SUSE

# 容器指南

以下指南介绍了 SUSE Linux Enterprise Server 容器生态系统。由于容器是一项不断发展的技术，本指南会定期更新、补充和改进，以反映最新的技术进展。

出版日期：2024-09-29

## 目录

# 1 Linux 容器简介

Linux 容器提供了轻量级虚拟化方法，可在单台主机上同时运行多个隔离的虚拟环境。此技术基于用于进程隔离的 Linux 内核名称空间，以及用于资源（CPU、内存、磁盘 I/O、网络等）管理的内核控制组 (cgroup)。

与 Xen 和 KVM 通过虚拟化层执行完整的 Guest 操作系统不同，Linux 容器共享并直接使用主机操作系统内核。

**使用容器的优势**

- **大小**：容器映像应该只包含运行应用程序所需的内容，而虚拟机则包含整个操作系统。
- **性能**：容器提供接近本机的性能，因为与虚拟化和模拟相比，其内核开销更低。
- **安全性**：使用容器可以将应用程序隔离成自给性单元，使其与主机系统的其余组件分开。
- **资源管理**：可以精细控制容器内的 CPU、内存、磁盘 I/O、网络接口等组件（通过 cgroup）。
- **灵活性**：容器映像包含运行应用程序所需的所有必要库、依赖项和文件，因此可以轻松地在多个主机上进行开发和部署。

**容器的限制**

- 容器共享主机系统的内核，因此容器必须使用主机提供的特定内核版本。
- 在 Linux 主机上只能容器化基于 Linux 的应用程序。
- 容器封装特定体系结构（例如 AMD64/Intel 64 或 AArch64）的二进制文件。因此，如果不采用模拟方式，为 AMD64/Intel 64 创建的容器只能在 AMD64/Intel 64 系统主机上运行。
- 容器本身并不比在容器外部执行二进制文件更安全，容器的整体安全性取决于主机系统。虽然可以通过 AppArmor 或 SELinux 配置文件来保护容器化应用程序，但容器安全性的实现需要部署可确保容器基础架构和应用程序安全性的工具和策略。

## 1.1 Podman 的重要概念和简介

尽管 Docker 开源引擎是用于处理映像和容器的流行解决方案，但具备诸多优势的 Podman 可以直接取代 Docker。有关 Podman 的详细信息，请参见第 14 节 “Podman 概述”。本章将会简要介绍重要概念，以及创建容器映像并使用 Podman 来运行容器的基本过程。

基本的 Podman 工作流程如下：


在本地计算机或云服务中运行容器通常涉及以下步骤：

1. 提取基本映像，方法是将其从注册表中提取到本地计算机。
2. 创建一个 `Dockerfile`，并使用它来基于基本映像构建一个自定义映像。
3. 使用创建的映像启动一个或多个容器。

要运行容器，需要有一个映像。映像包含运行应用程序所需的全部依赖项。例如，SLE BCI-Base 映像包含选择了极少量软件包的 SLE 发行套件。

尽管可以从头开始创建映像，但很少有应用程序能够在这种空环境中正常运行。因此，在大多数情况下，使用现有的基本映像更为可行。基本映像没有父项，也就是说，它不以其他映像为基础。

尽管您可以使用基本映像来运行容器，但基本映像的主要用途是充当创建自定义映像的基础，自定义映像可以运行包含特定应用程序、服务器、服务等的容器。

基本映像和自定义映像都可以通过称作“注册表”的映像储存库获得。除非明确指定了注册表，否则 Podman 将从 openSUSE (https://registry.opensuse.org) 和 Docker Hub (https://docker.io) 注册表中提取映像。尽管您可以手动提取基本映像，但 Podman 可以在构建自定义映像时自动提取基本映像。

要构建自定义映像，必须创建一个名为 `Containerfile` 或 `Dockerfile` 的特殊文件，其中包含构建指令。例如，`Dockerfile` 可以包含用于更新系统软件、安装所需应用程序、打开特定网络端口、运行命令等的指令


您不仅可以基于基本映像构建映像，而且可以基于自定义映像构建映像。因此您可以构建由多个层组成的映像。有关详细信息，请参考第 19 节 “创建自定义容器映像”。

# 2 用于构建映像和管理容器的工具

下面所述的所有工具（Open Build Service (https://openbuildservice.org) 除外）都包含在 **SUSE Linux Enterprise Server Containers 模块**中。您可以在 SUSE Customer Center (https://scc.suse.com/packages) 的 **Containers 模块**中查看完整的软件包列表。

## 2.1 SUSE 注册表

https://registry.suse.com 是 SLE 基本容器映像的官方来源。其中包含已经过测试、更新和认证的 SLE 基本容器映像。SUSE 注册表中的所有映像会定期重构建，以包含更新和修复。SUSE 注册表的 Web 用户界面列出了一部分可用映像：基本容器映像、开发堆栈容器映像、应用程序容器映像、SUSE Linux Enterprise Server 映像，以及不再享受一般支持的版本。该 Web 用户界面还提供了每个映像的其他信息，包括发布日期、支持级别、大小、映像摘要以及映像中的软件包。

## 2.2 Docker

Docker 是用于创建和管理容器的系统。其核心是 Docker Open Source Engine— (https://github.com/moby/moby)，这是一种轻量级虚拟化解决方案，可在单台主机上同时运行多个容器。Docker 容器是使用 Dockerfile 构建的。

## 2.3 Podman

Podman (https://podman.io) 表示 Pod Manager 工具。它是一个无守护程序的容器引擎，用于在 Linux 系统上开发、管理和运行 Open Container Initiative (OCI) (https://opencontainers.org/about/overview/) 容器，可以直接取代 Docker。Podman 是推荐用于 SLES 的容器运行时。有关 Podman 的总体介绍，请参见第 14 节 “Podman 概述”。

## 2.4 Buildah

Buildah (https://buildah.io) 是用于构建 OCI 容器映像的实用程序。它是 Podman 的补充工具。事实上，**`podman build`** 命令使用 Buildah 来执行容器映像构建。借助 Buildah，您可以从头开始、基于现有映像以及使用 Dockerfile 来构建映像。使用 Buildah 命令行工具和基于 OCI 的底层技术（例如 `containers/image` 和 `containers/storage`）构建的 OCI 映像具有可移植性，因此可以在 Docker 开源引擎环境中运行。有关安装和使用 Buildah 的信息，请参见第 18 节 “Buildah 概述”。

## 2.5 skopeo

skopeo (https://github.com/containers/skopeo) 是一个命令行实用程序，用于管理、检查容器映像和映像储存库并为其签名。skopeo 可用于检查远程与本地容器注册表中的容器和储存库。skopeo 可以在不同的存储后端之间复制容器映像。skopeo 包含在 SUSE Linux Enterprise Server 的 `Basesystem Module` 中。

## 2.6 Helm

Helm (https://helm.sh) 是 Kubernetes 软件包管理器，它是使用图表在 Kubernetes 群集上部署容器化应用程序的事实标准。Helm 可用于在一个或多个容器中安装、更新和去除应用程序。它还可以处理关联的资源，如配置、存储卷等。例如，可以使用它来部署 RMT 服务器（有关详细信息，请参见 RMT documentation (https://documentation.suse.com/sles/15-SP4/single-html/SLES-rmt/index.html#sec-rmt-deploy-kubernetes)）。

## 2.7 Distribution

Distribution (https://github.com/distribution/distribution) 是一个开源的注册表实现，用于通过 OCI 分发规范存储和分发容器映像。它为构建大规模注册表解决方案或运行简单专用注册表提供了简单、安全且可缩放的基础。Distribution 还可以镜像 Docker Hub，但not any other private registry (https://docs.docker.com/registry/recipes/mirror/#gotcha)。

## 2.8 Open Build Service

Open Build Service (OBS) (https://openbuildservice.org) 提供用于构建和存储 RPM 软件包（包括不同的容器格式）的免费基础架构。OBS Container Registry (https://registry.opensuse.org) 提供由 OBS 构建的所有容器映像的详细列表，使用命令完成将映像提取到本地环境的操作。您可以根据特定需求修改 OBS openSUSEcontainer image templates (https://build.opensuse.org/image_templates)，这是创建您自己的容器分支的最简单方法。您可以使用 `Dockerfile` 通过 Docker 工具从现有映像构建容器映像。也可以使用 KIWI NG 映像构建解决方案从头开始构建映像。

以下blog post (https://openbuildservice.org/2018/05/09/container-building-and-distribution/) 中提供了有关如何在 OBS 上构建映像的说明。

SUSE 容器映像称为 SLE 基本容器映像 (SLE BCI)，它们是仅有的官方容器映像。它们未在 https://build.opensuse.org 上提供，导出到该网站的 RPM 与内部 RPM 不同。这意味着无法在 https://build.opensuse.org 上构建官方支持的映像。

有关 SLE BCI 的详细信息，请参见第 4 节 “通用 SLE 基本容器映像”。

## 2.9 KIWI NG

KIWI (https://github.com/OSInside/kiwi) Next Generation (KIWI NG) 是一种用于构建映像的多功能工具。除容器映像、常规安装 ISO 映像以及虚拟机映像之外，KIWI NG 还可构建通过 PXE 或 Vagrant 盒子引导的映像。KIWI NG 中的主要构建基块是映像 XML 说明，这是一个包含 `config.xml` 或 `.kiwi` 文件以及脚本或配置数据的目录。使用 KIWI NG 创建映像的过程是完全自动化的，不需要任何用户交互。映像创建过程所需的任何信息由主要配置文件 `config.xml` 提供。可以使用 `config.sh` 和 `images.sh` 脚本自定义该映像。KIWI NG 是 Open Build Service (OBS) (https://openbuildservice.org) 的核心引擎。

### 注意

请务必将 KIWI NG（当前版本 9.20.9）与其不再维护的旧版本（7.x.x 或更早版本，现在称为 KIWI Legacy (https://documentation.suse.com/kiwi/)）加以区分。

有关如何安装 KIWI NG 以及使用它来构建映像的信息，请参见 KIWI NG documentation (https://osinside.github.io/kiwi/)。KIWI NG GitHub repository (https://github.com/OSInside/kiwi-descriptions) 上提供了一系列示例映像说明。

KIWI NG 手册页提供了使用该工具的相关信息。要访问手册页，请安装 `kiwi-man-pages` 软件包。

# 3 SLE 基本容器映像简介

SLE 基本容器映像 (SLE BCI) 是基于 SLES 15 的精简映像，可用于开发、部署和共享应用程序。SLE BCI 有两种类型：

- 通用 SLE BCI 可用于构建自定义容器映像和部署应用程序。
- 开发堆栈 SLE BCI 为以特定编程语言开发和部署应用程序提供精简的环境。

SUSE Registry (https://registry.suse.com) 上提供了 SLE 基本容器映像。其中包含已经过测试和更新的 SLE 基本容器映像。SUSE 注册表中的所有映像都要经历一个维护过程。构建的映像包含最新可用的更新和修复。SUSE 注册表的 Web 用户界面列出了一部分可用映像。有关 SUSE 注册表的信息，请参见第 2.1 节 “SUSE 注册表”。

SUSE 注册表中的 SLE 基本映像会收到安全更新，并涵盖在 SUSE 支持计划中。有关这些支持计划的详细信息，请参见第 22 节 “兼容性和支持条件”。

## 3.1 为何使用选择 SLE 基本容器映像

SLE BCI 提供了一个平台，用于创建基于 SLES 的自定义容器映像和可任意分发的容器化应用程序。SLE BCI 具有与 SLES 相同的可预测企业生命周期。SLE_BCI 15 SP3 和 SP4 储存库（SLE 储存库的子集）使 SLE BCI 能够访问 4000 个适用于 AMD64/Intel 64、AArch64、POWER 和 IBM Z 体系结构的软件包。储存库中的软件包享有 SUSE 的质量保证并接受其安全审计。以 FIPS 模式在主机上运行时，容器映像符合 FIPS 规范。除此之外，SUSE 还可以通过 SUSE 订阅计划为 SLE BCI 提供官方支持。

**安全性**

SLE_BCI 储存库中的每个软件包都接受安全审计，SLE BCI 享有与 SUSE Linux Enterprise Server 相同的 CVE 处理机制。所有发现和修复的漏洞将通过电子邮件、专门的 CVE pages (https://www.suse.com/security/cve/) 以 OVAL 和 CVRF 数据的形式公布。为确保供应链的安全，所有容器映像均已使用 Notary v1、Podman 的 GPG 签名和 Sigstore Cosign 进行签名。

**可靠性**

由于 SLE BCI 基于 SLES，因此它们具有相同级别的稳定性和质量保证。类似于 SLES，SLE BCI 可以通过维护更新来获得 bug 修复、改进和安全补丁。

**工具和集成**

SLE BCI 旨在为 hub.docker.com 上的流行容器映像提供直接替代项。您可以使用通用 SLE BCI 及其提供的现成工具来创建自定义容器映像，而开发堆栈 SLE BCI 为构建容器化应用程序提供基础和所需的工具。

**重新分发**

SLE 基本容器映像的 EULA (https://www.suse.com/de-de/licensing/eula/#bci) 条款较为宽松，您可以再分发基于 SLE 基本容器映像的自定义容器映像。

## 3.2 高亮显示数

- SLE BCI 与 SLES 完全兼容，但**无需订阅即可运行和分发**。
- 当主机操作系统以 FIPS 模式运行时，SLE BCI 会自动以 FIPS 兼容模式运行。
- 每个 SLE BCI 都包含 RPM 数据库，因此可以审计容器映像的内容。您可以使用 RPM 数据库来确定任意给定文件所属的 RPM 软件包的特定版本。这可以确保容器映像不容易受到已知漏洞和已修复漏洞的影响。
- 所有 SLE BCI（不包含 Zypper 的 SLE BCI 除外）都附带 `container-suseconnect` 服务。因此，在已注册的 SLES 主机上运行的容器可以访问完整的 SLES 储存库。当您首次运行 Zypper 时会自动调用 `container-suseconnect`，该服务会将正确的 SLES 储存库添加到正在运行的容器中。在未注册的 SLES 主机或非 SLES 主机上，该服务不执行任何操作。有关更多信息，请参见第 11.2 节 “将 container-suseconnect 与 SLE BCI 配合使用”。

