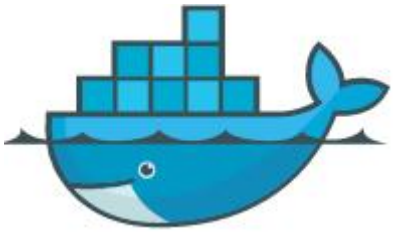
第 1 课 Docker介绍及安装

Docker 是一个开源的应用容器引擎，基于Go语言 并遵从 Apache2.0 协议开源。

Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。

容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。



Docker标志

Docker官网：http://www.docker.com

Docker中文网站：https://www.docker-cn.com

Docker Hub（仓库）官网：https://hub.docker.com

1. Docker介绍

1.1 什么是Docker

- Docker 是世界领先的软件容器平台。
- Docker 使用 Google 公司推出的 Go 语言 进行开发实现，基于 Linux 内核 的

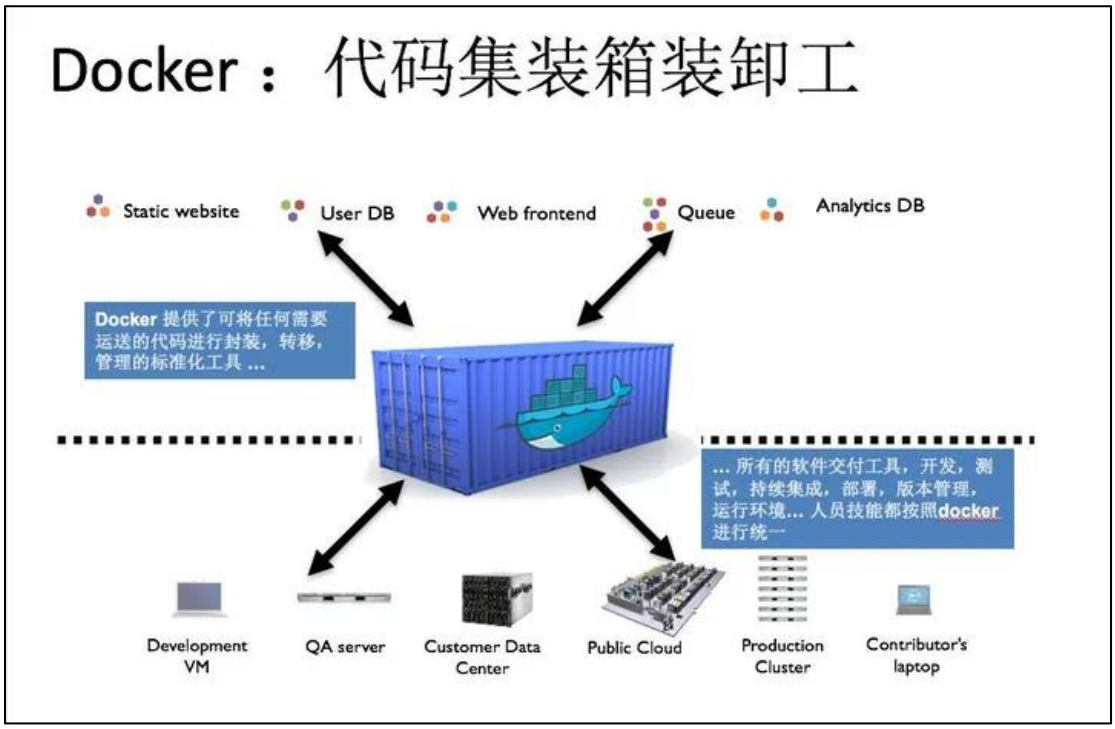
cgroup，namespace，以及AUFS类的UnionFS等技术，对进程进行封装隔离，属于操作系统层面的虚拟化技术。 由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。Docke最初实现是基于 LXC.

- Docker 能够自动执行重复性任务，例如搭建和配置开发环境，从而解放了开发人员以

便他们专注在真正重要的事情上：构建杰出的软件。

- 用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、

复制、分享、修改，就像管理普通的代码一样。



1.2 Docker思想

这个就是docker的logo，一条装满集装箱的鲸鱼，在鲸鱼背上，集装箱相互之间是隔离的，这也就是docker的核心思想了。

比如之前有多个应用在同一台服务器上运行，可能会有软件的端口占用冲突，现在隔离后就可以独自运行了。另外，docker可以最大化的利用服务器的能力。

1.3 Docker容器的特点

- 轻量

在一台机器上运行的多个 Docker 容器可以共享这台机器的操作系统内核；它们能够迅速启动，只需占用很少的计算和内存资源。镜像是通过文件系统层进行构造的，并共享一些公共文件。这样就能尽量降低磁盘用量，并能更快地下载镜像。

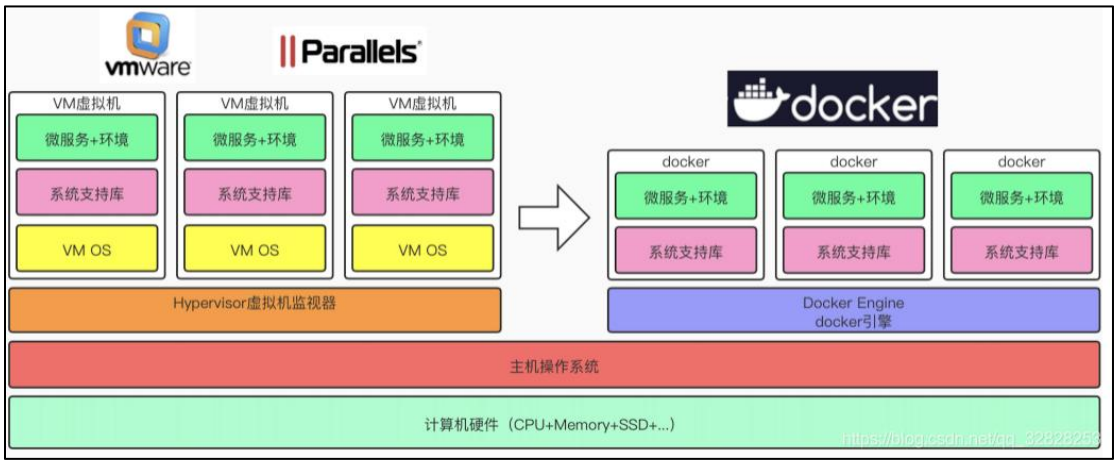
- 标准

Docker 容器基于开放式标准，能够在所有主流 Linux 版本、Microsoft Windows 以及包括 VM、裸机服务器和云在内的任何基础设施上运行。

- 安全

Docker 赋予应用的隔离性不仅限于彼此隔离，还独立于底层的基础设施。Docker 默认提供最强的隔离，因此应用出现问题，也只是单个容器的问题，而不会波及到整台机器。

1.4 对比虚拟机与Docker



- docker守护进程可以直接与主操作系统进行通信，为各个docker容器分配资源；

它还可以将容器与主操作系统隔离，并将各个容器互相隔离。虚拟机启动需要数分钟，而docker容器可以在数毫秒内启动。由于没有臃肿的从操作系统，docker可以节省大量的磁盘空间以及其他系统资源。

