SLE15 base image.

To automatically validate an image when you pull it, set the environment `DOCKER_CONTENT_TRUST` to `1`. For example:

```
env DOCKER_CONTENT_TRUST=1 docker pull registry.suse.com/suse/sle15:latest
```

17.2 Verifying SLE Base Container Images with Cosign

To verify a SLE BCI, run Cosign in the container. The command below fetches the signing key from the SUSE server and uses it to verify the latest SLE BCI-Base container image.

```
> podman run --rm -it registry.suse.com/suse/cosign verify \
        --key https://ftp.suse.com/pub/projects/security/keys/container-key.pem \
        registry.suse.com/bci/bci-base:latest | tail -1 | jq

    [
      {
        "critical": {
          "identity": {
            "docker-reference": "registry.suse.com/bci/bci-base"
          },
          "image": {
            "docker-manifest-digest":
 "sha256:52a828600279746ef669cf02a599660cd53faf4b2430a6b211d593c3add047f5"
          },
          "type": "cosign container image signature"
        },
        "optional": {
```

```
          "creator": "OBS"
        }
      }
    ]
```

The signing key can be used to verify all SLE BCIs, and it also ships with SUSE Linux Enterprise (the `/usr/share/container-keys/suse-container-key.pem` file).

You can also check SLE BCIs against rekor (https://github.com/sigstore/rekor), the immutable tamper resistant ledger. For example:

```
> podman run --rm -it -e COSIGN_EXPERIMENTAL=1 registry.suse.com/suse/cosign \
      verify --key https://ftp.suse.com/pub/projects/security/keys/container-key.pem \
      registry.suse.com/bci/bci-base:latest | tail -1 | jq
    [
      {
        "critical": {
          "identity": {
            "docker-reference": "registry.suse.com/bci/bci-base"
          },
          "image": {
            "docker-manifest-digest":
 "sha256:52a828600279746ef669cf02a599660cd53faf4b2430a6b211d593c3add047f5"
          },
          "type": "cosign container image signature"
        },
        "optional": {
          "creator": "OBS"
        }
      }
    ]
```

If verification fails, the output of the **`cosign verify`** command is similar to the one below.

```
Error: no matching signatures:
    crypto/rsa: verification error
    main.go:62: error during command execution: no matching signatures:
    crypto/rsa: verification error
```

17.3 Verifying Helm charts with Cosign

Cosign can also be used to verify Helm charts. This can be done using the following command:

```
> cosign verify --key https://ftp.suse.com/pub/projects/security/keys/container-key.pem
 registry.suse.com/path/to/chart
```

17.4 Verifying SLE Base Container Images with Podman

Before you can verify SLE BCIs using Podman, you must specify `registry.suse.com` as the registry for image verification.

Note

Skip this step on SUSE Linux Enterprise, as the correct configuration is already in place.

To do this, add the following configuration to `/etc/containers/registries.d/default.yaml`:

```
docker:
      registry.suse.com:
        use-sigstore-attachments: true
```

Instead of editing the `default.yaml`, you can create a new file in `/etc/containers/registries.d/` with a filename of your choice.

Next, modify the /etc/containers/policy.json (https://github.com/containers/image/blob/main/docs/containers-policy.json.5.md) file. Under the `docker` attribute, add the `registry.suse.com` configuration similar to the following:

```
{
      "default": [
        {
          "type": "insecureAcceptAnything"
        }
      ],
      "transports": {
        "docker-daemon": {
          "": [
            {
              "type": "insecureAcceptAnything"
            }
          ]
        },
        "docker": {
          "registry.suse.com": [
            {
              "type": "sigstoreSigned",
              "keyPath": "/usr/share/pki/containers/suse-container-key.pem",
              "signedIdentity": {
                "type": "matchRepository"
```

```
            }
          }
        ]
      }
    }
  }
```

The specified configuration instructs Podman, skopeo and Buildah to verify images under the `registry.suse.com` repository. This way,Podman checks the validity of the signature using the specified public key before pulling the image. It rejects the image if the validation fails.

Note

Do not remove existing entries in `transports.docker`. Append the entry for `registry.suse.com` to the list.

Fetch the public key used to sign SLE BCIs from SUSE Signing Keys (https://www.suse.com/support/security/keys/), or use the following command:

```
> sudo curl -s https://ftp.suse.com/pub/projects/security/keys/container–key.pem \
        -o /usr/share/pki/containers/suse-container-key.pem
```

Note

This step is optional on SUSE Linux Enterprise. The signing key is already available in `/usr/share/pki/containers/suse-container-key.pem`

Buildah, Podman and skopeo automatically verifies every image pulled from `registry.suse.com` from now on. There are no additional steps required.

If verification fails, the command returns an error message as follows:

```
> podman pull registry.suse.com/bci/bci-base:latest
    Trying to pull registry.suse.com/bci/bci-base:latest...
    Error: copying system image from manifest list: Source image rejected: Signature for
 identity registry.suse.com/bci/bci-base is not accepted
```

If there are no issues with the signed image and your configuration, you can proceed with using the container image.

18 Buildah overview

Buildah (https://buildah.io/) is tool for building OCI-compliant container images. Buildah can handle the following tasks:

- Create containers from scratch or from existing images.
- Create an image from a working container or via a `Dockerfile`.
- Build images in the OCI or Docker Open Source Engine image formats.
- Mount a working container's root file system for manipulation.
- Use the updated contents of a container's root file system as a file system layer to create a new image.
- Delete a working container or an image and rename a local container.

Compared to Docker Open Source Engine, Buildah offers the following advantages:

- The tool makes it possible to mount a working container's file system, so it becomes accessible by the host.
- The process of building container images using Buildah can be automated via scripts by using Buildah subcommands instead of a `Containerfile` or `Dockerfile`.
- Similar to Podman, Buildah does not require a daemon to run and can be used by unprivileged users.
- It is possible to build images inside a container without mounting the Docker socket, which improves security.

Both Podman and Buildah can be used to build container images. While Podman makes it possible to build images using Dockerfiles, Buildah offers an expanded range of image building options and capabilities.

18.1 Buildah installation

To install Buildah, run the command **`sudo zypper in buildah`**. Run the command **`buildah --version`** to check whether Buildah has been installed successfully.

If you already have Podman installed and set up for use in rootless mode, Buildah can be used in an unprivileged environment without any further configuration. If you need to enable rootless mode for Buildah, run the following command:

```
> sudo usermod --add-subuids 100000-165535 --add-subgids 100000-165535 USER
```

This command enables rootless mode for the current user. After running the command, log out and log in again to enable the changes.