### 注意：SLE_BCI 储存库

每个 SLE 服务包都有一个 SLE_BCI 储存库。这意味着，基于 SP4 的 SLE BCI 可以访问 SP4 的 SLE_BCI 储存库，所有基于 SP5 的 SLE BCI 可以使用 SP5 的 SLE_BCI 储存库，依此类推。每个 SLE_BCI 储存库包含除内核、引导加载程序、安装程序（包括 YaST）、桌面环境和超级管理程序（例如 KVM 和 Xen）之外的其他所有 SLE 软件包。

### 注意：请求缺失的软件包

如果 SLE_BCI 储存库不包含您所需的软件包，您可以采取两种做法。现有的 SUSE 客户可以提交功能请求。普通用户可以通过在 Bugzilla (https://bugzilla.suse.com/enter_bug.cgi?product=SUSE%20Linux%20Enterprise%20Base%20Container%20Images) ↗ 中报告问题来请求创建软件包。

# 4 通用 SLE 基本容器映像

通用 SLE BCI 有四个，其中每个容器映像附带最少量的一组软件包，以减小其大小。可将通用 SLE BCI 用作构建自定义容器映像的基础，或用作部署特定软件的平台。

SUSE 提供了以下几个通用 SLE BCI：BCI-Base、BCI-Minimal、BCI-Micro 和 BCI-BusyBox。这些 SLE BCI 可用作部署目标或用作创建自定义映像的基础。这些映像采用通用的 SLES 基础，并且它们都不随附特定的语言或应用程序堆栈。所有映像具有 RPM 数据库（即使特定的映像不包含 RPM 软件包管理器），该数据库可用于校验映像中每个文件的来源。每个映像包含 SLES 证书分发包，已部署的应用程序可以通过该分发包使用系统的证书来校验 TLS 连接。

下表简要概述了 SLE BCI-Base、SLE BCI-Minimal、SLE BCI-Micro 和 SLE BCI-BusyBox 之间的差异。

表 1： 支持矩阵

| 功能 | SLE BCI-Base | SLE BCI-Minimal | SLE BCI-Micro | SLE BCI-BusyBox |
|---|---|---|---|---|
| glibc | ✓ | ✓ | ✓ | ✓ |
| CA 证书 | ✓ | ✓ | ✓ | ✓ |
| rpm 数据库 | ✓ | ✓ | ✓ | ✓ |
| coreutils | ✓ | ✓ | ✓ | busybox |
| bash | ✓ | ✓ | ✓ | ✗ |
| rpm（二进制文件） | ✓ | ✓ | ✗ | ✗ |
| zypper | ✓ | ✗ | ✗ | ✗ |

## 4.1 SLE BCI-Base 和 SLE BCI-Init：需要灵活性时

此 SLE BCI 附带 Zypper 软件包管理器和免费的 SLE_BCI 储存库。这样，您便可以安装储存库中提供的软件并在构建期间自定义映像。该映像的缺点在于其大小。它是最大的通用 SLE BCI，因此它不一定是最佳的部署映像选项。

名为 SLE BCI-Init 的 SLE BCI-Base 变体中预安装了 systemd。在需要通过 systemd 在单个容器中管理服务的方案中，SLE BCI-Init 容器映像可能很有用。

### 重要：将 SLE BCI-init 与 Docker 配合使用

将 SLE BCI-init 容器与 Docker 配合使用时，必须使用以下参数才能使 `systemd` 在容器中正常工作：

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

要正确关闭容器，请使用以下命令：

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

## 4.2 SLE BCI-Minimal：不需要 Zypper 时

这是 SLE BCI-Base 映像的精简版本。SLE BCI-Minimal 不包含 Zypper，不过其中安装了 RPM 软件包管理器。这样就大幅减小了该映像的大小。但是，虽然 RPM 可以安装和去除软件包，但它缺乏对储存库和自动依赖项解析的支持。因此，SLE BCI-Minimal 映像旨在用于创建部署容器，然后在容器中安装所需的 RPM 软件包。尽管您可以安装所需的依赖项，但需要手动下载和解析这些依赖项。不过，不建议采用这种方法，因为它很容易出错。

## 4.3 SLE BCI-Micro：需要部署静态二进制文件时

此映像类似于 SLE BCI-Minimal，但不包含 RPM 软件包管理器。该映像的主要用例是部署在外部生成的或者在多阶段构建期间生成的静态二进制文件。由于没有直截了当的方法可以在容器映像中安装其他依赖项，因此我们建议仅在最终构建工件捆绑了所有依赖项，并且不存在外部运行时要求（例如 Python 或 Ruby）时，才使用 SLE BCI-Minimal 映像来部署项目。

## 4.4 SLE BCI-BusyBox：需要最小的映像且无需 GPLv3 授权时

类似于 SLE BCI-Micro，SLE BCI-BusyBox 映像仅附带最基本的工具。但是，这些工具是由 BusyBox 项目提供的。其好处是可以进一步减小大小。此外，该映像不包含 GPLv3 授权的软件。使用该映像时，请注意 BusyBox 工具和 GNU Coreutils 之间存在一些差异。因此，针对使用 GNU Coreutils 的系统编写的脚本可能需要经过修改才可用于 BusyBox。

## 4.5 大致大小

以下列表提供了每个 SLE BCI 的大致大小供您参考。请记住，提供的数字是粗略估计值。

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

# 5 使用 SUSE 注册表中的长期服务包支持容器映像

registry.suse.com/suse/ltss/ 上提供了长期服务包支持 (LTSS) 容器映像。要访问和使用此类容器映像，您必须拥有有效的 LTSS 订阅。

您必须先以用户身份登录 SUSE 注册表，然后才能提取或下载 LTSS 容器映像。可通过三种方式登录。

**使用主机系统的系统注册功能**

如果要用于构建或运行容器的主机系统已通过访问 LTSS 容器映像所需的正确订阅注册，那么您可以使用主机的注册信息来登录注册表。

文件 `/etc/zypp/credentials.d/SCCcredentials` 中包含了用户名和口令。这些身份凭证让您能够访问在相应主机系统的订阅下可用的任何容器。您可以使用这些身份凭证通过以下命令来登录 SUSE 注册表（请在 **`echo`** 命令前使用前导空格，以避免将身份凭证存储到外壳历史记录中）：

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

**使用单独的 SUSE Customer Center 注册代码**

如果主机系统未在 SUSE Customer Center 中注册，您可以使用有效的 SUSE Customer Center 注册代码登录注册表，如下所示：

```
set +o history
```

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

本例中的用户参数为逐字字符串 `regcode`，`SCC_REGISTRATION_CODE` 则是从 SUSE 获取的实际注册代码。

**使用组织镜像身份凭证**

您还可以使用组织镜像身份凭证来登录 SUSE 注册表，如下所示：

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

这些身份凭证让您能够访问组织拥有的所有订阅，包括与 SUSE 注册表中的容器映像相关的订阅。身份凭证具有较高权限，最好仅用于专用镜像注册表。

# 6 开发堆栈 SLE 基本容器映像

开发堆栈 SLE BCI 构建在 SLE BCI-Base 的基础之上。每个容器映像附带 Zypper 堆栈和免费的 `SLE_BCI` 储存库。此外，每个映像包含用于在特定语言环境中构建和部署应用程序的最常用工具。其中包括编译器或解释器等工具，以及特定于语言的软件包管理器。

下面概述了 SUSE Registry (https://registry.suse.com) 中提供的开发堆栈 SLE BCI。

**python**

随附 tag 以及 pip3、curl、git 工具中的 python3 版本。

**node**

附带 tag、npm 和 git 中的 nodejs 版本。可以使用 `npm install -g yarn` 命令安装 yarn 软件包管理器。

**openjdk**

随附 OpenJDK 运行时。旨在用于部署 Java 应用程序。

**openjdk-devel**

除了 OpenJDK 运行时之外，还包括 OpenJDK 的开发部分。默认入口点是 jshell 外壳，而不是 Bash。

**ruby**

基于 Ruby 2.5 的标准开发环境，随附 ruby、gem 和 bundler 以及 git 和 curl。

**rust**

随附 Rust 编译器和 Cargo 软件包管理器。

**golang**

随附 tag 中指定的 go 编译器版本。

**dotnet-runtime**

包含 Microsoft 的 .NET 运行时和 Microsoft .NET 储存库。

**dotnet-aspnet**

随附 Microsoft 的 ASP.NET 运行时和 Microsoft .NET 储存库。

**dotnet-sdk**

随附 Microsoft 的 .NET 和 ASP.NET SDK，以及 Microsoft .NET 储存库。

**php**

随附 tag 中指定的 PHP 版本。

# 7 应用程序 SLE 基本容器映像

应用程序 SLE BCI 是基于 SLE BCI 的容器映像，其中包含特定的应用程序，例如 PostgreSQL 数据库和 Performance Co-Pilot（系统级性能分析工具包）。应用程序 SLE BCI 在 SUSE Registry (https://registry.suse.com/#apps) 的专门部分中提供。

# 8 有关 SLE 基本容器映像的状态和生命周期的重要说明

除基本映像以外的其他所有容器映像目前都归类为技术预览，SUSE 不为它们提供官方支持。此信息会在 registry.suse.com (https://registry.suse.com) 网站上显示。此外，还会通过 `com.suse.supportlevel` 标签来指示某个容器映像是否仍处于技术预览状态。可以使用 skopeo 和 jq 实用程序检查所需 SLE BCI 的状态，如下所示：

```
> 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"
```

在上面的示例中，`bci-micro` 容器映像中的 `com.suse.supportlevel` 标签设置为 `techpreview`，表示该映像仍处于技术预览状态。相比之下，`bci-base` 容器映像则享受完全的 L3 支持。与通用 SLE BCI 不同，开发堆栈 SLE BCI 可能不遵循 SLES 分发套件的生命周期：只要相应的语言堆栈能够获得支持，语言堆栈 SLE BCI 就受支持。换言之，新版本的 SLE BCI（由 OCI 标记表示）可能会在 SLES Service Pack 的生命周期内发布，而旧版本可能不受支持。请访问 https://suse.com/lifecycle 确定相关容器是否仍受支持。

**重要**

SLE 基本容器映像在其支持周期结束后将不再更新。如果发生这种情况，您不会收到任何通知。

# 9 SLE 基本容器映像标签

SLE BCI 具有以下标签。

**com.suse.eula**

标示 SUSE EULA 的哪一部分适用于容器映像。

**com.suse.release-stage**

指示映像的当前发布阶段。

- `prototype` 指示容器映像处于 ALP 原型阶段。
- `alpha` 防止容器映像显示在 registry.suse.com Web 界面中，即使该映像在该界面中可用。该值还指示容器映像的 Alpha 质量。

- `beta` 在 registry.suse.com Web 界面的“Beta 容器映像”部分列出容器映像，并为该映像添加 Beta 标签。该值还指示容器映像的 Beta 质量。
- `released` 指示容器映像已发布并适合用于生产。

**com.suse.supportlevel**

显示容器的支持级别。

- `l2` 问题隔离，该技术支持级别旨在分析数据、重现客户问题、隔离问题区域，并针对级别 1 不能解决的问题提供解决方法，或完成准备工作以提交级别 3 处理。
- `l3` 问题解决，该技术支持级别旨在借助工程方法解决级别 2 支持所确定的产品缺陷。
- `acc` 与 SLE 基本容器映像一起交付的软件可能需要签署外部合同。
- `techpreview` 该映像不受支持，旨在用于概念证明方案。
- `unsupported` 不为映像提供支持。

**com.suse.lifecycle-url**

指向提供映像生命周期信息的 https://www.suse.com/lifecycle/ 页面。

## 9.1 使用 SLE BCI 标签

所有基本容器映像都包含构建时戳和说明等信息。此信息以附加到基本映像的标签形式提供，因此可用于派生的映像和容器。

下面是 `podman inspect` 显示的标签信息的示例：

```
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"
            },
    [...]
```

所有标签将显示两次，以确保派生映像中有关原始基本映像的信息仍然可见且不会被覆盖。

可以使用 Podman 来检索本地映像的标签。以下命令会列出所有标签，但仅列出 `bci-base:15.5` 映像的标签信息：

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

还可以检索特定标签的值：

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

上面的命令检索 `com.suse.sle.base.supportlevel` 标签的值。

使用 skopeo 工具可以检查映像的标签，而无需首先提取该映像。例如：

```
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 标记

标记用于引用映像。标记是映像名称的组成部分。与标签不同，标记可以自由定义，通常用于表示版本号。

如果某个标记存在于多个映像中，则将使用最新的映像。由映像维护者决定分配给容器映像的标记。

常规的标记格式为 `repository name:image version specification`（后者通常是版本号）。例如，最新发布的 SLE 15 SP2 映像的标记为 `suse/sle15:15.2`。

# 11 了解 SLE BCI

SLE BCI 有一些与类似产品（例如基于 Debian 或 Alpine Linux 的映像）不同的特性。了解细节有助于在尽可能短的时间内发挥 SLE BCI 的最大作用。

## 11.1 软件包管理器

SLES 中的默认软件包管理器是 Zypper。与 Debian 中的 APT 和 Alpine Linux 中的 APK 类似，Zypper 可为所有软件包管理任务提供一个命令行界面。下面是常用的容器相关 Zypper 命令的简要概览。

**安装软件包**

```
zypper --non-interactive install PACKAGE_NAME
```

**添加储存库**

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

**更新所有软件包**

```
zypper --non-interactive update
```

**去除软件包**

```
zypper --non-interactive remove --clean-deps PACKAGE_NAME
```

（`--clean-deps` 标志确保同时删除不再需要的依赖项）

**清理临时文件**