- 虚拟机更擅长于彻底隔离整个运行环境。例如，云服务提供商通常采用虚拟机技术隔离不同的用 户。而docker通常用于隔离不同的应用，例如前端，后端以及数据库。

- docker容器比虚拟机更加节省资源，速度更加快（启动、关闭、新建、删除）

1.5 为什么要用Docker

- 一致的运行环境

Docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现 “这段代码在我机器上没问题啊” 这类问题；

- 更快速的启动时间

可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间

- 隔离性

避免公用的服务器，资源会容易受到其他用户的影响。

- 弹性伸缩，快速扩展

善于处理集中爆发的服务器使用压力；

- 迁移方便

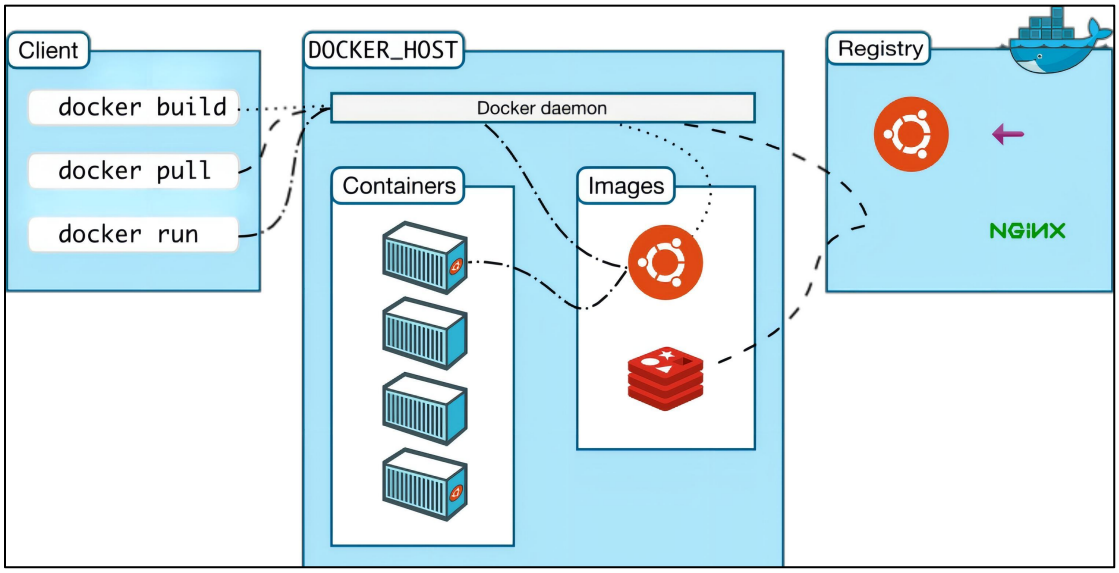
可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。

- 持续交付和部署

使用 Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署。

1.6 docker架构

docker使用客户端-服务器(client-server)架构。docker客户端(client)与docker守护进程（daemon）通信，后者负责构建、运行和分发docker容器的重担。docker客户端和守护进程可以在同一系统上运行，也可以将docker客户端连接到远程docker守护进程。docker客户端和守护进程通过UNIX套接字或网络接口使用REST API进行通信。另一个docker客户端是docker compose，它让你可以处理由一组容器组成的应用程序。



1.7 Docker 三个基本概念

1.7.1 镜像（Image）——一个特殊的文件系统

操作系统分为内核和用户空间。对于 Linux 而言，内核启动后，会挂载 root 文件系统为其提供用户空间支持。而Docker 镜像（Image），就相当于是一个 root 文件系统。

Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外,还包含了一些为运行时准备的一些配置参数(如匿名卷、环境变量、用户等)。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

Docker 设计时，就充分利用 Union FS的技术，将其设计为 分层存储的架构。镜像实际是由多层文件系统联合组成。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。 比如，删除前一层文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直跟随镜像。因此，在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，任何额外的东西应该在该层构建结束前清理掉。

分层存储的特征还使得镜像的复用、定制变的更为容易。甚至可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内容，构建新的镜像。

1.7.2 容器（Container)——镜像运行时的实体

镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等 。

容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的 命名空间。前面讲过镜像使用的是分层存储，容器也是如此。

容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。

按照 Docker 最佳实践的要求，容器不应该向其存储层内写入任何数据 ，容器存储层要保持无状态化。所有的文件写入操作，都应该使用数据卷（Volume）、或者绑定宿主目录，在这些位置的读写会跳过容器存储层，直接对宿主(或网络存储)发生读写，其性能和稳定性更高。数据卷的生存周期独立于容器，容器消亡，数据卷不会消亡。因此， 使用数据卷后，容器可以随意删除、重新 run ，数据却不会丢失。

1.7.3 仓库（Repository）——集中存放镜像文件的地方

镜像构建完成后，可以很容易的在当前宿主上运行，但是，如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，Docker Registry就是这样的服务。

一个 Docker Registry中可以包含多个仓库（Repository）；每个仓库可以包含多个标签（Tag）；每个标签对应一个镜像。所以说：镜像仓库是Docker用来集中存放镜像文件的地方类似于我们之前常用的代码仓库。

通常，一个仓库会包含同一个软件不同版本的镜像，而标签就常用于对应该软件的各个版本 。我们可以通过<仓库名>:<标签>的格式来指定具体是这个软件哪个版本的镜像。如果不给出标签，将以 latest 作为默认标签。

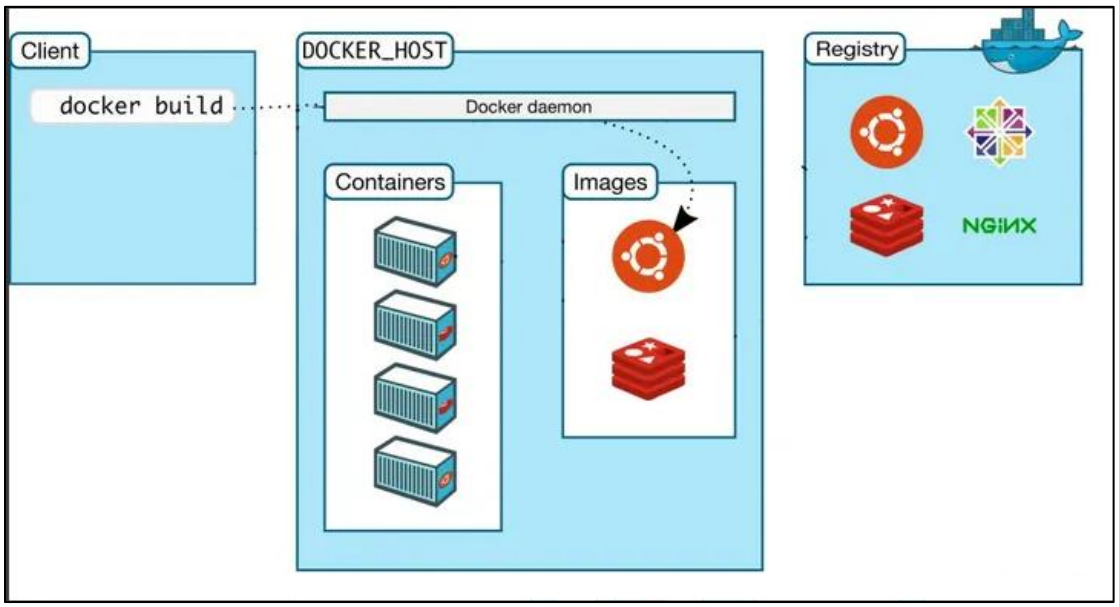
1.8 Docker运行机制

docker使用了常见的CS架构，也就是client-server模式，docker client负责处理用户输入的各种命令，比如docker build、docker run，真正工作的其实是server，也就是docker demon，值得注意的是，docker client和docker demon可以运行在同一台机器上。