The command above defines a range of local UIDs on the host, onto which the UIDs allocated to users inside the container are mapped. Note that the ranges defined for different users must not overlap. It is also important that the ranges do not reuse the UID of any existing local users or groups. By default, adding a user with the **`useradd`** command on SUSE Linux Enterprise automatically allocates subUID and subGID ranges.

Note: Buildah in rootless mode

In rootless mode, Buildah commands must be executed in a modified user namespace of the user. To enter this user namespace, run the command **`buildah unshare`**. Otherwise, the **`buildah mount`** command will fail.

18.2 Building images with Buildah

Instead of a special file with instructions, Buildah uses individual commands to build an image. Building an image with Buildah involves the following steps:

- run a container based on the specified image
- edit the container (install packages, configure settings, etc.)
- configure the container options
- commit all changes into a new image

While this process may include additional steps, such as mounting the container's file system and working with it, the basic workflow logic remains the same.

The following example can give you a general idea of how to build an image with Buildah.

```
container=$(buildah from suse/sle15) 1
```

```
buildah run $container zypper up ❷
buildah copy $container . /usr/src/example/ ❸
buildah config --workingdir /usr/src/example $container ❹
buildah config --port 8000 $container
buildah config --cmd "php -S 0.0.0.0:8000" $container
buildah config --label maintainer="Tux" $container ❺
buildah config --label version="0.1" $container
buildah commit $container example ❻
buildah rm $container ❼
```

❶ Specify a container (also called a working container) based on the specified image (in this case, `sle15`).

❷ Run a command in the working container you just created. In this example, Buildah runs the **`zypper up`** command.

❸ Copy files and directories to the specified location in the container. In this example, Buildah copies the entire contents of the current directory to `/usr/src/example/`.

❹ The **`buildah config`** commands specify container options. These include defining a working directory, exposing a port, and running a command inside the container.

❺ The **`buildah config --label`** command allows you to assign labels to the container. This may include `maintainer`, `description`, `version`, and so on.

❻ Create an image from the working container by committing all the modifications.

❼ Delete the working container.

19 Creating custom container images

To create a custom image, you need a base image of SUSE Linux Enterprise Server. You can use any of the pre-built SUSE Linux Enterprise Server images.

19.1 Pulling base SUSE Linux Enterprise Server images

To obtain a pre-built base image, use the following command:

```
> podman pull registry.suse.com/suse/IMAGENAME
```

For example, for SUSE Linux Enterprise Server 15, the command is as follows:

```
> podman pull registry.suse.com/suse/sle15
```

For information on obtaining specific base images, refer to *Section 3, "Introduction to SLE Base Container Images"*.

When the container image is ready, you can customize it as described in *Section 19.2, "Customizing container images"*.

19.2 Customizing container images

19.2.1 Repositories and registration

The pre-built images do not have any repositories 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 or a Repository Mirroring Tool (RMT) server, according to the configuration of the SUSE Linux Enterprise Server host that runs the container. The service obtains the list of repositories available for the product used by the container image. You can also directly declare extensions in your `Dockerfile`. For more information, see *Section 13.3, "container-suseconnect"*.

Note: SLE_BCI repository

The default base image includes the SLE_BCI repository. This repository is only used when a container is built or runs on a non-registered SLES host, or when registration credentials are not made available to the container. The repository provides a subset of SLE packages useful for customizing SLES container images. The repository is available without any registration, and it is not supported.

You do not need to add any credentials to the container image, because the machine credentials are automatically injected into the `/run/secrets` directory in the container by the docker daemon. 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 included in a container image, which means that there is no risk of your credentials leaking.

Note: Building images on systems registered with RMT

When the host system used for building container images is registered with RMT, the default behavior allows only building containers of the same code base as the host. For example, if your container host is an 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 an SLE 15 host, the host machine credentials for the target release can be injected into the container as outlined below.

Note that if the RMT server is using a self-signed certificate, the matching CA certificate needs to be added into the container at `CA_TRUSTSTORE/rmt-server.pem` for the certificate to be accepted.

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

To obtain the list of repositories, use the following command:

```
> sudo zypper repos
```

This automatically adds all the repositories to the container. For each repository added to the system, a new file is 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, run the command **`zypper ref -s`**. Including these files in a container image does not pose any security risk.

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

Now you can create a custom container image by using a `Dockerfile` as described in *Section 19.2.2, "Creating a custom image for SLE 12 SP5 and later"*.

After you have edited the `Dockerfile`, build the image by running the following command in the directory where the `Dockerfile` resides:

```
> podman build .
```

For more information about `podman build` options, see the official Podman documentation (https://docs.podman.io/en/latest/markdown/podman-build.1.html).

19.2.2 Creating a custom image for SLE 12 SP5 and later

The following `Dockerfile` creates a simple container image based on SUSE Linux Enterprise Server 15:

```
FROM registry.suse.com/suse/sle15
    RUN zypper ref -s && zypper -n in vim && zypper -n clean
```

When the Podman host machine is registered with an internal RMT server, the image requires the SSL certificate used by RMT:

```
FROM registry.suse.com/suse/sle15
    # Import the crt file of our private SMT server
    ADD http://smt./smt.crt /etc/pki/trust/anchors/smt.crt
    RUN update-ca-certificates && \
        zypper ref -s && \
        zypper -n in vim && \
        zypper -n clean
```

If you wish to add SLE extensions and modules to your images, refer to *Section 13.3.3, "Adding modules into the container or container Image"*.

19.2.3 Building container images in on-demand SLE instances in the public cloud

Building container images on SLE instances that were launched as on-demand or pay-as-you-go instances on a public cloud (AWS, GCE or Azure) requires additional steps. To install packages and updates, the on-demand public cloud instances are connected to the update infrastructure. This infrastructure is based on RMT servers operated by SUSE on public cloud providers.

Therefore, your machines need to locate the required services and authenticate with them. This can be done using the `containerbuild-regionsrv` service. This service is available in the public cloud images provided through the marketplaces of public cloud providers. Before building an image, this service must be started on the public cloud instance by running the following command:

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

To start it automatically on system boot, enable it:

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

The Zypper plug-ins provided by the SLE base images connect to this service and retrieve authentication details and information about which update server to talk to. For this to work, the container has to be built with host networking enabled, for example:

```
> podman build --network host build-directory/
```

Since update infrastructure in the public clouds is based upon RMT, the restrictions on building SLE images for SLE versions different from the SLE version of the host apply as well (see *Note: Building images on systems registered with RMT*).