```
zypper clean
```

有关使用 Zypper 的详细信息，请参见 https://documentation.suse.com/sles/html/SLES-all/cha-sw-cl.html#sec-zypper。

所述的所有命令都使用 `--non-interactive` 标志来跳过确认，因为在容器构建期间您无法手动同意这些提示。请记住，必须将该标志与任何修改系统的命令一起使用。另请注意，`--non-interactive` 不是一个“全部为是”标志。`--non-interactive` 确认用户的意图。例如，如果某个附带 `--non-interactive` 选项的安装命令需要导入新的储存库签名密钥，则该命令将会失败，因为该操作必须由用户自己来校验。

## 11.2 将 container-suseconnect 与 SLE BCI 配合使用

`container-suseconnect` (https://github.com/SUSE/container-suseconnect) 是 Zypper 随附的所有 SLE BCI 中的一个可用插件。当该插件检测到主机的 SUSE Linux Enterprise Server 注册身份凭证时，会使用这些身份凭证来向容器授予对 SUSE Linux Enterprise 储存库的访问权限。这包括不属于免费 SLE_BCI 储存库的其他模块和以前的软件包版本。有关如何使用 SLES、openSUSE 和非 SLES 主机的储存库的详细信息，请参见第 13.3 节 “container-suseconnect”。

## 11.3 常用软件集

下面的示例演示了如何在 SLE BCI 中以不同于 Debian 的方式完成某些任务。

**去除孤立的软件包**

- Debian：`apt-get autoremove -y`
- SLE BCI：如果您使用 `zypper --non-interactive remove --clean-deps PACKAGE_NAME` 去除已安装的软件包，则不需要此项

**获取容器的体系结构**

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

**安装编译所需的软件包**

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

**校验 GnuPG 签名**

- 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 软件包命名约定

SLE 软件包命名约定与 Debian、Ubuntu 和 Alpine 不同，与 Red Hat Enterprise Linux 的命名约定较为接近。主要差别在于库的开发软件包（即包含头文件和构建描述文件的软件包）在 SLE 中命名为 `PACKAGE-devel`，而在 Debian 和 Ubuntu 中则是命名为 `PACKAGE-dev`。如有疑问，请使用以下命令搜索软件包：`docker run --rm registry.suse.com/bci/bci-base:OS_VERSION zypper search PACKAGE_NAME`（请将 `OS_VERSION` 替换为相应的服务版本号，例如 `15.3` 或 `15.4`）。

## 11.5 添加 GPG 签名密钥

将外部储存库添加到容器或容器映像通常需要导入用于对软件包进行签名的 GPG 密钥。这可以使用 `rpm --import KEY_URL` 命令来完成。这会将密钥添加到 RPM 数据库，然后可以安装储存库中的所有软件包。

# 12 SLE 基本容器映像入门

可以直接从 SUSE Registry (https://registry.suse.com) 以 OCI 兼容容器映像的形式获取 SLE BCI，并可以像使用任何其他容器映像一样使用它们，例如：

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

或者，可以如下所示在 `Dockerfile` 中使用 SLE BCI：

```
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
```

然后可以使用 **`docker build .`** 或 **`buildah bud .`** 命令构建容器映像：

```
> 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 注册和联机储存库

要在 SUSE Linux Enterprise Server 上使用容器，首先必须启用 SLE Containers 模块。此模块由与容器相关的软件包组成，其中包括容器引擎和核心容器工具，例如本地注册表。有关 SLE 模块的详细信息，请参见 https://documentation.suse.com/sles/html/SLES-all/article-modules.html。

常规 SLE 订阅包含免费的 SLE Containers 模块。

## 13.1 使用 YaST 图形界面启用 Containers 模块

1. 启动 YaST，然后选择软件 › 软件储存库。
2. 单击添加打开附加产品对话框。
3. 在 `Registration Server` 中选择 `Extensions and Modules`，然后单击下一步。
4. 在可用扩展和模块列表中选择 `Containers Module 15 SP4 x86_64`，然后单击下一步。这会将 `Containers Module` 及其储存库添加到系统中。
5. 如果您使用 Repository Mirroring Tool，请更新 RMT 服务器上的储存库列表。

## 13.2 使用 SUSEConnect 从命令行启用 Containers 模块

您也可以使用以下命令添加 Containers 模块：

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

## 13.3 container-suseconnect

`container-suseconnect` (https://github.com/SUSE/container-suseconnect) 是 Zypper 随附的所有 SLE 基本容器映像中的一个可用插件。当该插件检测到主机的 SUSE Linux Enterprise Server 注册身份凭证时，会使用这些身份凭证来向容器授予对 SUSE Linux Enterprise 储存库的访问权限。这包括不属于免费 SLE_BCI 储存库的其他模块和以前的软件包版本。

### 13.3.1 在 SLES 和 openSUSE 上使用 container-suseconnect

如果您正在使用 Docker 运行已注册的 SLES 系统，`container-suseconnect` 会自动检测和使用订阅，而无需您执行任何操作。

在包含 Docker 的 openSUSE 系统上，您必须将文件 `/etc/SUSEConnect` 和 `/etc/zypp/credentials.d/SCCcredentials` 从已注册的 SLES 计算机复制到本地计算机。请注意，仅当您使用 RMT 管理注册身份凭证时才需要 `/etc/SUSEConnect` 文件。

### 13.3.2 在非 SLES 主机上使用 container-suseconnect 或将其与 Podman 和 Buildah 配合使用

需要一个已注册的 SLES 系统才能在非 SLES 主机上使用 `container-suseconnect` 或将其与 Podman 和 Buildah 配合使用。此系统可以是物理机、虚拟机或者安装并注册了 `SUSEConnect` 的 SLE BCI-Base 容器。

如果您不使用 RMT，请将 `/etc/zypp/credentials.d/SCCcredentials` 复制到开发计算机。否则请复制 `/etc/zypp/credentials.d/SCCcredentials` 和 `/etc/SUSEConnect` 文件。

可以使用以下命令获取 `SCCcredentials`（请将 `REGISTRATION_CODE` 替换为您的 SCC 注册码）

```
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"
```

如果您正在运行基于 SLE BCI 的容器，请将 `SCCcredentials`（以及可选的 `/etc/SUSEConnect`）挂载到正确的目标中。以下示例说明如何在当前工作目录中挂载 `SCCcredentials`：

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

请不要将 `SCCcredentials` 和 `SUSEConnect` 文件复制到容器映像中，以免无意中将这些文件添加到最终映像中。请改用机密，因为它们仅适用于单个层，而不是构建的映像的一部分。为此，请将 `SCCcredentials`（以及可选的 `SUSEConnect`）的副本放到文件系统上的某个位置，并修改调用 Zypper 的 **`RUN`** 指令，如下所示：

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

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

Buildah 支持通过 `--secret` 标志挂载机密，如下所示：

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

### 注意：已知问题

每次调用 Zypper 时，container-suseconnect 都会自动运行。如果未使用已注册的 SLES 主机，您可能会看到以下错误消息：

```
> 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.
```

可以忽略该消息，因为它仅表明 container-suseconnect 无法检索您的 SUSE Customer Center 身份凭证，因此无法添加完整的 SLE 储存库。您仍然拥有对 SLE_BCI 储存库的完全访问权限，并且可以继续按预期使用容器。

## 13.3.3 将模块添加到容器或容器映像

`container-suseconnect` 允许您自动将 SLE 模块添加到容器或容器映像。可添加哪些模块由环境变量 `ADDITIONAL_MODULES` 决定，该变量包含逗号分隔的模块名称列表。在 `Dockerfile` 中，可以使用 `ENV` 指令完成此操作，如下所示：

```
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 概述

Podman (https://podman.io/) 是 Pod Manager 工具的简称。它是一个无守护程序的容器引擎，用于在 Linux 系统上管理 Open Container Initiative (OCI) 容器。默认情况下，Podman 支持无 root 权限容器，这减少了运行容器时的攻击面。可以使用 Podman 通过 `Dockerfile` 和一系列与 Docker 开源引擎相同的命令来创建符合 OCI 标准的容器映像。例如，`podman build` 命令执行与 `docker build` 相同的任务。换言之，Podman 可以直接替代 Docker 开源引擎。

从 Docker 开源引擎迁移到 Podman 不需要对既有的工作流程进行任何更改。您无需重构建映像，可以使用完全相同的命令来构建和管理映像，以及运行和控制容器。

Podman 与 Docker 开源引擎的差别体现在两个方面：

- Podman 不使用守护程序，因此容器引擎在需要时将直接与映像注册表、容器和映像存储区交互。
- Podman 具有本机 systemd 集成功能，允许使用 systemd 来运行容器。Podman 支持使用 **`podman generate systemd`** 命令来生成所需的 systemd 单元文件。此外，Podman 可以在容器内部运行 systemd。
- Podman 无需 root 特权即可创建和运行容器。这意味着 Podman 既能以 `root` 用户身份运行，也能在非特权环境中运行。此外，非特权用户创建的容器在主机上无法获得高于容器创建者的特权。
- 可将 Podman 配置为通过读取 `/etc/containers/registries.conf` 文件来搜索多个注册表。
- Podman 可以从 Kubernetes 清单部署应用程序
- Podman 支持在容器内部启动 systemd，并且不要求采取可能存在风险的权宜解决方法。

使用 Podman 可以将多个容器组合成 Pod。Pod 共享同一个网络接口。将多个容器组合成一个 Pod 的典型场景是：一个容器运行数据库，另一个容器装有访问该数据库的客户端。有关 Pod 的更多信息，请参见第 21.1 节 “使用 Podman 的单容器主机”。

## 14.1 Podman 安装

要安装 Podman，请确保启用 SLE Containers 模块（参见第 13 节 “注册和联机储存库”），并运行命令 **`sudo zypper in podman`**。然后运行 **`podman info`** 来检查 Podman 是否已成功安装。

Podman 默认会以当前用户的身份启动容器。对于非特权用户，这意味着以无 root 权限模式启动容器。默认已为 SLE 中所有新建的用户启用了对无 root 权限容器的支持，无需执行额外的步骤。

如果 Podman 无法以无 root 权限模式启动容器，请检查 `/etc/subuid` 中是否存在当前用户的对应项：

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

如果未找到该项，请通过以下命令添加所需的子 UID 和子 GID 项：

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

要启用更改，请重引导计算机或停止当前用户的会话。要执行后一项操作，请运行 **`loginctl list-sessions | grep`** `USER` 并记下会话 ID。然后运行 **`loginctl kill-session`** `SESSION_ID` 以停止会话。

上面的 **`usermod`** 定义了主机上的一系列本地 UID，分配给容器中的用户的 UID 将映射到这些 UID。请注意，为不同用户定义的范围不得重叠。同样重要的是，这些范围不能重复使用现有本地用户或组的 UID。默认情况下，在 SUSE Linux Enterprise 中使用 **`useradd`** 命令添加用户会自动分配子 UID 和子 GID 范围。

使用包含 Podman 的无 root 权限容器时，建议使用 cgroups v2。与 cgroups v2 相比，cgroups v1 在功能方面存在限制。例如，cgroups v1 不允许对用户的子树进行适当的分层委派。此外，Podman 无法使用 cgroups v1 和 systemd 日志驱动程序正确读取容器日志。要启用 cgroups v2，请将以下内容添加到内核命令行：`systemd.unified_cgroup_hierarchy=1`

在 SUSE Linux Enterprise Server 上以无 root 权限模式使用 Podman 运行容器可能会失败，因为容器需要拥有 SUSE Customer Center 身份凭证的读取访问权限。例如，使用命令 **`podman run -it --rm registry.suse.com/suse/sle15 bash`** 运行容器，然后执行 **`zypper ref`** 会导致出现以下错误消息：

```
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
```

要解决该问题，请在主机上运行以下命令为当前用户授予所需的访问权限：

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

注销然后重新登录以应用更改。

要为多个用户授予所需的访问权限，请使用 **`groupadd`** `GROUPNAME` 命令创建一个专用组。然后使用以下命令更改 `/etc/zypp/credentials.d/` 目录中的文件的组所有权和权限。

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

之后，您便可以通过将特定用户添加到这个创建的组来为其授予写入访问权限。

### 14.1.1 关于无 root 权限容器的提示和技巧

Podman 将用户 ID 重新映射到无 root 权限容器。在以下示例中，Podman 将当前用户重新映射到容器中的默认用户：

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

请注意，即使您是容器中的 root 用户，也无法在容器外部获取超级用户特权。

与主机共享数据时，如果共享的文件属于容器和主机中的不同用户 ID，这种用户重新映射可能会产生负面影响。可以使用命令行标志 `--userns=keep-id` 来解决该问题，该标志可以保留容器中的当前用户 ID：

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

与绑定挂载一起使用时，标志 `--userns=keep-id` 可以提供类似的效果：

```
> 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 将容器的数据存储在存储图根目录中（默认为 `~/.local/share/containers/storage`）。由于 Podman 在无 root 权限容器中重新映射用户 ID 的方式，图根目录可能包含不是由当前用户拥有的，而是由子 UID 区域中分配给该用户的用户 ID 拥有的文件。由于这些文件不属于当前用户，因此您可能无法访问它们。

要读取或修改图根目录中的任何文件，请如下所示进入外壳：

```
> podman unshare bash
```

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

请注意，在启动无 root 权限容器时，**`podman unshare`** 像 **`podman run`** 一样执行用户重新映射。您无法通过 **`podman unshare`** 获取提升的特权。

请不要修改图根目录中的文件，因为这可能会损坏 Podman 的内部状态，并使容器、映像和卷无法正常运行。

### 14.1.2 有关无 root 权限容器的注意事项

由于非特权用户无法在 Linux 上配置网络名称空间，因此 Podman 依赖于名为 `slirp4netns` 的用户空间网络实现。该实现模拟完整的 TCP-IP 堆栈，可能会导致依赖于较高网络传输速率的工作负载出现严重的性能下降。也就是说，当网络传输速率缓慢时，无 root 权限容器会受到影响。

在 Linux 上，非特权用户无法打开端口号小于 1024 的端口。此限制也适用于 Podman，因此默认情况下，无 root 权限容器无法公开端口号小于 1024 的端口。可以使用 **`sysctl net.ipv4.ip_unprivileged_port_start=0`** 命令消除此限制。