接下来我们用几个命令来讲解一下docker的工作流程：

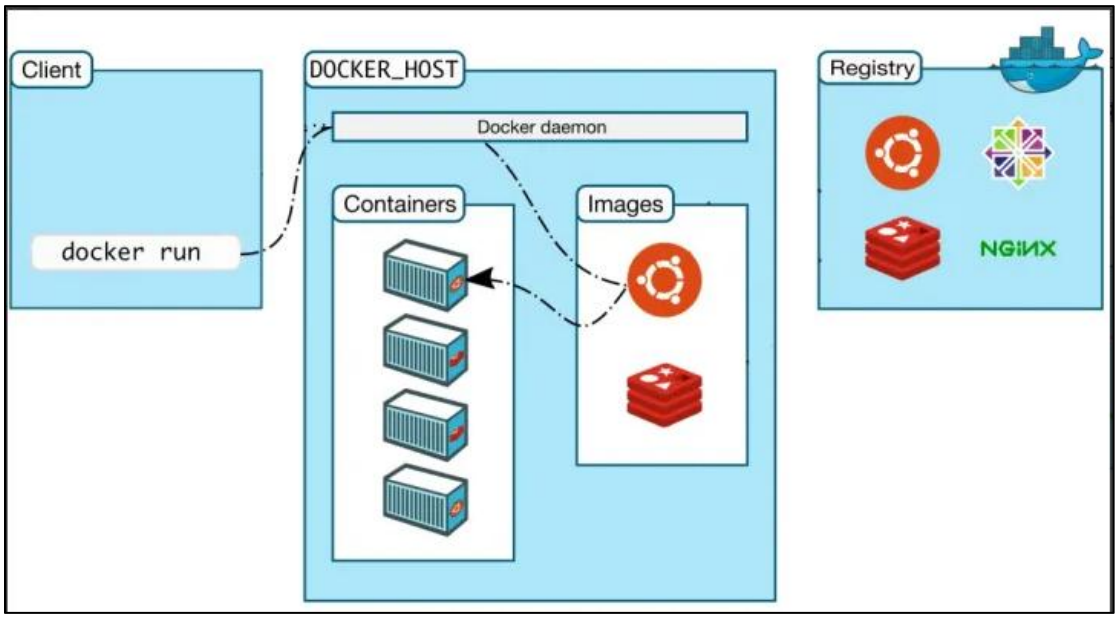
- **docker build**

当我们写完dockerfile交给docker“编译”时使用这个命令，那么client在接收到请求后转发给docker daemon，接着docker daemon根据dockerfile创建出“可执行程序”image。



- **docker run**

有了“可执行程序”image后就可以运行程序了，接下来使用命令docker run，docker daemon接收到该命令后找到具体的image，然后加载到内存开始执行，image执行起来就是所谓的容器（container）。



- **docker pull**

用户通过docker client发送命令，docker daemon接收到命令后向docker registry发送image下载请求，下载后存放在本地，这样就可以使用image了。

2.docker安装

官网安装参考手册：Install Docker Engine on Debian | Docker Docs

注意：树莓派默认出厂镜像已经安装好Docker，下面的步骤仅供学习参考。

这里使用阿里源进行下载

1) 按下“**Ctrl+Alt+T**”，打开命令行终端，输入“**sudo apt-get update**”，然后按下回车，更新 apt 软件包列表。

```
pi@raspberrypi:~ $ sudo apt-get update
Hit:1 https://mirrors.aliyun.com/docker-ce/linux/debian bookworm InRelease
Hit:2 http://archive.raspberrypi.com/debian bookworm InRelease
Hit:3 http://deb.debian.org/debian bookworm InRelease
Hit:4 http://deb.debian.org/debian-security bookworm-security InRelease
Hit:5 http://deb.debian.org/debian bookworm-updates InRelease
Reading package lists... Done
```

2) 输入指令“**sudo apt-get install ca-certificates curl gnupg**”安装所需的证书、工具和 GPG（GnuPG）密钥管理工具。

```
pi@raspberrypi:~ $ sudo apt-get install ca-certificates curl gnupg
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311).
curl is already the newest version (7.88.1-10+deb12u5).
gnupg is already the newest version (2.2.40-1.1).
gnupg set to manually installed.
The following packages were automatically installed and are no longer required:
  libslirp0 slirp4netns
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 101 not upgraded.
```

3) 输入指令“**sudo install -m 0755 -d /etc/apt/keyrings**”创建目录用于存储 GPG 密钥环。

```
pi@raspberrypi:~ $ sudo install -m 0755 -d /etc/apt/keyrings
```

4) 输入指令“**curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg**”从 Docker 官方获取 GPG 密钥并保存到密钥环中。

```
pi@raspberrypi:~ $ curl -fsSL https://download.docker.com/linux/debian/gpg | sud
o gpg --dearmor -o /etc/apt/keyrings/docker.gpg
File '/etc/apt/keyrings/docker.gpg' exists. Overwrite? (y/N) y
```

5) 输入指令“**sudo chmod a+r /etc/apt/keyrings/docker.gpg**”添加权限。

```
pi@raspberrypi:~ $ sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

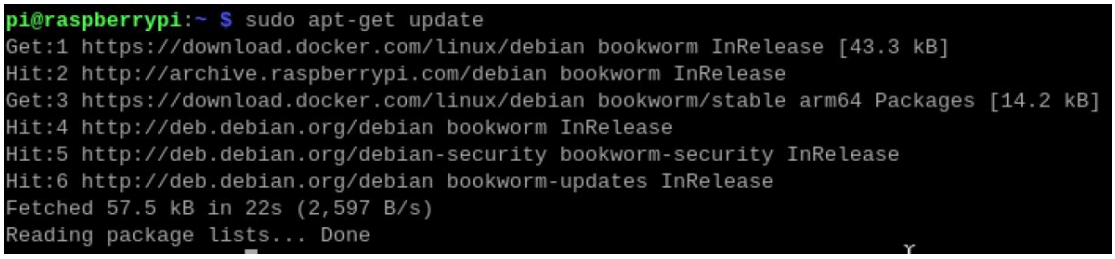
输入指令“**echo **

**"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian **

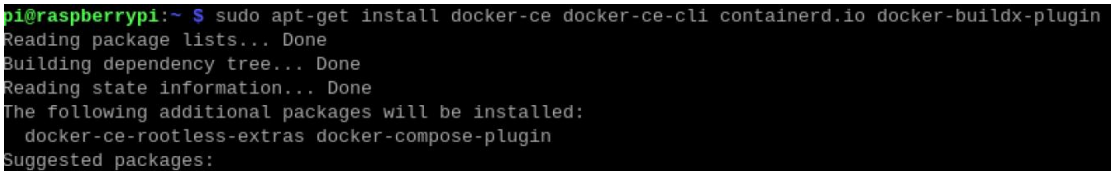
**$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | **

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null”将 Docker 存储库添加到 Apt 软件包源列表中。

6) 输入“**sudo apt-get update**”更新 apt 软件包列表。

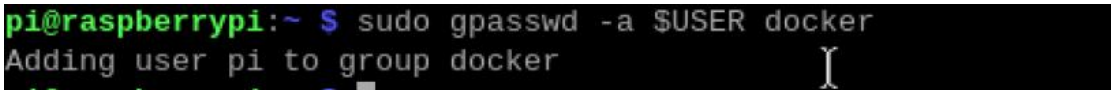


7) 输入“**sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin**”安装 Docker 及相关组件。

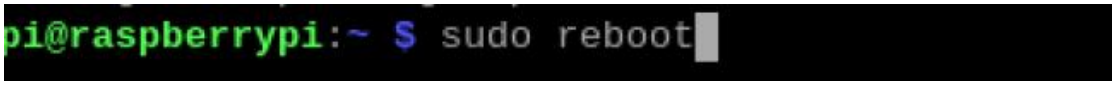


8) 输入“**sudo groupadd docker**” 创建 docker 用户组。

9) 输入“**sudo gpasswd -a $USER docker**”当前用户添加到这个 docker 用户组中，避免每次运行 Docker 命令时都要输入 sudo。



10) 输入“**sudo reboot**”重启树莓派。



11）下载完成之后，输入“**docker version**”查看 docker 版本。