20 Creating application container images

Applications that are suitable for running inside containers include daemons, Web servers, and applications that expose IP ports for communications. You can use Podman to automate the building and deployment processes by performing the build process inside a container, building an image, and then deploying containers based on the image.

Running an application inside a container has the following advantages:

- The image with the application is portable across servers running different Linux host distributions and versions.
- You can share the image of the application using a repository.
- You can use different versions of software in the container and on the host system, without creating dependency issues.
- You can run multiple instances of the same application that are independent from each other.

Using Podman to build applications has the following advantages:

- You can prepare an image of the complete build environment.
- The application can run in the same environment it was built in.
- Developers can test their code in the same environment as used in production.

The following section provides examples and recommendations on creating container images for applications. Before proceeding, make sure that you have activated your SUSE Linux Enterprise Server base image as described in *Section 19.1, “Pulling base SUSE Linux Enterprise Server images”*.

20.1 Running an application with specific package versions

If your application needs a version of a package different from the package installed on the system, you can create a container image that includes the package version the application requires. The following example Dockerfile allows building an image based on an up-to-date version of SUSE Linux Enterprise Server with an older version of the example package:

```
FROM registry.suse.com/suse/sle15
    LABEL maintainer=EXAMPLEUSER_PLAIN
    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"]
```

Build the image by running the following command in the directory that the Dockerfile resides in:

```
> podman build --tag tux_application:latest .
```

The Dockerfile example shown above performs the following operations during the image build process:

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

After a successful build of the tux_application image, you can start a container based on the new image using the following command:

```
> podman run -it --name application_instance tux_application:latest
```

Keep in mind that after closing the application, the container exits as well.

20.2 Running an application with a specific configuration

To run an instance using a different configuration, create a derived image and include the additional configuration with it. In the example below, an application called *example* is configured using the file `/etc/example/configuration_example`:

```
FROM registry.suse.com/suse/sle15 ❶
    RUN zypper ref && zypper --non-interactive in example ❷
    ENV BACKUP=/backup ❸
    RUN mkdir -p $BACKUP ❹
    COPY configuration_example /etc/example/ ❺
    ENTRYPOINT ["/etc/bin/example"] ❻
```

The above example `Dockerfile` performs the following operations:

❶ Pulls the `sle15` base image as described in *Section 19.1, "Pulling base SUSE Linux Enterprise Server images"*.

❷ Refreshes repositories and installations of the *example*.

❸ Sets a `BACKUP` environment variable (the variable persists to containers started from the image). You can always overwrite the value of the variable while running the container by specifying a new value.

❹ Creates the directory `/backup`.

❺ Copies the `configuration_example` to the image.

❻ Runs the `example` application.

20.3 Sharing data between an application and the host system

Podman allows sharing data between host and a container by using volumes. You can specify a mount point directly in the `Dockerfile`. However, you cannot specify a directory on the host system in the `Dockerfile`, as the directory may not be accessible at build time. Find the mounted directory under `/var/lib/docker/volumes/` on the host system.

Note: Discarding changes to the directory to be shared

After you specify a mount point by using the `VOLUME` instruction, all changes made to the directory with the **`RUN`** instruction are discarded. After the mount point is specified, the volume becomes a part of a temporary container, which is removed after a successful build. This means that for certain actions to take effect, they must be performed **before** specifying a mount point. For example, if you need to change permissions, do this before you specify the directory as a mount point in the `Dockerfile`.

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

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

Note

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

The following `Dockerfile` example builds an image containing a Web server that reads Web content from the host's file system:

```
FROM registry.suse.com/suse/sles12sp4
    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 the entire configuration to the image. The `data` directory is owned by the *admin* user and is used as a mount point to store Web pages.

20.4 Applications running in the background

If your application needs to run in the background as a daemon, or as an application exposing ports for communication, you can run the container in the background.

An example `Dockerfile` for an application exposing a port is as follows:

```
FROM registry.suse.com/suse/sle15 ❶
    LABEL maintainer=EXAMPLEUSER_PLAIN ❷
    ADD etc/ /etc/zypp/ ❸
```

```
RUN zypper refs && zypper refresh ❹
RUN zypper --non-interactive in apache2 ❺
RUN echo "The Web server is running" > /srv/www/htdocs/test.html ❻
# COPY data/* /srv/www/htdocs/ ❼
EXPOSE 80 ❽
ENTRYPOINT ["/usr/sbin/httpd"]
CMD ["-D", "FOREGROUND"]
```

❶ Pull the base image as described in *Section 19.1, “Pulling base SUSE Linux Enterprise Server images”*.

❷ Maintainer of the image (optional).

❸ The repositories and service files to be copied to `/etc/zypp/repos.d` and `/etc/zypp/services.d`. This makes them available on the host in the container.

❹ Command to refresh repositories and services.

❺ Command to install Apache2.

❻ Test line for debugging purposes. This line can be removed if everything works as expected.

❼ A **`COPY`** instruction to copy data from the host system to the directory in the container used by the server. The leading hash character (`#`) marks this line as a comment; it is not executed.

❽ The exposed port for the Apache Web server.

Note

To use port 80, make sure there is no other server software running on this port on the host.

To use the container, proceed as follows:

1. Prepare the host system for the build process.

   a. Make sure the host system is subscribed to the Server Applications Module of SUSE Linux Enterprise Server. To view installed modules or install additional modules, open YaST and select Add System Extensions or Modules.

   b. Make sure the SLE images from the SUSE Registry are installed as described in *Section 19.1, “Pulling base SUSE Linux Enterprise Server images”*.

   c. Save the `Dockerfile` in the `docker` directory.

d. Within the container, you need access to software repositories and services that are registered on the host. To make them available, copy repositories and service files from the host to the `docker/etc` directory:

```
> cd docker
    > mkdir etc
    > sudo cp -a /etc/zypp/{repos.d,services.d} etc/
```

Instead of copying all repository and service files, you can also copy only the subset that is required by the container.

e. Add Web site data (such as HTML files) into the `docker/data` directory. The contents of this directory are copied to the container image and are thus published by the Web server.

2. Build the container. Set a tag for your image with the `-t` option (in the command below, it is *EXAMPLEUSER_PLAIN*):

```
> docker build -t EXAMPLEUSER_PLAIN/apache2 .
```

Docker Open Source Engine executes the instructions provided in the `Dockerfile`: pull the base image, copy content, refresh repositories, install the Apache2, etc.

3. Start a container instance from the image created in the previous step:

```
> docker run --detach --interactive --tty EXAMPLEUSER_PLAIN/apache2
```

Docker Open Source Engine returns the container ID, for example:

```
7bd674eb196d330d50f8a3cfc2bc61a243a4a535390767250b11a7886134ab93
```

4. Point a browser at http://localhost:80/test.html. You should see the message *The Web server is running*.

5. To see an overview of running containers, run the **`docker ps --latest`** command:

```
> docker ps --latest
    CONTAINER ID        IMAGE               COMMAND                  [...]
    7bd674eb196d
    tux/apache2         "/usr/sbin/httpd -..."   [...]
```

To stop and delete the container, run the following command:

```
> docker rm --force 7bd674eb196d
```

You can use the resulting container to serve your data with the Apache2 Web server by following these steps:

1. In the `Dockerfile` :
   - In the example `Dockerfile`, comment the line that starts with `RUN echo` by adding a `#` character at its beginning.
   - In the example `Dockerfile`, uncomment the line starting with `COPY` by removing the leading `#` character.
2. Rebuild the image.
3. Run the image in detached mode:

   ```
   > docker run --detach --interactive --tty EXAMPLEUSER_PLAIN/apache2
   ```

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

   ```
   e43fff4ae9832ecdb7677c058a73039d7610c32145a1d9b6ad0a4ed52b5c4dc7
   ```

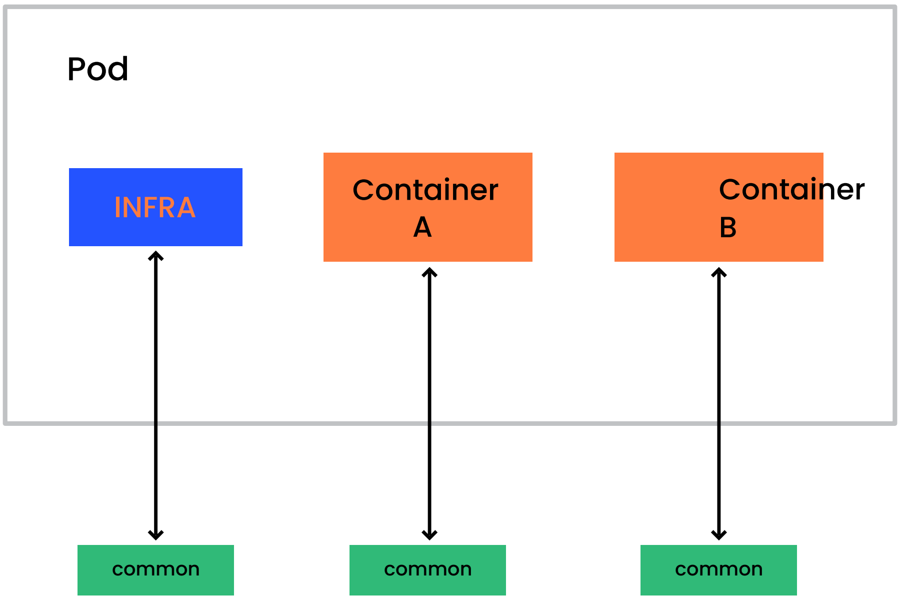
To view the published data, point a browser at http://localhost:80/test.html.

To avoid copying Web site data into the container, share a directory of the host with the container. For more information, see https://docs.docker.com/storage/volumes/.

21 Container orchestration

In a production environment, you are likely to manage multiple containers. To work with multiple containers, you have to group the containers into a pod that provides a specification for deploying and running the containers as well as allowing them to share storage and network resources. In other words, a pod encapsulates an application composed of multiple containers into a single unit. The concept of pod was introduced by Kubernetes (https://kubernetes.io/docs/concepts/workloads/pods/). Podman uses the same definition as Kubernetes.

Usually, containers within a pod can communicate directly with each other. Each pod contains an infrastructure container (INFRA), whose purpose is to hold the namespace. INFRA also enables Podman to add other containers to the pod. Port bindings, cgroup-parent values and kernel namespaces are all assigned to the infrastructure container. Therefore, you cannot change any of these values later.



Each container in a pod has its own instance of a monitoring program. The monitoring program watches the container's process and if the container dies, the monitoring program saves its exit code. The program also holds open the tty interface for the particular container. The monitoring program enables you to run containers in the detached mode when Podman exits, because this program continues to run and enables you to attach tty later.

21.1 Single container host with Podman

`podman pod` is the command-line tool for creating, removing, querying and inspecting pods. You can check all the subcommands of **`podman pod`** in the official upstream documentation (https://docs.podman.io/en/latest/markdown/podman-pod.1.html).

`podman pod create` creates a pod with a random name. You can use the `--name` parameter to assign the desired name to a pod.

```
> podman pod create
```

```
    344940492c00b6a19ececbc5b109351bf0a3b8b19b3c279a097da7a653c592d0
```

You can list our pods using the **podman pod list** command:

```
> podman pod list
    POD ID        NAME            STATUS      CREATED         INFRA ID      # OF CONTAINERS
    344940492c00  suspicious_curie  Created     2 minutes ago  617d7e3ce399  1
```

You can also list all containers with the pods they are associated with:

```
> podman ps -a --pod
    CONTAINER ID  IMAGE                     COMMAND     CREATED         STATUS      PORTS
  NAMES               POD ID        PODNAME
    617d7e3ce399  localhost/podman-pause:4.3.1-1669118400              5 minutes ago
 Created                 344940492c00-infra  344940492c00  suspicious_curie
```

The created pod has an infra container identified by the `localhost/podman-pause:4.x` name. The purpose of this container is to reserve the namespaces associated with the pod and allow Podman to add other containers to the pod.

Using the **podman run --pod** command, you can run a container and add it to the desired pod. For example, the command below runs a container based on the `suse/sle15` image and adds the container to the `suspicious_curie` pod:

```
> podman run -d --pod suspicious_curie registry.suse.com/bci/bci-base sleep 1h
    8f5af62a7c385bbd1a3a5cc3a53a8d0f8cf942adc26a065960d4232fcc93ac98
```

Warning

If this command shows the following warning, refer to Using container-suseconnect on non-SLE hosts or with Podman, Buildah, and nerdctl (https://documentation.suse.com/container/all/single-html/SLES-container/#sec-bci-suseconnect-podman-buildah-nerdctl):