要永久消除限制，请运行 **`sysctl -w net.ipv4.ip_unprivileged_port_start=0`**。

请注意，这会允许**所有**非特权应用程序绑定到端口号小于 1024 的端口。

### 14.1.3 podman-docker

由于 Podman 与 Docker 开源引擎兼容，因此它提供相同的命令行界面。您还可以安装软件包 `podman-docker`，以便可以将模拟的 Docker CLI 与 Podman 配合使用。例如，从注册表提取容器映像的 **`docker pull`** 命令会改而执行 **`podman pull`**。**`docker build`** 命令执行 **`podman build`** 等命令

Podman 还具有与 Docker 开源引擎兼容的套接字，可使用以下命令来启动该套接字：

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

旨在用来与 Docker 开源引擎通讯的应用程序可以使用 Podman 套接字通过 Podman 以透明方式启动容器。Podman 套接字可用于使用 **`docker compose`** 启动容器，而无需运行 Docker 开源引擎。

## 14.2 获取容器映像

### 14.2.1 配置容器注册表

默认情况下，Podman 配置为仅使用 SUSE 注册表。要使 Podman 首先搜索 SUSE 注册表并使用 Docker Hub 作为回退搜索源，请确保 `/etc/containers/registries.conf` 文件包含以下配置：

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

### 14.2.2 在注册表中搜索映像

使用 **`podman search`** 命令可以列出 `/etc/containers/registries.conf` 中定义的注册表中的可用容器。

要在所有注册表中搜索，请运行以下命令：

```
podman search go
```

要在特定的注册表中搜索，请运行以下命令：

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

### 14.2.3 下载（提取）映像

**`podman pull`** 命令从映像注册表提取映像：

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

例如：

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

请注意，如果未指定标记，Podman 将提取 `latest` 标记。

## 14.3 重命名映像和映像标记

标记用于为容器映像分配描述性名称，方便识别各个映像。

从 SUSE 注册表中提取 SLE BCI-Base 映像：

```
> 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
```

列出提取的映像：

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

将 SLE BCI-Base 映像重命名为 `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
```

将自定义标记 `1`（表示这是映像版本 1）添加到 `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
```

请注意，默认标记 `latest` 仍然存在。

## 14.4 部署容器映像

与 Docker 开源引擎类似，Podman 能够以交互模式运行容器，使您可以检查和处理映像。要以交互模式运行 `suse/sle15` 映像，请使用以下命令：

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

### 注意： 修复无法通过网络访问有 root 权限的容器的问题

有 root 权限的容器有时可能无法从外部访问。该问题是由 firewalld 重新加载其永久规则并丢弃由 Podman 的网络后端（CNI 或 Netavark）创建的任何临时规则引起的。临时的解决方法是使用 **`podman network reload --all`** 命令重新加载 Podman 网络。如果您使用 Netavark 1.9.0 或更高版本作为网络后端，那么可以使用 `netavark-firewalld-reload.service` 服务来永久解决该问题。启用并启动该服务的方式如下：

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

您可以分别通过运行 `podman info --format "{{.Host.NetworkBackend}}"` 和 `podman info --format "{{.Host.NetworkBackendInfo.Version}}"` 来检查所使用的后端和版本。

建议您为希望可从主机外部访问的容器添加永久防火墙规则。这样可以确保规则在防火墙重新加载和系统重引导时不会丢失。这种方法还提供了更大的灵活性（例如，它允许您将规则分配给特定的 firewalld 区域）。

## 14.5 使用 Podman 构建映像

Podman 可以从 `Dockerfile` 构建映像。**`podman build`** 命令的行为与 **`docker build`** 相同，并且接受相同的选项。

Podman 的配套工具 Buildah 提供了另一种构建映像的方式。有关 Buildah 的更多信息，请参见第 18 节 “Buildah 概述”。

# 15 设置 Docker 开源引擎

## 15.1 准备主机

在安装任何与 Docker 相关的软件包之前，需要启用 Containers 模块：

### 注意：内置 Docker 编排支持

容器编排是 Docker 开源引擎的一部分。尽管此功能已在 SUSE Linux Enterprise 中提供，但 SUSE 并不提供对它的支持，此功能仅作为技术预览提供。使用 Kubernetes 进行容器编排。有关细节，请参见 Kubernetes documentation (https://kubernetes.io/docs/getting-started-guides/kubeadm/)。

### 15.1.1 安装和配置 Docker 开源引擎

1. 安装 `docker` 软件包：

```
> sudo zypper install docker
```

2. 启用 Docker 服务，使其在系统引导时自动启动：

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

这样会同时启用 `docker.socket`。

3. 打开 `/etc/sysconfig/docker` 文件。搜索参数 `DOCKER_OPTS` 并添加 `--insecure-registry ADDRESS_OF_YOUR_REGISTRY`。

   a. 将 CA 证书添加到目录 `/etc/docker/certs.d/REGISTRY_ADDRESS`：

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

   b. 将 CA 证书复制到系统中：

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

4. 启动 Docker 服务：

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

   这样会同时启动 `docker.socket`。

Docker 守护程序将会侦听仅可供 root 用户以及 docker 组成员访问的本地套接字。docker 组是在安装软件包期间自动创建的。

要允许特定用户连接到本地 Docker 守护程序，请使用以下命令：

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

这会允许该用户与本地 Docker 守护程序通讯。

## 15.2 配置网络

要使容器能够访问外部网络，请启用 `ipv4 ip_forward` 规则。

### 15.2.1 Docker 开源引擎与 iptables 交互的方式

要了解有关容器如何相互交互以及如何与系统防火墙交互的详细信息，请参见 Docker documentation (https://docs.docker.com/v17.09/engine/userguide/networking/default_network/container-communication/)。

也可以阻止 Docker 开源引擎操作 `iptables`。请参见 Docker documentation (https://docs.docker.com/network/iptables/#prevent-docker-from-manipulating-iptables)。

## 15.3 存储驱动程序

Docker 开源引擎支持不同的存储驱动程序：

- vfs：如果 Docker 主机文件系统不支持写入时复制，将自动使用此驱动程序。此驱动程序比其他所列驱动程序更简单，不提供 Docker 开源引擎的某些优势，例如共享层。此驱动程序速度较慢，但很可靠。
- devicemapper：此驱动程序依赖于 device-mapper 精简配置模块。它支持写入时复制，因此能够提供 Docker 开源引擎的所有优势。
- btrfs：此驱动程序依赖于 Btrfs 来提供 Docker 开源引擎所需的所有功能。要使用此驱动程序，`/var/lib/docker` 目录必须位于 Btrfs 文件系统上。

SUSE Linux Enterprise 默认使用 Btrfs 文件系统。这会强制 Docker 开源引擎使用 btrfs 驱动程序。

可以通过更改 `/etc/sysconfig/docker` 文件中定义的 `DOCKER_OPTS` 变量值来指定要使用的驱动程序。可以手动进行此更改，也可以在 YaST 中浏览到系统 › /etc/sysconfig 编辑器 › 系统 › 管理 › DOCKER_OPTS 菜单，然后输入 `-s storage_driver` 字符串进行更改。

例如，要启用 devicemapper 驱动程序，请输入以下文本：

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

### 重要：挂载 /var/lib/docker

建议在单独的分区或卷上挂载 `/var/lib/docker`。这样，在文件系统损坏的情况下，操作系统能够不受影响地运行 Docker 开源引擎。

如果您为 `/var/lib/docker` 选择了 Btrfs 文件系统，强烈建议为其创建一个子卷。这可以确保从文件系统快照中排除该目录。如果不从快照中排除 `/var/lib/docker`，那么在开始部署容器后不久，就会出现文件系统耗尽磁盘空间的风险。此外，回滚到以前的快照也会重置 Docker 数据库和映像。有关详细信息，请参见https://documentation.suse.com/sles/html/SLES-all/cha-snapper.html#sec-snapper-setup-customizing-new-subvolume。

## 15.4 更新

对 `docker` 软件包进行的所有更新都会标记为“交互式”（即，不是自动更新），以避免发生可能会破坏运行中容器工作负载的意外更新。在应用 Docker 开源引擎更新之前，请停止所有正在运行的容器。

为防止数据丢失，请避免使用依赖于在 Docker 开源引擎更新后自动启动的容器的工作负载。尽管从技术上讲可以通过 `--live-restore` 选项使容器能够在更新期间保持运行状态，但这种更新可能会导致性能下降。SUSE 不支持此功能。

# 16 配置映像存储设备

在创建自定义映像之前，需要确定映像的存储位置。最简单的解决方案是将映像推送到 Docker Hub (https://hub.docker.com)。默认情况下，推送到 Docker Hub 的所有映像都是公用映像。切勿发布敏感数据或未授权的软件供公众使用。

您可以通过以下方式限制对自定义容器映像的访问：

- 付费订阅者可以通过 Docker Hub 创建私用储存库。
- 使用现场 Docker 注册表可以存储您的组织所用的全部容器映像。

您可以运行 Docker 注册表的本地实例，而不是使用 Docker Hub。Docker 注册表是一个用于存储和检索容器映像的开源平台。

## 16.1 运行 Docker 注册表

SUSE 注册表提供了一个容器映像，可用于将本地 Docker 注册表作为容器运行。在启动容器之前，请创建包含以下示例配置的 `config.yml` 文件：

```
version: 0.1
    log:
      level: info
    storage:
      filesystem:
        rootdirectory: /var/lib/docker-registry
```

```
    http:
      addr: 0.0.0.0:5000
```

另外，创建一个空目录以映射容器外部的 `/var/lib/docker-registry` 目录。此目录用于存储容器映像。

运行以下命令以从 SUSE 注册表提取注册表容器映像，并启动一个可以通过端口 5000 访问的容器：

```
> podman run -d --restart=always --name registry -p 5000:5000 \
    -v /PATH/config.yml:/etc/docker/registry/config.yml \
    -v /PATH/DIR:/var/lib/ \ docker-registry registry.suse.com/sles12/
registry:2.6.2
```

为了更方便地管理注册表，请创建相应的系统单元：

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

启用并启动注册表服务，然后校验其状态：

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

有关 Docker 注册表及其配置的更多细节，请参见 https://docs.docker.com/registry/ 上的官方文档。

## 16.2 限制

Docker 注册表存在两项主要限制：

- 它缺少任何形式的身份验证。这意味着，有权访问 Docker 注册表的任何人都可以向它推送映像以及从中提取映像，包括重写现有映像。
- 无法查看哪些映像已推送到 Docker 注册表。您需要记下其中存储的映像。此外，它没有搜索功能。

# 17 校验容器映像

校验容器映像可以确认其来源，从而确保供应链的安全性。本章提供有关为容器映像签名和校验容器映像的信息。

## 17.1 使用 Docker 校验 SLE 基本容器映像

通过 SUSE 注册表提供的映像签名存储在 Notary 中。您可以使用以下命令校验特定映像的签名：

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

例如，命令 **`docker trust inspect --pretty registry.suse.com/suse/sle15:latest`** 校验最新 SLE15 基本映像的签名。

要在提取映像时自动验证该映像，请将环境变量 `DOCKER_CONTENT_TRUST` 设置为 `1`。例如：

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

## 17.2 使用 Cosign 校验 SLE 基本容器映像

要校验 SLE BCI，请在容器中运行 Cosign。以下命令会从 SUSE 服务器提取签名密钥，并使用该密钥来校验最新的 SLE BCI 基本容器映像。

```
> podman run --rm -it gcr.io/projectsigstore/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"
        }
      }
    ]
```

签名密钥可用于校验所有 SLE BCI，SUSE Linux Enterprise 也附带了签名密钥（`/usr/share/container-keys/suse-container-key.pem` 文件）。

您还可以对照不可变防篡改帐本 rekor (https://github.com/sigstore/rekor) 检查 SLE BCI。例如：

```
> podman run --rm -it -e COSIGN_EXPERIMENTAL=1 gcr.io/projectsigstore/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"
        }
      }
```

```
    ]
```

如果校验失败，则 **`cosign verify`** 命令的输出如下所示。

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

## 17.3 使用 Podman 校验 SLE 基本容器映像

在使用 Podman 校验 SLE BCI 之前，您必须先指定 `registry.suse.com` 作为用于映像校验的注册表。

### 注意

在 SUSE Linux Enterprise 上请跳过此步骤，因为其中已经指定了正确的配置。

为此，请将以下配置添加到 `/etc/containers/registries.d/default.yaml`：

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

可以在 `/etc/containers/registries.d/` 中使用所需的文件名创建一个新文件，而不要编辑 `default.yaml`。

接下来，修改 /etc/containers/policy.json (https://github.com/containers/image/blob/main/docs/containers-policy.json.5.md) 文件。在 `docker` 属性下，添加如下所示的 `registry.suse.com` 配置：

```
{
      "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"
            }
          }
        ]
      }
    }
  }
```

指定的配置将指示 Podman、skopeo 和 Buildah 校验 `registry.suse.com` 储存库下的映像。这样，Podman 就会在提取映像之前使用指定的公共密钥检查签名有效性。如果校验失败，它会拒绝该映像。

### 注意

请不要去除 `transports.docker` 中的现有项。将 `registry.suse.com` 的对应项追加到列表中。

从 SUSE Signing Keys (https://www.suse.com/support/security/keys/) 提取用来为 SLE BCI 签名的公共密钥，或使用以下命令：

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

### 注意

在 SUSE Linux Enterprise 上可以不执行此步骤。签名密钥已在 `/usr/share/pki/containers/suse-container-key.pem` 中提供

从现在起，Buildah、Podman 和 skopeo 会自动校验从 `registry.suse.com` 提取的每个映像。无需执行额外的步骤。

如果校验失败，该命令会返回如下所示的错误消息：

```
> 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
```