```
pi@raspberrypi:~ $ docker version
Client: Docker Engine - Community
 Version:           25.0.0
 API version:       1.44
 Go version:        go1.21.6
 Git commit:        e758fe5
 Built:             Thu Jan 18 17:10:12 2024
 OS/Arch:           linux/arm64
 Context:           default

Server: Docker Engine - Community
 Engine:
  Version:          25.0.0
  API version:      1.44 (minimum version 1.24)
  Go version:       go1.21.6
  Git commit:       615dfdf
  Built:            Thu Jan 18 17:10:12 2024
  OS/Arch:          linux/arm64
  Experimental:     false
 containerd:
  Version:          1.6.27
  GitCommit:        a1496014c916f9e62104b33d1bb5bd03b0858e59
 runc:
  Version:          1.1.11
  GitCommit:        v1.1.11-0-g4bccb38
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

12）输入“**docker run hello-world**”，若出现以下内容则表示 Docker 安装成功。

```
pi@raspberrypi:~ $ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

第 2 课 Docker hello-world

Docker 官方提供的 hello-world 是一个轻量级的容器，用于测试 Docker 是否成功安装和运行。这个容器非常小，主要用于展示 Docker 的基本用法和验证 Docker 运行环境的正确性。

1) 按下“**Ctrl+Alt+T**”，打开命令行终端，输入“**docker run hello-world**”，Docker 将下载 hello-world 镜像（如果本地不存在）。

```
pi@raspberrypi:~ $ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

2) 在终端输入“**docker images**”列出本地主机上的所有镜像 。

```
pi@raspberrypi:~ $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    ee301c921b8a   8 months ago   9.14kB
```

在这里我们可以查看到“Hello_World”的相关信息，对应参数含义：

REPOSITORY： 镜像的仓库源

TAG:镜像的标签

IMAGE ID：镜像的ID

CREATED：镜像创建时间

SIZE：镜像大小

3） 在终端输入“**docker run -it hello-world**”运行hello-world镜像，终端会打印出相关的提示信息 。

```
pi@raspberrypi:~ $ docker run -it hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

4） 在终端输入“**docker inspect ee30**”，输出包含有关hello-world镜像详细信息的 JSON 数据。

```
pi@raspberrypi:~ $ docker inspect ee301c921b8a
[
    {
        "Id": "sha256:ee301c921b8aadc002973b2e0c3da17d701dcd994b606769a7e6eaa100b81d44",
        "RepoTags": [
            "hello-world:latest"
        ],
        "RepoDigests": [
            "hello-world@sha256:4bd78111b6914a99dbc560e6a20eab57ff6655aea4a80c50b0c5491968cbc2e6"
        ],
        "Parent": "",
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2023-05-02T16:49:27Z",
        "Container": "",
        "ContainerConfig": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": null,
            "Cmd": null,
            "Image": "",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null
```

第 3 课 Docker容器使用

1.帮助命令

docker info ：用于显示 Docker 系统的信息，包括 Docker 的配置和当前状态。它提供了有关 Docker 镜像、容器数、守护进程、存储驱动、网络和其他相关信息的详细概要。

docker --help： 用于获取 Docker 命令的帮助信息。执行此命令会列出所有可用 Docker 命令及其选项，以供参考。

2. 镜像命令

1) docker images：列出本地主机上的所有镜像 。

```
pi@raspberrypi:~ $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    ee301c921b8a   8 months ago   9.14kB
```

对应参数含义：

REPOSITORY： 镜像的仓库源

TAG:镜像的标签

IMAGE ID：镜像的ID

CREATED：镜像创建时间

SIZE：镜像大小

（ 同一个仓库源可以有多个TAG，代表这个仓库源的不同版本，使用REPOSITORY：TAG定义不同的镜像，如果不定义镜像的标签版本，docker将默认使用 lastest镜像）

该命令还提供了一些可选项

-a： 列出本地所有镜像

-q： 只显示镜像id

--digests： 显示镜像的摘要信息

2） docker pull：从选定的仓库下载镜像

```
pi@raspberrypi:~ $ docker pull --help

Usage:  docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Download an image from a registry

Aliases:
  docker image pull, docker pull

Options:
  -a, --all-tags                Download all tagged images in the repository
      --disable-content-trust   Skip image verification (default true)
      --platform string         Set platform if server is multi-platform capable
  -q, --quiet                   Suppress verbose output
```

使用方法 ：

docker pull 镜像仓库地址/镜像名：版本

示例（无法下载，仅做参考）： docker pull hiwonder/ros-foxy:1.0.0

例如下载debian镜像,按下“**Ctrl+Alt+T**”，打开命令行终端，输入“**docker pull debian**”，然后按下回车。

```
pi@raspberrypi:~ $ docker pull debian
Using default tag: latest
latest: Pulling from library/debian
5665c1f9a9e1: Pull complete
Digest: sha256:b16cef8cbcb20935c0f052e37fc3d38dc92bfec0bcfb894c328547f81e932d67
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
```

输入“**docker images**”，查看主机上的镜像，可以看到这里多了一个“**debian**”的镜像。

```
pi@raspberrypi:~ $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
debian        latest    60e0ade36b1a   11 days ago    139MB
hello-world   latest    ee301c921b8a   8 months ago   9.14kB
```

3） docker search：搜索对应DockerHub仓库中的镜像。

例如搜索ROS镜像，按下“**Ctrl+Alt+T**”，打开命令行终端，输入“**docker search ros**”，然后按下回车。

```
pi@raspberrypi:~ $ docker search ros
NAME                                              DESCRIPTION                                     STARS     OFFICIAL
ros                                               The Robot Operating System (ROS) is an open …  621       [OK]
osrf/ros                                          The Robot Operating System (ROS) is an open …  151
rostlab/pp1cs                                     Dockerimage for the exercise of Protein Pred…  0
osrf/ros2                                         **Experimental** Docker Images for ROS2 deve…  71
rosaenlg/server                                   RosaeNLG node.js Server                         0
rosaenlg/cli                                      Command Line Interface for RosaeNLG             0
rosaenlg/server-java                              RosaeNLG Java Server                            0
osrf/ros_legacy                                   Legacy docker image archive for Robot Operat…  3
rostlab/loctree3                                  https://github.com/Rostlab/LocTree3             0
rostlab/somena                                    https://github.com/Rostlab/someNA               0
rostlab/aquaria                                   Aquaria automatic build                         0
rostlab/seqvec                                    Modelling the Language of Life - Deep Learni…  0
uobflightlabstarling/rosbridge-suite                                                              0
rostlab/bio_embeddings                                                                            0
osrf/ros2_packaging_aarch64_jammy_39_snapshot                                                     2
osrf/ros2_packaging_jammy_39_snapshot                                                             0
osrf/ros2_batch_ci_aarch64_jammy_39_snapshot                                                      0
osrf/ros2_batch_ci_jammy_39_snapshot                                                              0
rostlab/ncov_casp_backend                                                                         0
uobflightlabstarling/ros-web-bridge                                                               0
uobflightlabstarling/ros_debug                                                                    0
uobflightlabstarling/ros_demo                                                                     0
ardupilot/ardupilot-dev-ros                       ArduPilot CI image for ROS2 build               0
osrf/space-ros                                    Docker images for the Space ROS project http…  1
osrf/icra2023_ros2_gz_tutorial                    Docker image for the ICRA London tutorial on…  2
```

该命令还提供了一些可选项