```
WARN[0005] Path "/etc/SUSEConnect" from "/etc/containers/mounts.conf" doesn't exist,
 skipping
    WARN[0005] Failed to mount subscriptions, skipping entry in /etc/containers/
mounts.conf: open /etc/zypp/credentials.d/SCCcredentials: permission denied
```

The command above adds a container that sleeps for 60 minutes and then exits. Run the **podman ps -a --pod** command again and you should see that the pod now has two containers.

You can also check the command **podman pod ps**:

```
> podman pod ps
    POD ID        NAME            STATUS      CREATED          INFRA ID      # OF CONTAINERS
```

```
   344940492c00  suspicious_curie Running     21 minutes ago  617d7e3ce399  2
```

To stop our newly created container named `objective_jemison`

```
> podman ps -a --pod
    CONTAINER ID  IMAGE                                     COMMAND     CREATED         STATUS
            PORTS       NAMES                POD ID        PODNAME
    617d7e3ce399  localhost/podman-pause:4.3.1-1669118400
    14 minutes ago  Up 4 minutes ago              344940492c00-infra  344940492c00
  suspicious_curie 8f5af62a7c38  registry.suse.com/bci/bci-base:latest  sleep 1h
   4 minutes ago   Up 4 minutes ago              objective_jemison   344940492c00
  suspicious_curie
    > podman stop objective_jemison
    objective_jemison
    > podman pod ps
    POD ID        NAME          STATUS      CREATED         INFRA ID      # OF CONTAINERS
    344940492c00  suspicious_curie  Degraded    25 minutes ago  617d7e3ce399  2
    > podman ps -a --pod
    CONTAINER ID  IMAGE                                     COMMAND     CREATED         STATUS
                       PORTS       NAMES                POD ID        PODNAME
    617d7e3ce399  localhost/podman-pause:4.3.1-1669118400                            25
 minutes ago  Up 15 minutes ago                         344940492c00-infra  344940492c00
  suspicious_curie 8f5af62a7c38  registry.suse.com/bci/bci-base:latest  sleep 1h    15
 minutes ago  Exited (137) 14 seconds ago              objective_jemison   344940492c00
  suspicious_curie
```

You can also stop the pod and all of its containers using **`podman pod stop`**:

```
# podman pod stop suspicious_curie
    344940492c00b6a19ececbc5b109351bf0a3b8b19b3c279a097da7a653c592d0
    > podman ps -ap
    CONTAINER ID  IMAGE                                     COMMAND     CREATED         STATUS
                      PORTS       NAMES                POD ID        PODNAME
    617d7e3ce399  localhost/podman-pause:4.3.1-1669118400                            29
 minutes ago  Exited (0) 7 seconds ago               344940492c00-infra  344940492c00
  suspicious_curie 8f5af62a7c38  registry.suse.com/bci/bci-base:latest  sleep 1h    19
 minutes ago  Exited (137) 3 minutes ago             objective_jemison   344940492c00
  suspicious_curie
```

You can also start and restart everything with **`sudo podman start`** *`CONTAINER_NAME`*, **`podman pod start`** *`POD_NAME`* or **`podman pod restart`** *`POD_NAME`*.

There are two ways to remove pods. You can use the **`podman pod rm`** command to remove one or more pods. Alternatively, you can remove all stopped pods using the **`podman pod prune`** command. To remove a pod or several pods, run the **`podman pod rm`** command as follows:

```
> podman pod rm POD
```

POD can be a pod name or a pod ID. To remove all currently stopped pods, use the **`podman pod prune`** command. Make sure that all stopped pods are intended to be removed before you run the **`podman pod prune`** command, otherwise there is a risk of removing pods that are still in use.

A container runtime makes it easy to launch an application distributed as a single container. But things get more complicated when you need to run applications consisting of multiple containers, or when it's necessary to start the applications automatically on system boot and restart them after they crash. While container orchestration tools like Kubernetes are designed for that exact purpose, they are intended to be used for highly distributed and scalable systems with hundreds of nodes, and not for a single machine. systemd and Podman are much better suited for the single-machine scenario, as they do not add another layer of complexity to your existing setup.

Podman supports creating systemd unit files with the **`podman generate systemd`** subcommand. The subcommand creates a systemd unit file, making it possible to control a container or pod via systemd. Using the unit file, you can launch a container or pod on boot, automatically restart it if a failure occurs, and keep its logs in journald.

The following example uses a simple NGINX container:

```
> podman run -d --name web -p 8080:80 docker.io/nginx
    c0148d8476418a2da938a711542c55efc09e4119909aea70e287465c6fb51618
```

Generating a systemd unit for the container can be done as follows:

```
> podman generate systemd --name --new web
    # container-web.service
    # autogenerated by Podman 4.2.0
    # Tue Sep 13 10:58:54 CEST 2022

    [Unit]
    Description=Podman container-web.service
    Documentation=man:podman-generate-systemd(1)
    Wants=network-online.target
    After=network-online.target
    RequiresMountsFor=%t/containers

    [Service]
    Environment=PODMAN_SYSTEMD_UNIT=%n
    Restart=on-failure
    TimeoutStopSec=70
    ExecStartPre=/bin/rm -f %t/%n.ctr-id
    ExecStart=/usr/bin/podman run \
            --cidfile=%t/%n.ctr-id \
            --cgroups=no-conmon \
            --rm \
```

```
            --sdnotify=conmon \
            --replace \
            -d \
            --name web \
            -p 8080:80 docker.io/nginx
    ExecStop=/usr/bin/podman stop --ignore --cidfile=%t/%n.ctr-id
    ExecStopPost=/usr/bin/podman rm -f --ignore --cidfile=%t/%n.ctr-id
    Type=notify
    NotifyAccess=all

    [Install]
    WantedBy=default.target
```

Podman outputs a unit file to the console that can be put either into the user unit systemd directories (`~/.config/systemd/user/` or `/etc/systemd/user/`) or into the system unit systemd directory (`/etc/systemd/system`) and control the container via systemd. The `--new` flag instructs Podman to recreate the container on a restart. This ensures that the systemd unit is self-contained, and it does not depend on any external state. The `--name` flag allows you to assign a user-friendly name to the container: without it, Podman uses container IDs instead of their names.

To control the container as a user unit, proceed as follows:

```
> podman generate systemd --name --new --files web
    /home/user/container-web.service
    > mv container-web.service ~/.config/systemd/user/
    > systemctl --user daemon-reload
```

Now the container can be started with **`systemctl --user start container-web`**:

```
> systemctl --user start container-web
    > systemctl --user is-active container-web.service
    active
```

Run the **`podman ps`** command to see the list of all running containers:

```
> podman ps
    CONTAINER ID  IMAGE                           COMMAND               CREATED
 STATUS             PORTS                 NAMES
    af92743971d2  docker.io/library/nginx:latest  nginx -g daemon o...  15 minutes ago
 Up 15 minutes ago  0.0.0.0:8080->80/tcp  web
```

One of the benefits of managing the container via systemd is the ability to automatically restart the container if it crashes. You can simulate a crash by sending `SIGKILL` to the main process in the container:

```
> podman ps
```

```
   CONTAINER ID  IMAGE                           COMMAND               CREATED
  STATUS                PORTS                 NAMES
   4c89582fa9cb  docker.io/library/nginx:latest  nginx -g daemon o...  About a minute
ago  Up About a minute ago  0.0.0.0:8080->80/tcp  web

   > kill -9 $(podman inspect --format "{{.State.Pid}}" web)

   > podman ps
   CONTAINER ID  IMAGE                           COMMAND               CREATED
STATUS            PORTS                 NAMES
   0b5be4493251  docker.io/library/nginx:latest  nginx -g daemon o...  4 seconds ago  Up
4 seconds ago  0.0.0.0:8080->80/tcp  web
```

Note that the container is **not** restarted when it is stopped gracefully, for example, via **`podman stop web`**. To always restart it, add the flag `--restart-policy=always` to **`podman generate systemd`**.

21.2 Updating container images

Using the described approach means that the container image is never updated. You can solve this problem by adding the `--pull=always` flag to the `ExecStart=` entry in the unit file. But be aware that this increases the start-up time of the container and updates the image on **every** restart. The latter also means that a container image update can make the container unavailable outside of a scheduled maintenance window due to a newly introduced bug.

The **`auto-update`** (https://docs.podman.io/en/latest/markdown/podman-auto-update.1.html) subcommand in Podman provides a possible solution. Add the label `io.containers.autoupdate=registry` to a container to make Podman pull a new version of the container image from the registry when running **`podman auto-update`**. This makes it possible to update all container images with a single command at a desired time, and without increasing the start-up time of the systemd units.

The auto-update feature can be enabled by adding the line `--label "io.containers.autoupdate=registry" \` to the `ExecStart=` entry of the container's systemd unit file. For the NGINX example, modify `~/.config/systemd/user/container-web.service` as follows:

```
ExecStart=/usr/bin/podman run \
            --cidfile=%t/%n.ctr-id \
            --cgroups=no-conmon \
            --rm \
            --sdnotify=conmon \
            --replace \
            -d \
```

```
            --name web \
            --label "io.containers.autoupdate=registry" \
            -p 8080:80 docker.io/nginx
```

After reloading the daemons and restarting the container, perform a dry run of the update (it will most likely not report any updates):

```
> podman auto-update --dry-run
    UNIT                   CONTAINER           IMAGE            POLICY      UPDATED
    container-web.service  87d263489307 (web)  docker.io/nginx  registry    false
```

It is good practice to have external testing in place to make sure that image updates are generally safe to be deployed. If you are confident of the quality of our container image, you can let Podman automatically apply image updates periodically by enabling the `podman-auto-update.timer`:

```
# only for the current user
    > systemctl --user enable podman-auto-update.timer
    Created symlink /home/user/.config/systemd/user/timers.target.wants/podman-auto-
update.timer → /usr/lib/systemd/user/podman-auto-update.timer.
    # or as root
    > sudo systemctl enable podman-auto-update.timer
    Created symlink /etc/systemd/system/timers.target.wants/podman-auto-update.timer → /
usr/lib/systemd/system/podman-auto-update.timer.
```

21.3 Managing multiple containers

Certain applications rely on more than one container to function, for example, a web front-end, a back-end server and a database. Docker compose (https://docs.docker.com/compose/) is popular tool for deploying multi-container applications on a single machine. While Podman does not support the **`compose`** command natively, in most cases compose files can be ported to a Podman pod and multiple containers.

The following example deploys a Drupal and PostgreSQL container in a single pod and manages these via systemd units. First, create a new pod that exposes the Drupal web interface:

```
> podman pod create -p 8080:80 --name drupal
    736cab072c49e68ad368ba819e9117be13ef8fa048a2eb88736b5968b3a19a64
```

Once the pod has been created, launch the Drupal front-end and the PostgreSQL database inside it:

```
> podman run -d --name drupal-frontend --pod drupal docker.io/drupal
```

```
    ffd2fbd6d445e63fb0c28abb8d25ced78f819211d3bce9d6174fe4912d89f0ca

    > podman run -d --name drupal-pg --pod drupal \
          -e POSTGRES_DB=drupal \
          -e POSTGRES_USER=user \
          -e POSTGRES_PASSWORD=pass \
          docker.io/postgres:11
    a4dc31b24000780d9ffd81a486d0d144c47c3adfbecf0f7effee24a00273fcde
```

This results in three running containers: the Drupal web interface, the PostgreSQL database and the pod's infrastructure container.

```
> podman ps
    CONTAINER ID  IMAGE                                    COMMAND               CREATED
            STATUS                 PORTS                  NAMES
    2948fa1476c6  localhost/podman-pause:4.2.0-1660228937                        2
 minutes ago       Up About a minute ago  0.0.0.0:8080->80/tcp  736cab072c49-infra
    ffd2fbd6d445  docker.io/library/drupal:latest          apache2-foregroun...  About a
 minute ago  Up About a minute ago  0.0.0.0:8080->80/tcp  drupal-frontend
    a4dc31b24000  docker.io/library/postgres:11            postgres              40
 seconds ago      Up 41 seconds ago      0.0.0.0:8080->80/tcp  drupal-pg
```

Creating a systemd unit for the pod is done similarly to a single container:

```
> podman generate systemd --name --new --files drupal
    /home/user/pod-drupal.service
    /home/user/container-drupal-frontend.service
    /home/user/container-drupal-pg.service
    > mv *service ~/.config/systemd/user/
    > systemctl daemon-reload --user
```

Since Podman is aware of which containers belong to the `drupal` pod and how their systemd units are called, it can correctly add the dependencies to the pod's unit file. This means that when you start or stop the pod, systemd ensures that all containers inside the pod are started or stopped automatically.

To check systemd's dependency handling, first stop the `drupal` pod and verify that no containers are currently running on the host:

```
> podman pod stop drupal
    736cab072c49e68ad368ba819e9117be13ef8fa048a2eb88736b5968b3a19a64
    > podman pod rm drupal
    736cab072c49e68ad368ba819e9117be13ef8fa048a2eb88736b5968b3a19a64
    > podman ps -a
    CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
```

Start the `drupal` pod via **`systemctl start --user pod-drupal.service`**, and systemd launches the containers inside the pod:

```
> systemctl start --user pod-drupal.service
    > podman ps
    CONTAINER ID  IMAGE                                    COMMAND               CREATED
       STATUS           PORTS                 NAMES
    d1589d3ac68b  localhost/podman-pause:4.2.0-1660228937                        5
 seconds ago  Up 5 seconds ago  0.0.0.0:8080->80/tcp  ca41b505bd13-infra
    a49bea53c20c  docker.io/library/postgres:11          postgres              4
 seconds ago  Up 5 seconds ago  0.0.0.0:8080->80/tcp  drupal-pg
    dc9dca018dad  docker.io/library/drupal:latest        apache2-foregroun...  4
 seconds ago  Up 5 seconds ago  0.0.0.0:8080->80/tcp  drupal-frontend
```

21.4 More on Podman

If you want to learn more about Podman and handling pod deployment, please check https://docs.podman.io/en/latest/ and https://github.com/containers/podman.

21.5 Multi-container host with Kubernetes

Kubernetes (https://kubernetes.io) is an open source container orchestration engine for automating deployment, scaling and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF (https://www.cncf.io/about)).

Kubernetes makes it possible for multiple machines (or servers or nodes) to work together and create a cluster that you can then interact with through APIs. We recommend using Rancher (https://ranchermanager.docs.rancher.com) for deploying Kubernetes clusters and managing applications running on top of them. A single Rancher setup can manage multiple Kubernetes clusters running anywhere: from bare-metal, to on-prem or cloud service providers.

For more information on Rancher, refer to the official Rancher documentation (https://ranchermanager.docs.rancher.com).

21.6 Lightweight Kubernetes (k3s)

K3s (https://k3s.io) ↗ is a lightweight CNCF-certified Kubernetes distribution built for IoT and Edge computing. Unlike Kubernetes, K3s is packaged as a single <60 MB binary and optimized for the Arm architecture.

For more info, refer to Introduction to K3s (https://www.suse.com/c/rancher_blog/introduction-to-k3s/) ↗ and How to install K3s and Rancher on SUSE Linux Enterprise Server (https://documentation.suse.com/trd/kubernetes/single-html/kubernetes_ri_rancher-k3s-sles/index.html#id-introduction) ↗.

22 Compatibility and support conditions

The term "support" refers to two distinct concepts: a) technical enablement of a feature or combination of, for example, host and container, and b) enterprise support as delivered by SUSE to SUSE customers. Enterprise support requires a subscription for SUSE products according to https://www.suse.com/products/terms_and_conditions.pdf ↗. Technical enablement is described below.

22.1 Support for SLES hosts

Consult the following support and compatibility matrix to make sure that the desired host system and container combination is compatible and supported.

TABLE 2: SUPPORT MATRIX

Host ↓ Container image →	SLES 12	SLES 15
SLES 12 SP5	✓	❋
SLES 15	✓	✓
SLE Micro	✓	✓
SUSE Liberty Linux	✓	✓

✓ Fully supported

❋ Limited support (see the Limited support note)

Important: Limited support note

SUSE provides limited support for SLES 15 GA-based containers running on SLES 12 SP5 hosts due to the fact that containerized applications can make system calls not available in the host's kernel. To avoid potential risks and compatibility problems, SUSE recommends using the same Service Pack release for both containers and hosts.

SLE BCIs support the following architectures: AMD64/Intel 64, AArch64, POWER and IBM Z. Container architecture must match the architecture of the host. Mismatching container and host scenarios are not supported.

In most scenarios, all SLE containers are expected to be interoperable if the application and its dependencies do not interact directly with kernel version-specific data structures and their derivatives (`ioctl`, `/proc`, `/sys`, `routing`, `iptables`, `nftables`, `BTF`, `(e)BPF`, etc.) or modules (KVM, OVS, SystemTap, etc.). Support for `ioctl` and access to `/proc` is limited to the most common scenarios needed by unprivileged users.

22.2 Support for non-SLES hosts

While SUSE-based containers are fully supported, issues in the host environment must be handled by the host environment vendor. SUSE supports components that are part of the SUSE base containers. Packages from SUSE repositories are also supported. Additional components and applications in the containers are not covered by SUSE support. A SLE subscription is required for building derived containers.

Containers based on SLES 12 SP5 and SLES 15 (all service packs) are supported according to their official lifecycles and the following table.

The following third-party container host platforms are supported.

TABLE 3: SUPPORT FOR NON-SLES HOSTS

Container host platform	Container runtime	Support status
Rancher Kubernetes Engine (RKE)	docker	✓
Rancher Kubernetes Engine 2 (RKE2)	containerd	✓

Container host platform	Container runtime	Support status
K3S	containerd	✓
Red Hat OpenShift	cri-o	✓
Microsoft Azure Kubernetes Service (AKS)	containerd	✓
Google Kubernetes Engine (GKE)	containerd	❋
Amazon Elastic Container Service for Kubernetes (EKS)	containerd	✓
IBM Hyper Protect Platform	docker/podman	✓

✓ Fully compatible and fully supported

❋ Workload specific: fully supported but compatibility depends on type of container (privileged or unprivileged) and on the application interactions (direct with kernel-version-specific data structures, kernel-version-specific modules, etc.)

Refer to the Rancher Support Matrix (https://www.suse.com/suse-rancher/support-matrix/all-supported-versions/) ↗ for more information regarding support for Rancher-related products.

22.3 Support plans

There are three guiding principles of SUSE container support.

1. The container image lifecycle follows the lifecycle of the related products.
   For example, SLES 15 SP4 container images follow the SLES 15 SP4 lifecycle.

2. Container release status also matches the status of the related product.
   For example, if SLES 15 SP4 is in Alpha, Beta, RC or GA stage, the related containers have the same release status.

3. Containers are built using the packages from the related products.

For example, SLES 15 SP4 container images are built using the same packages as the main SLES 15 SP4 release.

For further information, refer to the Product Support Lifecycle (https://www.suse.com/lifecycle) ↗ page and the documentation available for specific container images on SUSE Registry (https://registry.suse.com) ↗.

Container images can have different support status, and they can have limited support. Refer to the appropriate https://registry.suse.com ↗ page for the further information about a specific container image.

22.4 Containers and host environments support overview

The following support options are valid for SLES containers on SUSE host environments.

Containers and host environments delivered by SUSE are fully supported. This also applies to all products under support, including both general support and Long Term Service Pack Support (https://www.suse.com/products/long-term-service-pack-support/) ↗ (LTSS).

Partner containers and host environments with a joint engineering collaboration agreement are fully supported. This applies to both the container and host environment as well as all products under support (both general and LTSS) covered by the agreement.

While SUSE-based containers are fully supported, issues in the host environment must be handled by the host environment vendor. SUSE supports components that come from the SUSE base containers. Packages from SUSE repositories are also supported. Additional components and applications in the containers are not covered by SUSE support. No subscription is required for building derived containers based on the content of the SLE BCIs or the SLE_BCI Repository. To build containers that include packages from the full SLE universe, you need a subscription that grants you access to the repositories containing these packages.

Any container and host environment not mentioned above has limited support. Details can be discussed with the SUSE Support Team responsible for triaging the issue and recommending alternative solutions. In any other case, issues in the host environment must be handled by the host environment vendor.

22.5 Technology previews

Container images labeled as *Tech Preview* are provided by SUSE to give you an opportunity to test new technologies within your environment and share your feedback. If you test a technology preview, contact your SUSE representative to share your experiences and use cases. Your input is helpful for future development.

Technology previews come with the following limitations:

- Technology previews may be functionally incomplete, unstable, and not suitable for production use.
- Technology previews are not supported.
- Technology previews may be available only for specific hardware architectures.
- Specifics and functionality of technology previews are subject to change. As a result, upgrading to subsequent releases of a technology preview may not be possible and may require a fresh installation.
- Technology previews can be canceled at any time. For example, this might happen if SUSE discovers that a preview does not meet the customer or market needs, or that it does not comply with enterprise standards. SUSE does not commit to providing a supported version of such technologies in the future.

Container images are labeled as *Tech Preview* and are marked as such at registry.suse.com (https://registry.suse.com) ↗. Additionally, container images that are technology previews include the `com.suse.supportlevel="techpreview"` label in the container image metadata. You can check whether the metadata includes the label using the `docker inspect` command, or an appropriate command in other container runtimes.

22.6 Test platform and environments

SLE Base Container Images are tested with the following platforms and environments:

- Supported openSUSE Leap (AMD64/Intel 64 only)
- All supported SUSE Linux Enterprise Server versions and hardware architectures
- SUSE Liberty versions 8 and 9 (AMD64/Intel 64 only)

- Current Ubuntu LTS (AMD64/Intel 64 only)
- CentOS stream (only)
- All major public cloud Kubernetes platforms, including Microsoft Azure Kubernetes Service, Google Kubernetes Engine, and Amazon Web Services

SLE Base Container Images are tested using Podman and Docker. SLE Base Container Images that use FIPS-certified crypto libraries are tested on FIPS-certified platforms.

23 Troubleshooting

23.1 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.

23.1.1 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):

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