如果已签名的映像和您的配置没有问题，您可以继续使用容器映像。

# 18 Buildah 概述

Buildah (https://buildah.io/) 是用于构建符合 OCI 标准的容器映像的工具。Buildah 可处理以下任务：

- 从头开始或者基于现有映像创建容器。
- 基于工作容器或者通过 `Dockerfile` 创建映像。
- 构建 OCI 或 Docker 开源引擎映像格式的映像。
- 挂载工作容器的根文件系统以进行操作。
- 使用容器根文件系统的已更新内容作为文件系统层来创建新映像。
- 删除工作容器或映像，以及重命名本地容器。

与 Docker 开源引擎相比，Buildah 具有以下优势：

- 使用该工具可以挂载工作容器的文件系统，使之可供主机访问。
- 通过使用 Buildah 子命令的脚本（而不是 `Containerfile` 或 `Dockerfile`）可以自动执行使用 Buildah 构建容器映像的过程。

- 与 Podman 类似，Buildah 不需要运行守护程序，可由非特权用户使用。
- 可以在不挂载 Docker 套接字的情况下在容器内部构建映像，这样可以提高安全性。

Podman 和 Buildah 都可用于构建容器映像。虽然 Podman 可让用户使用 Dockerfile 构建映像，但 Buildah 提供的映像构建选项和功能更广泛。

## 18.1 Buildah 安装

要安装 Buildah，请运行命令 **`sudo zypper in buildah`**。运行命令 **`buildah --version`** 检查是否已成功安装 Buildah。

如果您已安装 Podman 并将其设置为在无 root 权限模式下使用，则无需进行进一步配置，就能在非特权环境中使用 Buildah。如果需要为 Buildah 启用无 root 权限模式，请运行以下命令：

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

此命令将为当前用户启用无 root 权限模式。运行该命令后，注销然后重新登录即可启用更改。

以上命令会在主机上定义一系列本地 UID，分配给容器中的用户的 UID 将映射到这些 UID。请注意，为不同用户定义的范围不得重叠。同样重要的是，这些范围不能重复使用任何现有本地用户或组的 UID。默认情况下，如果在 SUSE Linux Enterprise 中使用 **`useradd`** 命令添加用户，将会自动分配 subUID 和 subGID 范围。

### 注意：无 root 权限模式下的 Buildah

在无 root 权限模式下，Buildah 命令必须在用户的已修改用户名称空间中执行。要进入此用户名称空间，请运行命令 **`buildah unshare`**。否则，**`buildah mount`** 命令将会失败。

## 18.2 使用 Buildah 构建映像

Buildah 使用单独的命令来构建映像，而不使用包含指令的特殊文件。使用 Buildah 构建映像涉及以下步骤：

- 运行基于指定映像的容器
- 编辑容器（安装软件包、配置设置等）

- 配置容器选项
- 将所有更改提交到新映像中

虽然此过程可能包括其他步骤（例如，挂载并使用容器的文件系统），但基本的工作流程逻辑是相同的。

下面的示例可让您大致了解如何使用 Buildah 构建映像。

```
container=$(buildah from suse/sle15) ❶
    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 指定基于指定的映像（在本例中为 `sle15`）的容器（也称为工作容器）。

❷ 在刚刚创建的工作容器中运行命令。在此示例中，Buildah 将运行 **`zypper up`** 命令。

❸ 将文件和目录复制到容器中的指定位置。在此示例中，Buildah 会将当前目录的全部内容复制到 `/usr/src/example/`。

❹ **`buildah config`** 命令指定容器选项。其中包括定义工作目录、公开端口以及在容器内部运行命令。

❺ **`buildah config --label`** 命令用于向容器分配标签。标签包括 `maintainer`、`description`、`version` 等。

❻ 通过提交所有修改基于工作容器创建映像。

❼ 删除工作容器。

# 19 创建自定义容器映像

要创建自定义映像，需要有 SUSE Linux Enterprise Server 的基本映像。您可以使用任何预构建的 SUSE Linux Enterprise Server 映像。

## 19.1 提取 SUSE Linux Enterprise Server 基本映像

要获取预构建的基本映像，请使用以下命令：

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

例如，对于 SUSE Linux Enterprise Server 15，命令如下：

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

有关获取特定基本映像的信息，请参见第 3 节 “SLE 基本容器映像简介”。

容器映像准备就绪后，便可以按照第 19.2 节 “自定义容器映像”中所述对其进行自定义。

## 19.2 自定义容器映像

### 19.2.1 储存库和注册

预构建的映像未配置任何储存库，也不包含任何模块或扩展。它们包含一个 zypper service (https://github.com/SUSE/container-suseconnect)，该服务根据运行容器的 SUSE Linux Enterprise Server 主机的配置联系 SUSE Customer Center 或 Repository Mirroring Tool (RMT) 服务器。该服务会获取容器映像所用产品的可用储存库列表。您还可以直接在 `Dockerfile` 中声明扩展。有关详细信息，请参见第 13.3 节 “container-suseconnect”。

#### 注意：SLE_BCI 软件源

默认基本映像包含 SLE_BCI 储存库。仅当某个容器是在未注册的 SLES 主机上构建的或是在此类主机上运行时，或者当容器无法使用注册身份凭证时，才使用此储存库。该储存库提供一部分可用于自定义 SLES 容器映像的 SLE 软件包。无需注册即可使用该储存库，但 SUSE 不对其提供支持。

您不需要在容器映像中添加任何身份凭证，因为 docker 守护程序会自动将计算机身份凭证插入到容器中的 `/run/secrets` 目录。这同样适用于主机系统的 `/etc/SUSEConnect` 文件，该文件会自动插入到 `/run/secrets` 目录中。

### 注意：身份凭证和安全性

`/run/secrets` 目录的内容永远不会包含在容器映像中，这表示不存在身份凭证泄露的风险。

### 注意：在已于 RMT 中注册的系统上构建映像

如果用于构建容器映像的主机系统已在 RMT 中注册，则默认行为只允许构建其代码库与主机相同的容器。例如，如果您的容器主机是 SLE 15 系统，则默认只能在该主机上构建基于 SLE 15 的映像。要为不同的 SLE 版本（例如，SLE 15 主机上的 SLE 12）构建映像，可将目标版本的主机计算机身份凭证插入到容器中，如下所述。

请注意，如果 RMT 服务器使用自我签名证书，则需要将匹配的 CA 证书添加到 `CA_TRUSTSTORE/rmt-server.pem` 中的容器才能接受证书。

如果主机系统已在 SUSE Customer Center 中注册，则此限制不适用。

要获取储存库列表，请使用以下命令：

```
> sudo zypper repos
```

这会自动将所有储存库添加到容器中。对于添加到系统的每个储存库，都会在 `/etc/zypp/repos.d` 下创建一个新文件。这些储存库的 URL 包含一个在 12 小时后自动失效的访问令牌。要续订令牌，请运行命令 **`zypper ref -s`**。将这些文件包含在容器映像中不会带来任何安全风险。

要使用一组不同的身份凭证，请将自定义 `/etc/zypp/credentials.d/SCCcredentials` 文件放入容器映像中。该文件中的计算机身份凭证包含您要使用的订阅。这同样适用于 `SUSEConnect` 文件：要覆盖运行容器的主机系统上的现有文件，请将自定义 `/etc/SUSEConnect` 文件添加到容器映像中。

现在，您便可以按照第 19.2.2 节 “为 SLE 12 SP5 和更高版本创建自定义映像”中所述使用 `Dockerfile` 来创建自定义容器映像。

编辑 `Dockerfile` 后，通过在 `Dockerfile` 所在的目录中运行以下命令来构建映像：

```
> podman build .
```

有关 `podman build` 选项的详细信息，请参见official Podman documentation (https://docs.podman.io/en/latest/markdown/podman-build.1.html)。

### 19.2.2 为 SLE 12 SP5 和更高版本创建自定义映像

下面的 `Dockerfile` 会创建一个基于 SUSE Linux Enterprise Server 15 的简单容器映像：

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

如果 Podman 主机计算机已在内部 RMT 服务器中注册，则该映像需要 RMT 使用的 SSL 证书：

```
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
```

如果您要将 SLE 扩展和模块添加到映像中，请参见第 13.3.3 节 “将模块添加到容器或容器映像”。

### 19.2.3 在公有云中的按需 SLE 实例上构建容器映像

如果要在公有云（AWS、GCE 或 Azure）中作为按需或即用即付实例启动的 SLE 实例上构建容器映像，您需要执行额外的步骤。要安装软件包和更新，需将按需公有云实例连接到更新基础架构。此基础架构基于 SUSE 在公有云提供商处运营的 RMT 服务器工作。

因此，您的计算机需要找到所需的服务并向其进行身份验证。这可以使用 `containerbuild-regionsrv` 服务来完成。通过公有云提供商的商城提供的公有云映像中提供此服务。在构建映像之前，必须通过运行以下命令在公有云实例上启动此服务：

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

要让该服务在系统引导时自动启动，请启用该服务：

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

SLE 基本映像提供的 Zypper 插件将连接到此服务，并检索身份验证细节以及有关要与哪个更新服务器通讯的信息。为此，必须在启用主机网络的情况下构建容器，例如：

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

由于公有云中的更新基础架构基于 RMT 工作，因此，为不同于主机 SLE 版本的 SLE 版本构建 SLE 映像也存在同样的限制（参见注意：在已于 RMT 中注册的系统上构建映像）。

# 20 创建应用程序容器映像

适合在容器内部运行的应用程序包括守护程序、Web 服务器以及公开 IP 通讯端口的应用程序。您可以使用 Podman 通过在容器内部执行构建过程、构建映像，然后部署基于映像的容器来自动完成构建和部署过程。

在容器内部运行应用程序具有以下优势：

- 包含应用程序的映像可在运行不同 Linux 主机发行套件和版本的服务器之间移植。
- 可以使用储存库来共享应用程序的映像。
- 可以在容器和主机系统中使用不同的软件版本，而不会造成依赖关系方面的问题。
- 可以运行同一应用程序多个彼此独立的实例。

使用 Podman 构建应用程序可获得以下优势：

- 可以准备整个构建环境的映像。
- 应用程序可以在构建它的同一个环境中运行。
- 开发人员可以在同一环境中测试其代码，就如同在生产环境中使用时一样。

下一节提供了有关为应用程序创建容器映像的示例和建议。在继续操作之前，请确保已按照第 19.1 节 “提取 SUSE Linux Enterprise Server 基本映像” 中所述激活 SUSE Linux Enterprise Server 基本映像。

## 20.1 运行采用特定软件包版本的应用程序

如果您的应用程序所需的软件包版本不同于系统上安装的软件包版本，您可以创建一个包含应用程序所需软件包版本的容器映像。下面的示例 `Dockerfile` 允许使用较旧的 `example` 软件包版本构建基于最新 SUSE Linux Enterprise Server 版本的映像：

```
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"]
```

通过在 `Dockerfile` 所在的目录中运行以下命令来构建映像：

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

上面显示的 `Dockerfile` 示例会在映像构建过程中执行以下操作：

1. 更新 SUSE Linux Enterprise Server 储存库。
2. 安装所需的 `example` 软件包版本。
3. 将应用程序软件包复制到映像。必须在构建环境中放置二进制 RPM。
4. 将应用程序解包。
5. 最后两个步骤会在容器启动后运行应用程序。

成功构建 `tux_application` 映像后，可使用以下命令启动基于新映像的容器：

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

请记住，关闭应用程序后，容器也会退出。

## 20.2 运行采用特定配置的应用程序

要运行使用不同配置的实例，请创建派生映像并在其中包含附加配置。以下示例使用文件 `/etc/example/configuration_example` 配置名为 example 的应用程序：

```
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
```

以上示例 `Dockerfile` 执行以下操作：

❶ 按第 19.1 节 “提取 SUSE Linux Enterprise Server 基本映像”中所述提取 `sle15` 基本映像。

❷ 刷新 example 的储存库和安装。

❸ 设置 `BACKUP` 环境变量（在从映像启动的容器中持久保存的变量）。您始终可以在运行容器时通过指定新值来重写该变量的值。

❹ 创建 `/backup` 目录。

❺ 将 `configuration_example` 复制到映像。

❻ 运行 `example` 应用程序。

## 20.3 在应用程序与主机系统之间共享数据

Podman 允许使用卷在主机和容器之间共享数据。您可以直接在 `Dockerfile` 中指定挂载点。但是，不能在 `Dockerfile` 中指定主机系统上的目录，因为该目录在构建时可能无法访问。在主机系统上的 `/var/lib/docker/volumes/` 下找到挂载的目录。

### 注意：丢弃对要共享的目录所做的更改

使用 `VOLUME` 指令指定挂载点后，将会丢弃使用 **`RUN`** 指令对目录所做的所有更改。指定挂载点后，该卷将成为临时容器的一部分，而成功构建后会去除该临时容器。这表示要使某些操作生效，必须在指定挂载点**之前**执行这些操作。例如，如果您需要更改权限，请先进行所需更改，然后再在 `Dockerfile` 中将该目录指定为挂载点。

运行容器时，使用 `-v` 选项指定主机系统上的特定挂载点：

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

### 注意

如果您在容器中指定相同的挂载点，`-v` 选项会重写 **`VOLUME`** 指令。

以下 `Dockerfile` 示例将构建一个映像，其中包含一个从主机文件系统读取 Web 内容的 Web 服务器。

```
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"]
```

上面的示例会将 Apache Web 服务器安装到映像，并将整个配置复制到该映像。`data` 目录由 admin 用户拥有，用作存储网页的挂载点。

## 20.4 在后台运行的应用程序

如果您的应用程序需要作为守护程序在后台运行，或者作为公开通讯端口的应用程序运行，您可以在后台运行容器。

公开端口的应用程序的示例 `Dockerfile` 如下所示：