```
pi@raspberrypi:~ $ docker search --help

Usage:  docker search [OPTIONS] TERM

Search Docker Hub for images

Options:
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print search using a Go template
      --limit int       Max number of search results
      --no-trunc        Don't truncate output
```

-f, --filter filter: 根据提供的条件过滤输出结果。

--format string: 使用 Go 模板格式化输出。

--limit int: 限制搜索结果的最大数量。

--no-trunc: 不截断输出。

4) docker rmi： 删除镜像

docker rmi -f [镜像id] ： 删除单个镜像

docker rmi -f [镜像名:tag] [镜像名:tag] ： 删除多个镜像

docker rmi -f $(docker images -qa) ： 删除全部镜像

3.容器命令

有镜像才能创建容器，这里使用 debian 的镜像来测试，按下“**Ctrl+Alt+T**”，打开命令行终端，输入“**docker pull debian**”，然后按下回车。

```
pi@raspberrypi:~ $ docker pull debian
Using default tag: latest
latest: Pulling from library/debian
5665c1f9a9e1: Pull complete
Digest: sha256:b16cef8cbcb20935c0f052e37fc3d38dc92bfec0bcfb894c328547f81e932d67
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
```

3.1 docker run 运行镜像启动容器

命令参数说明 ：docker run [OPTIONS] IMAGE [COMMAND][ARG...]

常用参数：

--name="Name" ：给容器指定一个名字

-d ：后台方式运行容器，并返回容器的id!

-i ：以交互模式运行容器，通过和 -t 一起使用

-t ：给容器重新分配一个终端，通常和 -i 一起使用

-P ：随机端口映射（大写）

-p ：指定端口映射（小写），一般可以有四种写法

ip:hostPort:containerPort

ip::containerPort

hostPort:containerPort (常用)

这里以运行debian镜像为例，在命令行终端，输入“**docker run -it debian /bin/bash**”，然后按下回车。

pi@raspberry：是主机系统界面，root@9aeb4cc795e6：则是在docker容器内界面。

```
pi@raspberrypi:~ $ docker run -it debian /bin/bash
root@9aeb4cc795e6:/#
```

参数说明：

-i：交互式操作。

-t：终端。

debian ：**debian** 镜像。

/bin/bash：放在镜像名后的是命令，这里我们希望有个交互式 Shell，因此用的是/bin/bash。

要退出终端，直接在终端输入“**exit**”，按下回车。

```
pi@raspberrypi:~ $ docker run -it debian /bin/bash
root@9aeb4cc795e6:/# exit
exit
pi@raspberrypi:~ $
```

3.2 docker ps 列出所有运行的容器

命令参数说明 ：docker ps [OPTIONS]

常用参数说明 ：

-a ：列出当前所有正在运行的容器 + 历史运行过的容器

-l ：显示最近创建的容器

-n=? ：显示最近n个创建的容器

-q ：静默模式，只显示容器编号。

在命令行终端，输入“**docker ps -a**”，然后按下回车，显示出正在运行和历史运行过的容器。其中container id是容器的ID，image是该容器使用的镜像名称，created是容器创建时间，status是容器当前状态

```
pi@raspberrypi:~ $ docker ps -a
CONTAINER ID   IMAGE         COMMAND       CREATED             STATUS                         PORTS     NAMES
9aeb4cc795e6   debian        "/bin/bash"   10 minutes ago      Exited (0) 8 minutes ago                 intelligent_ramanujan
e3fd00242b77   hello-world   "/hello"      About an hour ago   Exited (0) About an hour ago             infallible_dirac
20b65a5ef772   hello-world   "/hello"      2 hours ago         Exited (0) 2 hours ago                   romantic_pasteur
```

3.3 退出容器

一共有两种退出容器的指令：

1） 直接在终端输入“**exit**”，按下回车，此时容器会停止运行并退出。

```
pi@raspberrypi:~ $ docker run -it debian /bin/bash
root@9aeb4cc795e6:/# exit
exit
pi@raspberrypi:~ $
```

2） 使用快捷键组合“**crtl+P+Q**”，此时容器会直接退出但不停止运行，我们可以在终端输入指令“**docker ps**”,查看到正在运行的容器。

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
56d746b94607   debian    "/bin/bash"   7 minutes ago   Up 7 minutes             optimistic_kirch
```

3.4 多终端进入正在运行的容器

用两个指令可以进入其他终端正在运行的容器：

docker attach ：直接进入容器启动命令的终端，不会启动新的进程，如果从这个容器退出，会导致容器停止运行。

docker exec： 是在容器中打开新的终端，并且可以启动新的进程，如果从这个容器退出，容器还可以正常运行，一般使用该方法进入容器。

3.4.1 docker attach

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker ps**”，显示当前

正在运行的容器。

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS              PORTS     NAMES
004c7b83e76c   debian    "/bin/bash"   9 seconds ago        Up 6 seconds                  quirky_mccarthy
cd1df225f0bf   debian    "/bin/bash"   About a minute ago   Up About a minute             nostalgic_black
```

2） 在终端输入“**docker attach 004c**”（容器的id可以简写，只要是该容器唯一标识就行），进入正在运行的容器中。

```
pi@raspberrypi:~ $ docker attach 004c
root@004c7b83e76c:/#
```

3） 在终端输入“**exit**”，按下回车，此时容器会停止运行并退出。

```
pi@raspberrypi:~ $ docker attach 004c
root@004c7b83e76c:/# exit
exit
```

4） 在终端输入“**docker ps**”，显示当前正在运行的容器，可以看到刚才退出容器的同时也关闭了容器。

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
cd1df225f0bf   debian    "/bin/bash"   6 minutes ago   Up 6 minutes             nostalgic_black
```

3.4.2 docker exec

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker ps**”，显示当前正在运行的容器。

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
efe969b704a7   debian    "/bin/bash"   8 seconds ago   Up 5 seconds             fervent_williams
cd1df225f0bf   debian    "/bin/bash"   8 minutes ago   Up 7 minutes             nostalgic_black
```

2） 在终端输入“**docker exec -it efe9 /bin/bash**”（容器的id可以简写，只要是该容器唯一标识就行），进入正在运行的容器中。

```
pi@raspberrypi:~ $ docker exec -it efe9 /bin/bash
```

3） 在终端输入“**exit**”，按下回车，此时会退出容器。

```
pi@raspberrypi:~ $ docker exec -it efe9 /bin/bash
root@efe969b704a7:/# exit
exit
```

4） 在终端输入“**docker ps**”，显示当前正在运行的容器，可以看到刚才退出容器后，容器并没有关闭。

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
efe969b704a7   debian    "/bin/bash"   3 minutes ago    Up 3 minutes              fervent_williams
cd1df225f0bf   debian    "/bin/bash"   11 minutes ago   Up 11 minutes             nostalgic_black
```

3.5启动停止容器

docker start [容器id or 容器名] ：启动容器

docker restart [容器id or 容器名] ：重启容器

docker stop [容器id or 容器名] ：停止容器

docker kill [容器id or 容器名] ：强制停止容器

3.6 删除容器

docker rm [容器id] ：删除指定容器

docker rm -f $(docker ps -a -q) ：删除所有容器

docker ps -a -q|xargs docker rm ：删除所有容器

4.常用其他命令

4.1 查看容器中运行的进程信息

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker ps**”，显示当前正在运行的容器。

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
efe969b704a7   debian    "/bin/bash"   8 seconds ago    Up 5 seconds              fervent_williams
cd1df225f0bf   debian    "/bin/bash"   8 minutes ago    Up 7 minutes              nostalgic_black
```

2） 在终端输入“**docker top efe9**”，显示容器内运行的进程信息。