```
> 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 IMAGE2 was created from IMAGE1.

```
> sudo container-diff diff daemon://IMAGE1 daemon://IMAGE2 --type=history
```

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

24 Terminology

Container

A *container* is a running instance based on a particular container image. Each *container* can be distinguished by a unique container ID.

Control groups

Control groups, also called `cgroups` , are a Linux kernel feature that allows aggregating or partitioning tasks (processes) and all their children into hierarchically-organized groups, to manage their resource limits.

Docker Open Source Engine

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

- **Daemon:.** + The server side of Docker Open Source Engine, which manages all Docker objects (images, containers, network connections used by containers, etc.).
- **REST API:.** + Applications can use this API to communicate directly with the daemon.
- **CLI client:.** + 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.

Dockerfile

A *Dockerfile* provides instructions on how to build a container image. Docker Open Source Engine reads instructions in the *Dockerfile* and builds a new image according to the instructions.

Image

An *image* is a read-only template used to create a *container*. A Docker image is made of 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 (https://docs.docker.com/engine/reference/glossary#image).

Container image

A *container image* is an unchangeable, static file that includes executable code so it can run an isolated process on IT infrastructure. The image is comprised of system libraries, system tools, and other platform settings a program needs to run on a containerization platform. A container image is compiled from file system layers built on top of a parent or base image.

Base image

A *base image* is an image that does not have a parent image. In a Dockerfile, a base image is identified by the `FROM scratch` directive.

Parent image

The image that serves as the basis for another container image. In other words, if an image is not a base image, it is derived from a parent image. In a Dockerfile, the `FROM` directive is pointing to the parent image. Most Docker containers are created using parent images.

Namespaces

Docker Open Source Engine uses Linux *namespaces* for its containers, which 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 automatically manage the containers. The act of automatic container management is called container orchestration and is typically handled by Kubernetes.

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

- public registry: Any (usually registered) user can download and use images. A typical example of a public registry is Docker Hub (https://hub.docker.com/).
- private registry: Access is restricted to particular users, or from a particular private network.

Repository
: A *repository* is storage for images in a *registry*.

25 Improving the documentation

Feedback and contributions to this documentation can be submitted using any of the following options.

Service requests and support
: For services and support options available for your product, see https://www.suse.com/support/. To open a service request, you need a SUSE subscription registered at SUSE Customer Center. Go to https://scc.suse.com/support/requests, log in, and click *Create New*.

Bug reports
: Report issues with the documentation at https://bugzilla.suse.com/ (this requires a Bugzilla account). To simplify the process, use the *Report an issue link* in the HTML version of this document. Point the cursor to the desired sentence, and click *Report an issue* in the *Give feedback* section of the right-side navigation panel. This automatically selects the correct product and category in Bugzilla and adds a link to the current section. You can now write your bug report.

Contributions
: To contribute to this documentation, use the *Edit source document* link in the HTML version of this document (this requires a GitHub account). Point the cursor to the desired sentence, and click *Report an issue* in the *Give feedback* section of the right-side navigation panel. This takes you to the source code on GitHub, where you can open a pull request.

Note: Edit source document only available for English

The *Edit source* document links are only available for the English version of each document. For all other languages, use the *Report an issue link* as described above.

For more information about the documentation environment used for this documentation, refer to the repository's *README* file at https://github.com/SUSE/doc-unversioned/blob/main/README.adoc

Mail

You can also report errors and send feedback concerning the documentation to `doc-team@suse.com`. Include the document title, the product version, and the publication date of the document. Additionally, include the relevant section number and title (or provide the URL), and provide a concise description of the problem.

26 Legal Notice

27 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, Tex-info 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 non-commercially, 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.