```
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"]
```

1. 按第 19.1 节 “提取 SUSE Linux Enterprise Server 基本映像”中所述提取基本映像。
2. 映像的维护者（可选）。
3. 要复制到 `/etc/zypp/repos.d` 和 `/etc/zypp/services.d` 的储存库和服务文件。如此这些文件便可在容器中的主机上使用。
4. 用于刷新储存库和服务的命令。
5. 用于安装 Apache2 的命令。
6. 用于调试的测试代码行。如果一切符合预期，则可以删除此行。
7. `COPY` 指令将主机系统中的数据复制到服务器所用容器中的目录。前导井号字符 (`#`) 会将此行标记为注释，系统不会执行此行。
8. Apache Web 服务器的公开端口。

### 注意

要使用端口 80，请确保主机中没有任何其他服务器软件在此端口上运行。

要使用容器，请执行以下操作：

1. 准备好用于执行构建过程的主机系统。

   a. 确保主机系统已订阅 SUSE Linux Enterprise Server 的 Server Applications 模块。要查看已安装的模块或安装其他模块，请打开 YaST 并选择“添加系统扩展或模块”。

   b. 确保已按照第 19.1 节 “提取 SUSE Linux Enterprise Server 基本映像”中所述安装了 SUSE 注册表中的 SLE 映像。

   c. 将 `Dockerfile` 保存到 `docker` 目录中。

   d. 在容器中，您需要访问已在主机上注册的储存库和服务。要使这些储存库和服务可用，请将主机中的相应文件复制到 `docker/etc` 目录：

      ```
      > cd docker
      ```

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

您也可以不复制所有储存库和服务文件，只复制容器所需的一部分文件。

e. 将网站数据（例如 HTML 文件）添加到 `docker/data` 目录中。此目录的内容将复制到容器映像，从而由 Web 服务器发布。

2. 构建容器。使用 `-t` 选项为映像设置标签（在下面的命令中，标签为 `EXAMPLEUSER_PLAIN`）：

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

Docker 开源引擎会执行 `Dockerfile` 中提供的指令：提取基本映像、复制内容、刷新储存库、安装 Apache2 等

3. 从上一步中创建的映像启动容器实例：

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

Docker 开源引擎将返回容器 ID，例如：

```
7bd674eb196d330d50f8a3cfc2bc61a243a4a535390767250b11a7886134ab93
```

4. 在浏览器中访问 http://localhost:80/test.html。您应该会看到消息 Web 服务器正在运行。

5. 要查看正在运行的容器的概览，请运行 **`docker ps --latest`** 命令：

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

要停止并删除容器，请运行以下命令：

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

您可以执行以下步骤来使用生成的容器通过 Apache2 Web 服务器提供数据：

1. 在 `Dockerfile` 中：

    - 在示例 `Dockerfile` 中，注释掉以 `RUN echo` 开头的行，方法是在此行的开头添加 `#` 字符。
    - 在示例 `Dockerfile` 中，取消注释以 `COPY` 开头的行，方法是去除前导 `#` 字符。

2. 重构建映像。

3. 在分离模式下运行映像：

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

    Docker 开源引擎将返回容器 ID，例如：

    ```
    e43fff4ae9832ecdb7677c058a73039d7610c32145a1d9b6ad0a4ed52b5c4dc7
    ```

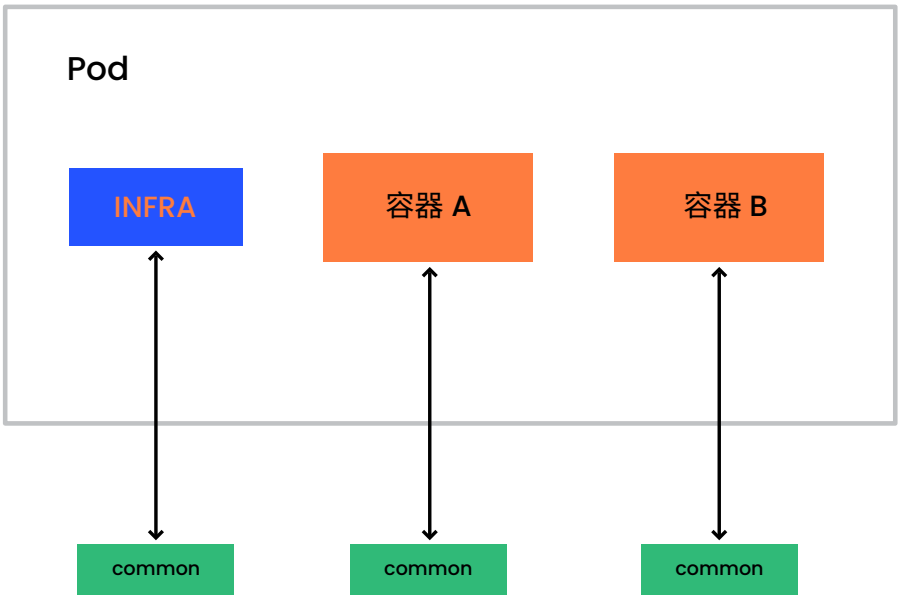
要查看已发布的数据，请在浏览器中指向 http://localhost:80/test.html 。

为了避免将网站数据复制到容器中，请与容器共享主机的某个目录。有关详细信息，请访问 https://docs.docker.com/storage/volumes/ 。

# 21 容器编排

在生产环境中，您可能会管理多个容器。要使用多个容器，您必须将多个容器组合成一个 Pod，该 Pod 提供有关部署和运行容器的规范，并允许它们共享存储和网络资源。换言之，Pod 会将包含多个容器的应用程序封装成一个单元。pod 的概念是由Kubernetes (https://kubernetes.io/docs/concepts/workloads/pods/) Podman 使用了与 Kubernetes 相同的定义。

通常，一个 Pod 中的容器可以直接相互通讯。每个 Pod 包含一个基础架构容器 (INFRA)，其用途是保存名称空间。INFRA 还允许 Podman 在 Pod 中添加其他容器。端口绑定、cgroup-parent 值和内核名称空间全都分配给基础架构容器。因此，以后无法更改这些值中的任何一个。




Pod 中的每个容器具有自身的监控程序实例。该监控程序会监控容器的进程，如果该容器消亡，监控程序会保存其退出代码。该程序还保持打开特定容器的 tty 接口。当 Podman 退出时，您可以使用该监控程序以分离模式运行容器，因为此程序会持续运行，并允许您稍后附加 tty。

## 21.1 使用 Podman 的单容器主机

**`podman pod`** 是用于创建、去除、查询和检查 Pod 的命令行工具。您可以在官方上游文档 (https://docs.podman.io/en/latest/markdown/podman-pod.1.html) 中查看 **`podman pod`** 的所有子命令。

**`podman pod create`** 会创建一个随机命名的 Pod。可以使用 `--name` 参数为 Pod 分配所需名称。

```
> podman pod create
    344940492c00b6a19ececbc5b109351bf0a3b8b19b3c279a097da7a653c592d0
```

可以使用 **`podman pod list`** 命令列出 Pod：

```
> podman pod list
```

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

还可以列出所有容器及其关联的 Pod：

```
> 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
```

创建的 Pod 包含一个通过 `localhost/podman-pause:4.x` 名称标识的 infra 容器。此容器的用途是预留与 Pod 关联的名称空间，并使 Podman 能够在 Pod 中添加其他容器。

使用 **`podman run --pod`** 命令可以运行容器并将其添加到所需的 Pod。例如，以下命令会运行一个基于 `suse/sle15` 映像的容器，并将该容器添加到 `suspicious_curie` Pod：

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

### 警告

如果此命令显示以下警告，请参见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
```

上面的命令会添加一个休眠 60 分钟然后退出的容器。再次运行 **`podman ps -a --pod`** 命令，您应该会看到该 Pod 现在包含两个容器。

您也可以使用命令 **`podman pod ps`** 来查看：

```
> podman pod ps
```

```
    POD ID        NAME          STATUS      CREATED         INFRA ID      # OF
 CONTAINERS
    344940492c00  suspicious_curie Running     21 minutes ago  617d7e3ce399  2
```

要停止新建的 `objective_jemison` 容器，请使用以下命令：

```
> 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
```

还可以使用 **`podman pod stop`** 停止 Pod 及其所有容器：

```
# 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
```

您还可以使用 **sudo podman start** CONTAINER_NAME、**podman pod start** POD_NAME 或 **podman pod restart** POD_NAME 来启动和重启动一切对象。

可以通过两种方式去除删除 Pod。可以使用 **podman pod rm** 命令去除一个或多个 Pod。或者，可以使用 **podman pod prune** 命令去除所有已停止的 Pod。要去除一个或多个 Pod，请如下所示运行 **podman pod rm** 命令：

```
> podman pod rm POD
```

POD 可以是 Pod 名称或 Pod ID。要去除所有当前已停止的 Pod，请使用 **podman pod prune** 命令。在运行 **podman pod prune** 命令之前，请确保所有已停止的 Pod 是您想要去除的 Pod，否则会存在去除仍在使用的 Pod 的风险。

使用容器运行时可以轻松启动作为单个容器分发的应用程序。但是，如果您需要运行包含多个容器的应用程序，或者需要在系统引导时自动启动应用程序，并在应用程序崩溃后将其重启动，事情会变得更复杂。虽然 Kubernetes 之类的容器编排工具正好可以解决这样的需求，但它们适合用于包含数百个节点的高度分布式可缩放系统，而不适合用于单台计算机。systemd 和 Podman 更适合单计算机场景，因为它们不会在现有设置中增加另一层复杂性。

Podman 支持使用 **podman generate systemd** 子命令创建 systemd 单元文件。该子命令创建一个 systemd 单元文件，使您可以通过 systemd 控制容器或 Pod。使用该单元文件，可以在系统引导时启动容器或 Pod，在发生故障时自动重启动容器或 Pod，并将其日志保存在 journald 中。

以下示例使用一个简单的 NGINX 容器：

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

可如下所示为容器生成 systemd 单元：

```
> 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 将一个单元文件输出到控制台，可将此文件放入用户单元 systemd 目录（`~/.config/systemd/user/` 或 `/etc/systemd/user/`）或系统单元 systemd 目录（`/etc/systemd/system`）中，它可以通过 systemd 控制容器。`--new` 标志指示 Podman 在重启动时重新创建容器。这可以确保 systemd 单元是自给性的，不依赖于任何外部状态。`--name` 标志可用于为容器分配一个用户友好的名称：如果不指定此标志，Podman 将使用容器 ID 而不是其名称。

要将容器作为用户单元进行控制，请执行以下命令：

```
> podman generate systemd --name --new --files web
```

```
    /home/user/container-web.service
    > mv container-web.service ~/.config/systemd/user/
    > systemctl --user daemon-reload
```

现在可以使用 **`systemctl --user start container-web`** 启动容器：

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

运行 **`podman ps`** 命令以查看所有正在运行的容器的列表：

```
> 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
```

通过 systemd 管理容器的好处之一是能够在容器崩溃时自动重启动容器。可以通过将 `SIGKILL` 发送到容器中的主进程来模拟崩溃：

```
> 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
```

请注意，如果容器是正常停止的（例如通过 **`podman stop web`**），则**不会**重启动。要使容器始终可重启动，请将标志 `--restart-policy=always` 添加到 **`podman generate systemd`** 中。

## 21.2 更新容器映像

使用所述的方法意味着容器映像永远不会更新。可以通过将 `--pull=always` 标志添加到单元文件中的 `ExecStart=` 项来解决该问题。但请注意，这会增加容器的启动时间，并在**每次**重启动时更新映像。后一种行为还意味着，容器映像更新可能会因新造成的 bug 而导致容器在安排的维护时段之外不可用。

Podman 中的 **`auto-update`** (https://docs.podman.io/en/latest/markdown/podman-auto-update.1.html) 子命令提供了一种可行的解决方法。将标签 `io.containers.autoupdate=registry` 添加到容器，使 Podman 在运行 **`podman auto-update`** 时从注册表中提取容器映像的新版本。这样，就可以在所需的时间使用一条命令更新所有容器映像，而且不会增加 systemd 单元的启动时间。

可以通过将 `--label "io.containers.autoupdate=registry" \` 一行添加到容器 systemd 单元文件的 `ExecStart=` 项来启用自动更新功能。对于 NGINX 示例，请如下所示修改 `~/.config/systemd/user/container-web.service`：

```
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
```

重新加载守护程序并重启动容器后，执行更新试运行（此操作很可能不会报告任何更新）：

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

最好进行外部测试，以确保映像更新在一般情况下可以安全部署。如果您对我们的容器映像的质量有信心，可以通过启用 `podman-auto-update.timer` 来让 Podman 自动定期应用映像更新：

```
# 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 管理多个容器

某些应用程序需要依赖于多个容器才能正常运行，例如 Web 前端、后端服务器和数据库。Docker compose (https://docs.docker.com/compose/) 是在一台计算机上部署多容器应用程序的流行工具。虽然 Podman 并不原生支持 **`compose`** 命令，但在大多数情况下，可以将 compose 文件移植到某个 Podman Pod 和多个容器。

以下示例会在单个 Pod 中部署 Drupal 和 PostgreSQL 容器，并通过 systemd 单元管理这些容器。首先，创建一个公开 Drupal Web 界面的新 Pod：

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

创建该 Pod 后，启动 Drupal 前端及其包含的 PostgreSQL 数据库：

```
> 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
```

这会生成三个正在运行的容器：Drupal Web 界面、PostgreSQL 数据库和 Pod 的基础架构容器。

```
> 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
```

为 Pod 创建 systemd 单元的方式类似于为单个容器创建此单元：

```
> 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
```

Podman 知道哪些容器属于 `drupal` Pod 以及如何调用这些容器的 systemd 单元，因此可以正确将依赖项添加到 Pod 的单元文件中。这意味着，当您启动或停止 Pod 时，systemd 会确保该 Pod 中的所有容器自动启动或停止。