```
pi@raspberrypi:~ $ docker top efe9
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
root                3186                3163                0
18:01               pts/0               00:00:00            /bin/bash
```

相关参数具体解释如下：

UID：进程的用户标识符（User ID）。

PID：进程的进程标识符（Process ID）。

PPID：父进程的进程标识符（Parent Process ID）。

C：进程的 CPU 利用率。

STIME：进程的启动时间。

TTY：与进程关联的终端设备。

TIME：进程的累计 CPU 使用时间。

CMD：进程的命令行。

4.2 查看容器/镜像的元数据

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker ps**”，显示当前正在运行的容器。

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
efe969b704a7   debian    "/bin/bash"   8 seconds ago   Up 5 seconds             fervent_williams
cd1df225f0bf   debian    "/bin/bash"   8 minutes ago   Up 7 minutes             nostalgic_black
```

2） 在终端输入“**docker inspect efe9**”，输出包含有关容器或镜像详细信息的 JSON 数据。


pi@raspberrypi:~ $ docker inspect efe9
[
{
"Id": "efe969b704a7bbbc02a8ae84c05993e89f70fe86b9a6e53df5933c99f0a0d637",
"Created": "2024-01-22T10:01:09.554324422Z",
"Path": "/bin/bash",
"Args": [],
"State": {
"Status": "running",
"Running": true,
"Paused": false,
"Restarting": false,
"OOMKilled": false,
"Dead": false,
"Pid": 3186,
"ExitCode": 0,
"Error": "",
"StartedAt": "2024-01-22T10:01:11.688664835Z",
"FinishedAt": "0001-01-01T00:00:00Z"
},
"Image": "sha256:60e0ade36b1ab9227b0b8fc5a9731a3a5b9217e9feeb4ba7a2f8cb84addef87b",
"ResolvConfPath": "/var/lib/docker/containers/efe969b704a7bbbc02a8ae84c05993e89f70fe86b9a6e53df5933c99f0a0d637/resolv.
conf",
"HostnamePath": "/var/lib/docker/containers/efe969b704a7bbbc02a8ae84c05993e89f70fe86b9a6e53df5933c99f0a0d637/hostname"
,
"HostsPath": "/var/lib/docker/containers/efe969b704a7bbbc02a8ae84c05993e89f70fe86b9a6e53df5933c99f0a0d637/hosts",
"LogPath": "/var/lib/docker/containers/efe969b704a7bbbc02a8ae84c05993e89f70fe86b9a6e53df5933c99f0a0d637/efe969b704a7bb
bc02a8ae84c05993e89f70fe86b9a6e53df5933c99f0a0d637-json.log",
"Name": "/fervent_williams",

第 4 课 Docker镜像深入理解和制作

1.镜像的理解

1、镜像是一种轻量级、可执行的独立软件包，它包含运行某个软件所需要的所有内容。我们将应用程序、配置打包成一个成型的、可交付、可部署的运行环境，包括代码、运行时所需要的库、环境变量和配置文件等，这个大包好的运行环境就是image镜像文件。

2、只有通过镜像文件才能生成docker容器实例。

2.UnionFS（联合文件系统）

1、Union文件系统（UnionFS）是一种分层的、轻量级的、高性能的文件系统，它是docker镜像的基础，并且支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂在到同一个虚拟文件系统下。

2、镜像可以通过分层来进行继承，基于基础镜像，可以制作各种具体的应用镜像。

Union文件系统的特性：一次性同时加载多个文件系统，但是从外面来看，只能看到一个文件系统；联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有分层的文件和目录。

3.镜像分层

下载一个镜像时，注意观察下载的日志输出，可以看到是一层一层的在下载：

```
pi@raspberrypi:~ $ docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
6988ac25ab22: Pull complete
cd2cec5cf6db: Pull complete
e703d567fd0e: Pull complete
de5675f5f5a6: Pull complete
28476d81f322: Pull complete
ccea770348da: Pull complete
842d004c6831: Pull complete
37f7eaf29320: Pull complete
03498da0acb4: Pull complete
00456c5c10e4: Pull complete
Digest: sha256:d7c20c5ba268c558f4fac62977f8c7125bde0630ff8946b08dde44135ef40df3
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest
```

可以通过在终端输入指令“docker image inspect mysql”查看镜像分层。

```
pi@raspberrypi:~ $ docker image inspect mysql
[
    {
        "Id": "sha256:b068ea59c6773ae58f27b4f289fecbb12cc416ba2d839736f8c62ad93d43fd49",
        "RepoTags": [
            "mysql:latest"
        ],
        "RepoDigests": [
            "mysql@sha256:d7c20c5ba268c558f4fac62977f8c7125bde0630ff8946b08dde44135ef40df3"
        ],
```

```
        "Layers": [
            "sha256:486ffb2b78bfe1b7de0674b28ecea6dfe79457f6fa4d03bd36d0879aeee68aa9",
            "sha256:df476ce8a77c66711cd397c3940177979afbdb8eb46153b0f7abd0525de9c519",
            "sha256:8fd3e8a3a23536b6cac42d2fa4ae230c99f84c2872bb4142d63db87d0a835f8d",
            "sha256:8d2f103f0e6304a4fa1b5898625e03ace744676a68666e1038b59561083c2077",
            "sha256:4eb705e240740b35d1d257ad084e1019fe6cc3f2729a31edf6e73ba50c7b2aab",
            "sha256:e2fbbef5f179aae0314176e2fd0a3428f3655d8cf3a5c2402cd659fead82ea38",
            "sha256:7b15982fa391b892c657e20948ac01855ad202353b5269ddc367ba386d220d9f",
            "sha256:4e6b6bd5aa87ba4b2774f3f0a06bb2f7600ffba2b9d235fc1583a53edbb970a2",
            "sha256:bf624f50d72ed948606a5b2ad612d9f62ae0f0e0e889a34f84aaef1f73cbf815",
            "sha256:3e43f9bac43a4acbf89b5fe45b493b9693670bd0fa67b7d0f6f13e7838a44c1e"
        ]
```

3.1分层理解

所有的 docker 镜像都起始于一个基础镜像层，当进行修改或增加新的内容时，就会在当前镜像层之上，创建新的镜像层。

举一个简单的例子，假如基于debian创建一个新的镜像，这就是新镜像的第一层；如果在该镜像中添加 python包，就会在基础镜像层之上创建第二个镜像层；如果继续添加一个安全补丁，就会创建第三个镜像层，每一层一层的叠加。

docker镜像都是只读的，当容器启动时，一个新的可写层被加载到镜像的顶部，这一层就是我们通常说的容器层，容器之下的都叫镜像层。

3.2 docker镜像采用分层的好处

资源共享，比如有多个镜像都从相同的Base镜像构建而来，那么宿主机只需在磁盘上保留一份base镜像，同时内存中也只需要加载一份base镜像，这样就可以为所有的容器服务了，而且镜像的每一层都可以被共享。

4.制作镜像

当我们从 docker 镜像仓库中下载的镜像不能满足我们的需求时，我们可以通过以下两种方式对镜像进行更改。

① 使用docker commit从已经创建的容器中更新镜像，并且提交这个镜像。

② 使用 Dockerfile 指令来创建一个新的镜像。

4.1 从容器中提交一个镜像

指令解释：docker commit -m="提交的描述信息" -a="作者" 容器ID要创建的目标镜像名:[标签名]【也可省略 -m -a 参数】

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker ps -a**”，显示所有运行过的容器，找到需要提交容器的ID。

```
pi@raspberrypi:~ $ docker ps -a
CONTAINER ID   IMAGE         COMMAND       CREATED          STATUS                      PORTS     NAMES
04b86cdec880   hello-world   "/hello"      28 minutes ago   Exited (0) 28 minutes ago             hardcore_grothendieck
f4b0d06a2fe7   hello-world   "/bin/sh"     29 minutes ago   Created                               charming_blackburn
067b778e4486   hello-world   "/bin/bash"   30 minutes ago   Created                               reverent_wiles
f16dc4876948   debian        "/bin/bash"   31 minutes ago   Exited (0) 30 minutes ago             practical_hodgkin
6c8e3babad11   hello-world   "/bin/bash"   31 minutes ago   Created                               happy_jennings
a450933f3aa0   hello-world   "/bin/bash"   32 minutes ago   Created                               flamboyant_mayer
eab350ee35da   hello-world   "/bin/bash"   34 minutes ago   Created                               nifty_edison
d2408c2fb32f   hello-world   "/hello"      41 minutes ago   Exited (0) 41 minutes ago             dreamy_spence
efe969b704a7   debian        "/bin/bash"   3 hours ago      Exited (137) 33 minutes ago           fervent_williams
004c7b83e76c   debian        "/bin/bash"   3 hours ago      Exited (0) 3 hours ago                quirky_mccarthy
cd1df225f0bf   debian        "/bin/bash"   3 hours ago      Exited (137) 33 minutes ago           nostalgic_black
109081ef693e   debian        "/bin/bash"   3 hours ago      Exited (255) 3 hours ago              xenodochial_cartwright
dccc090bee5b   debian        "/bin/bash"   3 hours ago      Exited (0) 3 hours ago                gifted_poincare
56d746b94607   debian        "/bin/bash"   3 hours ago      Exited (255) 3 hours ago              optimistic_kirch
2672149d25ba   debian        "bash"        3 hours ago      Exited (0) 3 hours ago                great_wilson
9aeb4cc795e6   debian        "/bin/bash"   4 hours ago      Exited (0) 4 hours ago                intelligent_ramanujan
```

2） 在终端输入“**docker commit 9aeb4cc795e6 debain:1.0**”，提交容器ID 9aeb4cc795e6的镜像并命名为debian，标签名为1.0。

```
pi@raspberrypi:~ $ docker commit 9aeb4cc795e6 debain:1.0
sha256:965cecd53c559bcce980346a3439f01555b61396ccc3b023323d77a6cb64096d
```

3） 在终端输入“**docker images**”，查看主机中的镜像，可以看到现在多了一个debain镜像。

```
pi@raspberrypi:~ $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
debain        1.0       965cecd53c55   5 minutes ago    139MB
test/debian   v1        7f96d767c7be   16 hours ago     139MB
mysql         latest    b068ea59c677   4 days ago       638MB
debian        latest    60e0ade36b1a   12 days ago      139MB
hello-world   latest    ee301c921b8a   8 months ago     9.14kB
```

4.2 dockerfile制作镜像

指令解释：docker build -f dockerfile 文件路径 -t 新镜像名字:TAG . # docker build 命令最后有一个 . 表示当前目录

使用命令 docker build，从零开始来创建一个新的镜像。为此，我们需要创建一个 Dockerfile 文件，其中包含一组指令来告诉 Docker 如何构建我们的镜像。

关于dockerfile的编写请参考：https://docs.docker.com/develop/develop images/dockerfile_best-practices/

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**vim Dockerfile**”，创建并打开文件。

```
pi@raspberrypi:~ $ vim Dockerfile
```

2） 在终端输入下图内容，按下“**ESC**”键，输入“**：wq**”退出并保存文件。

```
File  Edit  Tabs  Help
FROM    debian
MAINTAINER      Fisher "fisher@sudops.com"

RUN     /bin/echo 'root:123456' |chpasswd
RUN     useradd runoob
RUN     /bin/echo 'runoob:123456' |chpasswd
RUN     /bin/echo -e "LANG=\"en_US.UTF-8\"" >/etc/default/local
EXPOSE  22
EXPOSE  80
CMD     /usr/sbin/sshd -D
~
```

3） 在终端输入“**docker build -f dockerfile -t debain:1.1 .** ”，构建一个新镜像。

```
pi@raspberrypi:~ $ docker build -f Dockerfile -t debian:1.1 .
[+] Building 14.3s (9/9) FINISHED                                  docker:default
 => [internal] load build definition from Dockerfile                        0.2s
 => => transferring dockerfile: 318B                                        0.0s
 => [internal] load metadata for docker.io/library/debian:latest            0.0s
 => [internal] load .dockerignore                                           0.3s
 => => transferring context: 2B                                             0.0s
 => [1/5] FROM docker.io/library/debian:latest                              0.5s
 => [2/5] RUN     /bin/echo 'root:123456' |chpasswd                         2.5s
 => [3/5] RUN     useradd runoob                                            2.8s
 => [4/5] RUN     /bin/echo 'runoob:123456' |chpasswd                       2.5s
 => [5/5] RUN     /bin/echo -e "LANG="en_US.UTF-8"" >/etc/default/local     2.9s
 => exporting to image                                                      1.2s
 => => exporting layers                                                     1.2s
 => => writing image sha256:30555f5a4f74883ee7441379af1bc9230163721edf4e4   0.0s
 => => naming to docker.io/library/debian:1.1                               0.0s
```

4） 在终端输入“**docker images** ”，可以看到我们刚刚构建的新镜像。

```
pi@raspberrypi:~ $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
debian        1.1       30555f5a4f74   19 minutes ago   139MB
test/debian   v1        7f96d767c7be   17 hours ago     139MB
mysql         latest    b068ea59c677   4 days ago       638MB
debian        latest    60e0ade36b1a   12 days ago      139MB
ros           melodic   3365d517511a   6 weeks ago      1.2GB
hello-world   latest    ee301c921b8a   8 months ago     9.14kB
```

第 5 课 Docker硬件交互和数据处理

1.硬件挂载（端口绑定）

将自己需要挂载的设备接到主板上，在宿主机中建立udev规则（/etc/udev/rules.d/）

以下是以USB摄像头挂载为例，实际情况根据自己的设备挂载 。

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入

“**docker run -it --device=/dev/video0 debian:latest /bin/bash**”，启动debian容器并将USB摄像头挂载。

```
pi@raspberrypi:~ $ docker run -it --device=/dev/video0 debian:latest /bin/bash
```

2） 在终端输入“**ls /dev/**”，按下回车，查看当前容器下挂载的设备，可以看到刚刚的USB摄像头设备“**video0**”已经挂载上了。

```
pi@raspberrypi:~ $ docker run -it --device=/dev/video0 debian:latest /bin/bash
root@80ead1e0089b:/# ls /dev/
console  fd  full  mqueue  null  ptmx  pts  random  shm  stderr  stdin  stdout  tty  urandom  video0  zero
```

2. docker中GUI的显示

在宿主机中安装：

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入

“**sudo apt-get install tigervnc-standalone-server tigervnc-viewer**”，按下回车，输入“**y**”进行下载。

```
pi@raspberrypi:~ $ sudo apt-get install tigervnc-standalone-server tigervnc-viewer
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libfile-readbackwards-perl libfltk-images1.3 tigervnc-common tigervnc-tools xfonts-base
Suggested packages:
  xfonts-100dpi | xfonts-75dpi xfonts-scalable
The following NEW packages will be installed:
  libfile-readbackwards-perl libfltk-images1.3 tigervnc-common tigervnc-standalone-server tigervnc-tools tigervnc-viewer
  xfonts-base
0 upgraded, 7 newly installed, 0 to remove and 108 not upgraded.
Need to get 7,294 kB of archives.
After this operation, 11.9 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

2） 在终端输入“sudo apt-get install x11-xserver-utils ”，按下回车，进行下载。