要检查 systemd 的依赖项处理结果，请先停止 `drupal` Pod，然后校验主机上当前是否未运行任何容器：

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

通过 **`systemctl start --user pod-drupal.service`** 启动 `drupal` Pod，systemd 会启动该 Pod 中的容器：

```
> 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 有关 Podman 的详细信息

如要了解更多关于 Podman 和处理 Pod 部署的信息，请访问 https://docs.podman.io/en/latest/ 和 https://github.com/containers/podman。

## 21.5 使用 Kubernetes 的多容器主机

Kubernetes (https://kubernetes.io) 是一个开源容器编排引擎，用于自动执行容器化应用程序的部署、缩放和管理工作。该开源项目由云原生计算基金会 (CNCF (https://www.cncf.io/about)) 主办。

使用 Kubernetes 可使多台计算机（或者服务器或节点）能够协同工作并形成一个群集，然后您可以通过 API 来与此群集交互。我们建议使用 Rancher (https://ranchermanager.docs.rancher.com) 来部署 Kubernetes 群集并管理在群集中运行的应用程序。单个 Rancher 设置可以管理在任何位置运行的多个 Kubernetes 群集：从裸机到本地或云服务提供商。

有关 Rancher 的详细信息，请参见官方 Rancher 文档 (https://ranchermanager.docs.rancher.com)。

## 21.6 轻量级 Kubernetes (k3s)

K3s (https://k3s.io) 是专为物联网和边缘计算构建且经过 CNCF 认证的轻量级 Kubernetes 发行版。与 Kubernetes 不同，K3s 封装为单个不到 60 MB 的二进制文件，并针对 Arm 架构进行了优化。

有关详细信息，请参见Introduction to K3s (https://www.suse.com/c/rancher_blog/introduction-to-k3s/) 和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 兼容性和支持条件

术语“支持”指两个不同的概念：a) 功能或组合（例如主机和容器）的技术支持，以及 b) SUSE 向 SUSE 客户提供的企业支持。根据 https://www.suse.com/products/terms_and_conditions.pdf，企业支持要求订阅 SUSE 产品。技术支持如下所述。

## 22.1 对 SLES 主机的支持

请查阅以下支持和兼容性矩阵，确保所需的主机系统和容器组合兼容且受支持。

表 2： 支持矩阵

| 主机 ↓ 容器映像 → | SLES 12 | SLES 15 |
|---|---|---|
| **SLES 12 SP5** | ✓ | * |
| **SLES 15** | ✓ | ✓ |
| **SLE Micro** | ✓ | ✓ |

✓ 完全支持

* 有限支持（请参见“有限支持说明”）

### 重要：有限支持说明

由于容器化应用程序可能导致系统调用在主机内核中不可用，SUSE 为 SLES 12 SP5 主机上运行的基于 SLES GA 的容器提供了有限支持。为了避免潜在的风险和兼容性问题，SUSE 建议为容器和主机使用相同的服务包版本。

SLE BCI 支持以下体系结构：AMD64/Intel 64、AArch64、POWER 和 IBM Z。容器的体系结构必须与主机的体系结构相匹配。不匹配的容器和主机场景不受支持。

在大多数情况下，如果应用程序及其依赖项不直接与特定于内核版本的数据结构及其衍生结构（`ioctl`、`/proc`、`/sys`、`routing`、`iptables`、`nftables`、`BTF`、`(e)BPF` 等）或模块（KVM、OVS、SystemTap 等）交互，则所有 SLE 容器都应该可互操作。仅在非特权用户所需的最常见场景中提供对 `ioctl` 的支持和对 `/proc` 的访问权限。

## 22.2 对非 SLES 主机的支持

虽然基于 SUSE 的容器受到完全支持，主机环境中的问题仍必须由主机环境供应商负责处理。SUSE 支持属于 SUSE 基本容器的组件。还支持来自 SUSE 储存库的软件包。容器中的其他组件和应用程序不在 SUSE 支持的范围内。构建派生容器需要 SLE 订阅。

基于 SLES 12 SP5 和 SLES 15（所有服务包）的容器根据其官方生命周期和下表所述内容享受支持。

**支持以下第三方容器主机平台。**

表 3： 对非 SLES 主机的支持

| 容器主机平台 | 容器运行时 | 支持状态 |
| --- | --- | --- |
| **Rancher Kubernetes Engine (RKE)** | docker | ✓ |
| **Rancher Kubernetes Engine 2 (RKE2)** | containerd | ✓ |
| **K3S** | containerd | ✓ |
| **Red Hat OpenShift** | cri-o | ✓ |
| **Microsoft Azure Kubernetes Service (AKS)** | containerd | ✓ |
| **Google Kubernetes Engine (GKE)** | containerd | * |

| 容器主机平台 | 容器运行时 | 支持状态 |
| --- | --- | --- |
| **Amazon Elastic Container Service for Kubernetes (EKS)** | containerd | ✓ |
| **IBM Hyper Protect Platform** | docker/podman | ** |

✓ 完全兼容且完全受支持

* 取决于工作负载：完全受支持，但兼容性视容器类型（特权或非特权）和应用程序交互（直接与内核版本特定的数据结构、内核版本特定的模块等交互）而定

** 在完全兼容和全面支持测试完成前，处于临时支持状态。

有关对 Rancher 相关产品的支持的详细信息，请参见 Rancher 支持矩阵 (https://www.suse.com/suse-rancher/support-matrix/all-supported-versions/)。

## 22.3 支持计划

有三项 SUSE 容器支持的指导原则。

1. 容器映像生命周期遵循相关产品的生命周期。
   例如，SLES 15 SP4 容器映像遵循 SLES 15 SP4 生命周期。
2. 容器发布状态还与相关产品的状态相匹配。
   例如，如果 SLES 15 SP4 处于 Alpha、Beta、RC 或 GA 阶段，则相关容器具有同样的发布状态。
3. 容器是使用相关产品中的软件包构建的。
   例如，SLES 15 SP4 容器映像是使用与 SLES 15 SP4 主版本相同的软件包构建的。

有关更多信息，请访问 Product Support Lifecycle (https://www.suse.com/lifecycle) 页面，以及适用于 SUSE Registry (https://registry.suse.com) 中特定容器映像的文档。

容器映像可以处于不同的支持状态，因此对它们的支持可能有限制。有关特定容器映像的更多信息，请参见相应的 https://registry.suse.com 页面。

## 22.4 容器和主机环境支持概览

以下支持选项对 SUSE 主机环境中的 SLES 容器有效。

SUSE 提供的容器和主机环境完全受支持。这也适用于所有受支持的产品，包括一般支持和 Long Term Service Pack Support (https://www.suse.com/products/long-term-service-pack-support/) (LTSS)。

签署了联合工程协作协议的合作伙伴容器和主机环境受到完全支持。此层级适用于协议涵盖的容器、主机环境以及支持（包括一般支持和 LTSS）下的所有产品。

虽然基于 SUSE 的容器受到完全支持，主机环境中的问题仍必须由主机环境供应商负责处理。SUSE 支持来自 SUSE 基本容器的组件，还支持来自 SUSE 储存库的软件包。容器中的其他组件和应用程序不在 SUSE 支持的范围内。如果要基于 SLE BCI 或 SLE_BCI 储存库的内容构建派生容器，不需要订阅产品。要构建包含完整 SLE 系列中的软件包的容器，您需要购买订阅，使自己能够访问包含这些软件包的储存库。

上面未提到的任何容器和主机环境享受有限支持。相关细节请与 SUSE 支持团队讨论，该团队负责对问题进行分类并给出备用解决方案建议。在任何其他情况下，主机环境中的问题都必须由主机环境供应商负责处理。

## 22.5 技术预览

标记为技术预览的容器映像由 SUSE 提供，让您有机会在环境中测试新技术并分享反馈。如果您测试了技术预览，请联系 SUSE 代表，以分享您的经验和用例。您的反馈对于我们的未来开发非常有帮助。

技术预览存在以下限制：

- 技术预览可能在功能上不完整、不稳定或者不适合在生产环境中使用。
- 技术预览不受支持。
- 技术预览可能仅适用于特定的硬件体系结构。

- 技术预览的细节和功能可能随时会发生变化。因此，用户可能无法升级到技术预览的后续版本，而只能全新安装。
- 我们随时可能会取消技术预览。例如，如果 SUSE 发现某个预览不符合客户或市场的需求，或者不符合企业标准，就可能会取消该预览。SUSE 不承诺未来将提供此类技术的受支持版本。

容器映像标记为技术预览，并且在 registry.suse.com (https://registry.suse.com) 上也采用这种标记。此外，作为技术预览提供的容器映像在容器映像元数据中包含 `com.suse.supportlevel="techpreview"` 标签。可以使用 `docker inspect` 命令或其他容器运行时中的相应命令检查元数据是否包含该标签。

# 23 查错

## 23.1 使用 container-diff 分析容器映像

如果在 SLE 基本容器映像基础上构建的自定义 Docker 开源引擎容器映像无法按预期方式工作，`container-diff` 工具可帮助您分析该映像并收集用于查错的相关信息。

`container-diff` 可以分析映像的变化，具体方式是计算映像之间的差异并以直观易懂且可处理的格式呈现差异。该工具可以找出系统软件包、语言级软件包和容器映像中的文件的差异。

`container-diff` 可以处理本地容器映像（使用前缀 `daemon://`）、远程注册表中的映像（使用前缀 `remote://`），以及另存为 `.tar` 存档的映像。可以使用 `container-diff` 来计算映像的本地版本与远程版本之间的差异。

要安装 `container-diff`，请运行 **`sudo zypper in container-diff`** 命令：

### 23.1.1 基本的 container-diff 命令

命令 **container-diff analyze** IMAGE 对单个映像运行标准分析。它默认会返回容器映像的哈希和大小。如需可帮助您识别和修复问题的详细信息，请使用特定的分析器。使用 --type 参数指定所需的分析器。两个最有用的分析器是 history（返回映像层创建说明列表）和 file（返回文件系统内容的列表，包括名称、路径和大小）：

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

要查看所有可用参数及其简要说明，请运行 **container-diff analyze --help** 命令。

使用 **container-diff diff** 命令可以比较两个容器映像并检查两者的差异。类似于 **container-diff analyze** 命令，**container-diff diff** 支持多个参数。下面的示例命令会比较两个映像，并返回有关如何基于 IMAGE1 创建 IMAGE2 的说明列表。

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

要查看所有可用参数及其简要说明，请运行 **container-diff diff --help** 命令。

# 24 术语

**容器**

容器是基于特定容器映像的正在运行的实例。每个容器都可通过唯一的容器 ID 来识别。

**控制组**

控制组（也称为 cgroups）是一项 Linux 内核功能，可用于将任务（进程）及其所有子项聚合或划分成分层组织的组，以管理其资源限制。

**Docker 开源引擎**

Docker 开源引擎是一种服务器-客户端类型的应用程序，用于执行与容器相关的所有任务。Docker 开源引擎由以下组件构成：

- **守护程序：** + Docker 开源引擎的服务器端，负责管理所有 Docker 对象（映像、容器、容器使用的网络连接等）。
- **REST API：** + 应用程序可以使用此 API 来直接与守护程序通讯。
- **CLI 客户端：** + 可让您与守护程序通讯。如果守护程序与 CLI 客户端在不同的计算机上运行，CLI 客户端可以使用 Docker 开源引擎提供的网络套接字或 REST API 进行通讯。

### Dockerfile

Dockerfile 提供有关如何构建容器映像的指令。Docker 开源引擎会读取 Dockerfile 中的指令，并根据这些指令构建新映像。

### 映像

映像是用于创建容器的只读模板。Docker 映像由一系列相互叠加构建的层组成。每个层对应于一项永久性更改，例如，应用程序的一项更新。更改存储在称作 Dockerfile 的文件中。有关细节，请参见the official Docker documentation (https://docs.docker.com/engine/reference/glossary#image)。

### 容器映像

容器映像是一个不可更改的静态文件，它包含可执行代码，因此可以在 IT 基础架构上运行独立的进程。映像由系统库、系统工具以及在容器化平台上运行程序所需的其他平台设置组成。容器映像通过在父映像或基本映像上构建的文件系统层进行编译。

### 基本映像

基本映像是没有父映像的映像。在 Dockerfile 中，基本映像由 `FROM scratch` 指令进行识别。

### 父映像

充当其他容器映像基础的映像。换言之，如果某个映像不是基本映像，则它派生自父映像。在 Dockerfile 中，`FROM` 指令指向父映像。大多数 Docker 容器都是使用父映像创建的。

### 名称空间

Docker 开源引擎为其容器使用 Linux 名称空间，这样可以隔离为特定容器预留的资源。

**编排**

在生产环境中，您通常需要使用每个集群节点上有许多容器的群集。容器必须彼此协作，因而您需要通过一个框架来自动管理容器。自动管理容器的行为称为容器编排，通常由 Kubernetes 处理。

**注册表**

注册表是已创建映像的存储区。它通常包含多个储存库。注册表有两种类型：

- 公共注册表：任何用户（通常是已注册的用户）都可以下载和使用映像。公共注册表的一个典型示例是 Docker Hub (https://hub.docker.com/)。
- 专用注册表：仅限特定用户访问，或者从特定专用网络访问。

**软件源**

软件源是注册表中的映像的储存区。

# 25 改进文档

可以使用以下任一选项提交对本文档的反馈和补充。

**服务请求和支持**

有关产品可用的服务和支持选项，请参见 https://www.suse.com/support/。要创建服务请求，需在 SUSE Customer Center 中注册订阅的 SUSE 产品。请转到 https://scc.suse.com/support/requests 并登录，然后单击新建。

**Bug 报告**

在 https://bugzilla.suse.com/ 中报告文档问题（需要 Bugzilla 帐户）。要简化此过程，可以使用本文档 HTML 版本中的报告问题链接。将光标指向所需的句子，然后在右侧导航面板的提供反馈部分单击报告问题。这样会自动在 Bugzilla 中选择正确的产品和类别，并添加当前章节的链接。您现在可以编写 bug 报告。

**贡献**

要帮助改进本文档，请使用本文档 HTML 版本中的 Edit source document（编辑源文档）链接（需要 GitHub 帐户）。将光标指向所需的句子，然后在右侧导航面板的提供反馈部分单击报告问题。这样您将会转到 GitHub 上的源代码，可以在其中提出拉取请求。

### 注意：“Edit source document”（编辑源文档）仅适用于英语版本

Edit source document（编辑源文档）链接仅适用于每个文档的英语版本。对于其他所有语言，请按上文所述使用报告问题链接。

有关本文档使用的文档环境的详细信息，请参见储存库的 README 文件（网址为 https://github.com/SUSE/doc-unversioned/blob/main/README.adoc）

**邮件**

您也可以将有关本文档的错误以及反馈发送至 `doc-team@suse.com`。请在其中包含文档标题、产品版本和文档发布日期。此外，请包含相关的章节号和标题（或者提供 URL），并提供问题的简要说明。

# 26 法律声明



# 27 GNU 自由文档许可证

## 0. 导言

此许可证的目的是赋予手册、教科书或其他功能性的和有用的文档以“自由”：即保证每个人都有复制和再分发此类文档的有效自由，无论是否进行修改，也无论将其用于商业或非商业用途。其次，此许可证为作者和出版者保留了因工作获得声誉但不视为对他人所做修改负责的方式。

本许可证是一种“非盈利版权”，这意味着从该文档衍生的作品也必须是以同一方式自由的。它补充了 GNU 通用公共许可证（为自由软件设计的非盈利版权许可证）。

我们设计此许可证旨在将其用于免费软件的手册，因为免费软件需要自由文档：免费程序所附手册应具有与软件本身同样的自由。但是此许可证不限于软件手册；它可以用于任何文本作品，无论主题如何或它是否作为印刷书籍出版。建议本许可证主要用于目的是指导或参考的作品。

## 1. 适用性和定义

此许可证适用的对象：由版权所有者在其中明确声明可按照此许可证条款以任何媒体分发的任何手册和其他作品。此类声明授予在此处所述的条款和条件下使用该作品的全球无限期无版权许可证。下述“文档”指任何此类手册或作品。任何公众成员都是一个被许可人，以下称为“您”。如果您以需要版权法许可的任何方式复制、修改或分发该作品，则表示您接受该许可证。

该文档的“修改版本”表示包含该文档或其一部分（或者逐字复制或者有修改和/或翻译为另一语言）的任何作品。

“次要章节”是该文档的命名附录或扉页章节，专门讲述该文档的出版者或作者与该文档整个主题（或相关问题）的关系，不包含与整个主题相关的内容。（因此，如果该文档是数学课本的一部分，则辅助部分可能不说明任何数学问题。）这种关系可以是与主题或相关问题的历史联系，或与它们相关的法律、商业、哲学、伦理或政治地位。

在该文档基于此许可证项发布的声明中，“固定章节”是将其标题指定为固定章节标题的一些辅助章节。如果一个章节不适用上述辅助章节的定义，则不允许将其指定为固定章节。该文档可能不包含固定章节。如果该文档不标识任何固定章节，则表示没有固定章节。

在该文档基于此许可证项发布的声明中，“封页文本”是作为封面文本或封底文本列出的简短文本段落。封面文本最多 5 个单词，封底文本最多 25 个单词。

文档的“透明”副本是一个机器可读的副本，使用公众可以得到其规范的格式表达，这样的副本适合于使用通用文本编辑器、（对于像素构成的图像）通用绘图程序、（对于绘制的图形）广泛使用的绘画编辑器直接修改文档，也适用于输入到文本格式处理程序或自动翻译成各种适于适用于输入到文本格式处理程序的格式。一个用其他透明文件格式表示的副本，如果该格式的标记（或缺少标记）已经构成了对读者的后续修改的障碍，那么就是不透明的。表示实质性数量的文本的图像格式都是不透明的。不“透明”的副本称为“不透明”。

适于作为透明副本的格式的示例有：没有标记的纯 ASCII 文本、Texinfo 输入格式、LaTeX 输入格式、使用公共可用 DTD 的 SGML 或 XML，符合标准的简单 HTML、可以人为修改的 PostScript 或 PDF。透明图像格式的示例有 PNG、XCF 和 JPG。不透明的格式包括：仅可以被私有版权的字处理软件使用的私有版权格式、所用的 DTD 和/或处理工具不是广泛可用的的 SGML 或 XML、机器生成的 HTML、一些字处理器生成的只用于输出目的的 PostScript 或 PDF。

对于印刷书籍，“扉页”就是扉页本身以及随后的一些用于补充的页，显然本许可资料需要出现在扉页上。对于那些没有扉页的作品形式，“扉页”代表接近作品最突出标题的、在文本正文之前的文本。

“命名为 XYZ”的章节表示文档的一个特定的子单元，其标题就是 XYZ 或在括号中包含 XYZ 且后跟 XYZ 的其他语言翻译文本。（这里 XYZ 代表下面提及的特定章节名称，比如“致谢”、“题献”、“签名”或“历史”。）要在修改文档时对这类章节“保留标题”就是依据此定义保持这样一个“命名为 XYZ”的章节。

文档可能在文档遵照此许可证的声明后面包含免责声明。这些免责声明应作为参考信息包含在此许可证中，但是只能将其视作免责声明：这些免责声明暗指的任何其他含义均无效，且对此许可证的含义不产生任何影响。

## 2. 逐字复制

您可以用任何媒体复制并分发文档，无论是出于商业还是非商业目的，只要保证此许可证、版权声明和声称此许可证应用于文档的声明都完整地、无任何附加条件地存在于所有副本中。不能使用任何技术手段阻碍或控制您制作或发布的副本的阅读或再次复制。不过您可以在副本交易中得到报酬。如果发布足够多的副本，则您必须遵循下面第三节中的条件。

您也可以在如上的条件下出租副本和向公众放映副本。

## 3. 大量复制

如果您出版的文档印刷版副本（或是有印制封页的其他媒体副本）多于 100 份，而文档的许可证声明中要求有封页文本，则您必须将它清晰地置于封页之上，封面文本在封面上，封底文本在封底上。封面和封底上还必须标明您是这些副本的出版者。封面必须同等显著地完整展现标题的所有文字。您可以在封页上加入其他资料。改动仅限于封页的复制，只要保持文档的标题不变并满足这些条件，可以在其他方面被视为逐字复制。

如果需要加上的文本对于封面或封底过多，无法明显地表示，您应该在封页上列出前面的（在合理的前提下尽量多），把其他的放在邻近的页面上。

如果您出版或分发了超过 100 份文档的不透明副本，则必须在每个不透明副本中包含一份计算机可读的透明副本，或是在每个不透明副本中给出一个计算机网络地址，通过这个地址，网络公共用户可以使用标准网络协议下载文档的无任何附加资料的完整透明副本。如果您选择后者，则必须在开始大量分发非透明副本的时候采用相当谨慎的步骤，保证透明副本在其所给出的位置在（直接或通过代理和零售商）分发最后一次该版本的非透明副本的时间之后一年之内始终是有效的。

在重新大量发布副本之前，请您（但不是必须）与文档的作者联系，以便他们可以有机会向您提供文档的更新版本。

## 4. 修改

在上述第 2、3 节的条件下，您可以复制和分发文档的修改版本，前提是严格按照此许可证发布修改后的文档，将修改版本用作文档，从而允许任何拥有此修改版副本的人执行分发或修改。另外，在修改版中，您需要做到如下几点：

**A.** 用于与文档以及以前各个版本（如果有，应该列在文档的“历史”章节中）显著不同的扉页（和封页，如果有）。如果那个版本的原始发行者允许的话，您可以使用和以前版本相同的标题。

**B.** 与作者一样，在扉页上列出承担修改版本中的修改的作者责任的一个或多个人或实体和至少五个文档的原作者（如果原作者不足五个就全部列出），除非他们免除了您的这个责任。

**C.** 与原来的发行者一样，在扉页上列出修改版的发行者的姓名。

**D.** 保持该文档的全部版权声明不变。

E. 在与其他版权声明邻近的位置加入恰当的针对您的修改的版权声明。

F. 在紧接着版权声明的位置加入许可声明，按照下面附录中给出的形式，以本许可证给公众授于是用修订版本的权利。

G. 保持原文档的许可声明中的全部不可变章节、封面文字和封底文字的声明不变。

H. 包含一份未作任何修改的本协议的副本。

I. 保持命名为“历史”的章节不变，保持它的标题不变，并在其中加入一项，至少声明扉页上的已修改版本的标题、年份、新作者和出版者。如果文档中没有命名为“历史”章节，则请新建它，并加入一项以声明原文档扉页上所列的标题、年份、作者与出版者，再在其后加入如上所说的描述修改版本的项。

J. 如果文档中有用于公共用户访问的文档透明副本的网址，则保持网址不变，并同样提供它所基于的以前文档版本的网址。这些网址可以放在“历史”章节。您可以不给出那些在原文档发行之前已经发行至少四年的版本给出的网址，或者该版本的发行者授权不列出网址。

K. 对于任何命名为“致谢”或“题献”的章节，保持其标题不变，并保持其全部内容以及对每位贡献者致谢和/或题献的语气不变。

L. 保持文档的所有固定章节不变，不改变它们的标题和内容。章节的编号或相当的内容不被认为是章节标题的一部分。

M. 删除命名为“签名”的章节。这样的章节不可以被包含在修改后的版本中。

N. 不要把任何现有章节重命名为“签名”或与任何不可变章节相冲突的标题。

O. 保持任何免责声明不变。

如果修改版本加入了新的符合次要章节定义的引言或附录章节，并且不含有从原文档中复制的内容，您可以选择将其标记为固定。如果需要这样做，则将它们的标题加入修改版本许可声明的不可变章节列表之中。这些标题必须和其他章节的标题相区分。

您可以加入一个命名为“签名”的章节，只要它只包含对您的修改版本由不同的各方给出的签名，例如书评或是声明文本已经被一个组织认定为一个标准的权威定义。

您可以加入一个最多 5 个字的段落作为封面文本和一个最多 25 个字的段落作为封底文本，将它们加入修改版本的封页文本列表末端。一个实体只可以添加（或编排）一段封面和一段封底文本。如果原文档已经为该封页（封面或封底）包含了封页文本，由您或您所代表的实体先前加入或排列的文本，不能再新加入一个，但您可以在原来的发行者的显式许可下替换掉原来的那个。

作者和发行者不能通过本许可证授权公众使用他们的名字推荐或暗示认可任何一个修改版本。

## 5. 组合文档

遵照第 4 节所说的修改版本的规定，您以将文档和其他文档合并并以本许可证发布，只要您在合并结果中包含原文档的所有不可变章节，对它们不加以任何改动，并在合并结果的许可声明中将它们全部列为不可变章节，而且维持原作者的免责声明不变。

合并作品仅需要包含一份此许可证，多个相同的固定章节可以由一个副本取代。如果有多个名称相同但内容不同的固定章节，通过在章节名称后面的括号中加上原作者或出版者的姓名（如果已知）来加以区别，或者使用唯一编号加以区别。并对合并作品许可声明中的固定章节列表中的章节标题做相同的调整。

在合并过程中，必须合并不同原始文档中任何命名为“历史”的章节，从而形成新的命名为“历史”的章节；类似地，还要合并命名为“致谢”和“题献”的章节。必须删除所有命名为“签名”的章节。

## 6. 文档的合集

您可以制作一个文档和其他文档的合集，在本许可证下发布，并在合集中将不同文档中的多个本许可证的副本以一个单独的副本来代替，只要您在文档的其他方面遵循本许可证的逐字复制的条款即可。

您可以从一个这样的合集中提取一个单独的文档，并将它在本许可证下单独发布，只要您想这个提取出的文档中加入一份本许可证的副本，并在文档的其他方面遵循本许可证的逐字复制的原则。

## 7. 独立作品的聚合体

将文档或其派生品以及其他独立和无关文档或作品编撰在一个储存卷中或分发媒体上，这称为文档的“聚合体”，前提是编撰成品的著作权对其使用者的法律权限的限制未超出各个独立作品的许可范围。当基于此许可证发布的文档包含在一个聚合体中时，此许可证不适用于聚合体中的本非该文档派生作品的其他作品。

如果第 3 节中的封页文本要求适用于这些文档的副本，则若文档在聚合体中所占的比重小于全作品的一半，文档的封页文本可以放置在聚合体内包含文档部分的封页上，或是电子文档中的等效部分。否则，它必须位于整个聚合体的印刷的封页上。

## 8. 翻译

翻译被视为一种修改，因此您可以根据第 4 节的条款分发文档的翻译。将固定章节替换为翻译内容需要经得其版权所有者的特别许可，但除了这些固定章节的原始版本之外，您还可以包含一部分或所有固定章节的翻译。您可以包含一个此许可证以及所有许可证声明和免责声明的翻译版本，前提是同时包含它们的原始英文版本。当翻译版本和英文版发生冲突的时候，原始版本有效。

如果在文档中有命名为“致谢”、“题献”或“历史”的章节，保持标题（第 1 节）的要求（第 4 节）恰恰需要更换实际的标题。

## 9. 终止

除非此许可证中有明确规定，否则您不能对该文档进行复制、修改、分授许可或分发。在此许可证规定外对该文档所进行的任何复制、修改、分授许可或分发都是无效的，并且将自动终止您在此许可证下所拥有的权利。但是，对于在此许可证的规定下从您这里获得副本或权利的各方，只要其完全遵守此许可证的规定，其许可证将不会被终止。

## 10. 本许可的未来修订版本

自由软件基金会有时会发布 GNU 自由文档许可证的新的修订版版本。这些新版本的主旨和精神与当前版本是一致的，但在解决新问题的具体细节方面可能有所不同。请参见 https://www.gnu.org/copyleft/。

许可证的每个版本都有一个不同的版本号。如果文档指定了适用于它的此许可证“或任何后续版本”的特定带编号版本，则您可以选择遵从指定版本或自由软件基金会发布的任何随后版本（非草稿）的条款和条件。如果文档没有指定此许可证的版本号，您可以选择自由软件基金会发布的任何许可证版本（非草稿）。

## 附录：如何针对您的文档使用此许可证

```
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”.
```

如果您有固定章节、封面文本和封底文本，请将“with...Texts”部分替换为：

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

如果有不可变章节而没有封页文本，或这三种内容（不可变章节、封面文本、封底文本）的任何其他组合，请合并这两个备选项以适应您的情况。

如果您的文档包含不一般的程序代码示例，建议同时选择自由软件许可证（如 GNU 通用公共许可证）发布这些示例，以允许它们可以用于自由软件。