```
pi@raspberrypi:~ $ sudo apt-get install x11-xserver-utils
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
x11-xserver-utils is already the newest version (7.7+9+b1).
x11-xserver-utils set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 108 not upgraded.
```

3） 在终端输入“**xhost +** ”，按下回车，进行下载。

```
pi@raspberrypi:~ $ xhost +
access control disabled, clients can connect from any host
```

4) 在终端输入

“docker run -it --env="DISPLAY" --env="QT_X11_NO_MITSHM=1" -v /tmp/.X11-unix:/tmp/.X11-unix debian:latest /bin/bash”，按下回车，进入容器。

```
pi@raspberrypi:~ $ docker run -it --env="DISPLAY" --env="QT_X11_NO_MITSHM=1" -v /tmp/.X11-unix:/tmp/.X11-unix debian:latest /bin/bash
root@2faedbd22176:/#
```

5） 在终端输入“**apt-get update** ”，按下回车，更新软件列表。

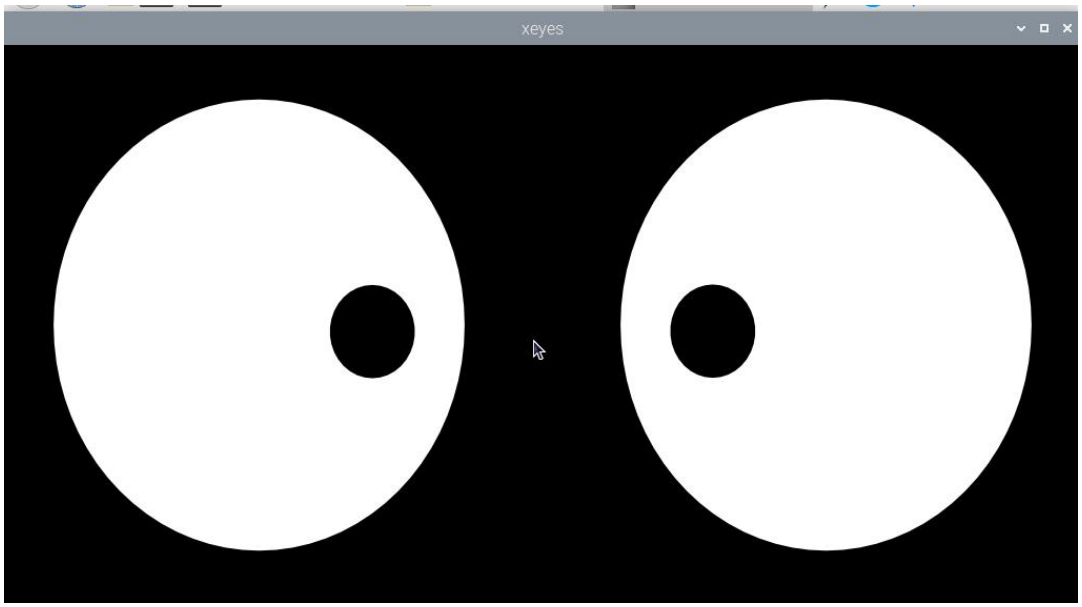
```
root@2faedbd22176:/# apt-get update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [52.1 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main arm64 Packages [8685 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main arm64 Packages [12.5 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main arm64 Packages [132 kB]
Fetched 9080 kB in 5s (1757 kB/s)
Reading package lists... Done
root@2faedbd22176:/# apt install -y x11-apps
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

6） 在终端输入“**apt-get install -y x11-apps** ”，按下回车，安装X11图像界面。

```
root@2faedbd22176:/# apt install -y x11-apps
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bsdextrautils fontconfig-config fonts-dejavu-core groff-base libbrotli1 libbsd0 libexpat1 libfontconfig1 libfreetype6
  libgdbm6 libice6 libpipeline1 libpng16-16 libsm6 libuchardet0 libx11-6 libx11-data libx11-xcb1 libxau6 libxaw7
  libxcb-damage0 libxcb-present0 libxcb-xfixes0 libxcb1 libxcursor1 libxdmcp6 libxext6 libxfixes3 libxft2 libxi6 libxkbfile1
  libxmu6 libxmuu1 libxpm4 libxrender1 libxt6 man-db x11-common xbitmaps
```

7） 在终端输入“**xeyes** ”，按下回车，会出现一双眼睛，它会跟着鼠标移动。

```
root@2faedbd22176:/# xeyes
```



3. docker容器和宿主机互传文件

3.1 使用cp指令拷贝文件

3.1.1 从容器内拷贝文件到主机上

指令：docker cp [容器id:容器内路径] [目的主机路径]

1） 在终端输入“**docker run -it debian /bin/bash**”，进入容器。

```
pi@raspberrypi:~ $ docker run -it debian /bin/bash
root@d2d0e43b41eb:/#
```

2） 在终端输入“**cd**”，切换到用户的主目录。

```
root@d2d0e43b41eb:/# cd
```

3） 在终端输入“**ls**”，查看路径下的目录，这里可以看到在该路径下是一个空目录。

```
root@d2d0e43b41eb:~# ls
```

4） 在终端输入“**touch test.txt**”，创建一个txt文件。

```
root@d2d0e43b41eb:~# touch test.txt
```

5） 在终端输入“**pwd**”，查看当前路径。

```
root@d2d0e43b41eb:~# pwd
/root
```

6） 按下“crtl+P+Q”，退出容器但不停止。

7） 在终端输入“**docker cp d2d0e43b41eb:/root/test.txt ~/**”，将刚刚创建好的test.txt文件复制到宿主机中。

```
pi@raspberrypi:~ $ docker cp d2d0e43b41eb:/root/test.txt ~/
Successfully copied 1.54kB to /home/pi/
```

8） 在终端输入“**cd /home/pi**”，进入到文件路径下。

```
pi@raspberrypi:~ $ cd /home/pi/
```

9） 在终端输入“**ls**”，可以看到test.txt已经在路径下了。

```
pi@raspberrypi:~ $ ls
Bookshelf       debian.tar  Dockerfile  Downloads  Pictures   Templates  Videos
debian_1.1.tar  Desktop     Documents   Music      Public     test.txt
```

3.1.2 从宿主机拷贝文件到容器上

指令：docker cp [宿主机文件路径] [容器id:容器内路径]

1） 在终端输入“**touch test_1.txt**”，创建test_1.txt文件。

```
pi@raspberrypi:~ $ touch test_1.txt
```

2） 在终端输入“**docker cp test_1.txt d2d0e43b41eb:/root/**”，将test_1.txt文件复制到容器d2d0e43b41eb的/root/路径下（注意：这里的容器d2d0e43b41eb需要保持运行状态）。

```
pi@raspberrypi:~ $ docker cp test_1.txt d2d0e43b41eb:/root/
Successfully copied 1.54kB to d2d0e43b41eb:/root/
```

3） 在终端输入“**docker exec -it d2d0e43b41eb /bin/bash**”，进入容器。

```
pi@raspberrypi:~ $ docker exec -it d2d0e43b41eb /bin/bash
```

4） 在终端输入“**cd**”，切换到用户的主目录。

```
root@d2d0e43b41eb:/# cd
```

5） 在终端输入“**ls**”，查看路径下的目录，这里可以看到在文件已经从宿主机复制到容器中了。

```
root@d2d0e43b41eb:~# ls
test.txt  test_1.txt
```

4.使用数据卷

4.1 数据卷概述

将应用和运行的环境打包形成容器运行，运行可以伴随着容器，但是我们对于数据的要求是希望能够持久化的。就好比，安装一个mysql，结果把容器删了，就相当于删库跑路了，这肯定不行。所以我们希望容器之间有可能可以共享数据，docker容器产生的数据，如果不通过docker commit 生成新的镜像，使得数据作为镜像的一部分保存下来，那么当容器删除后，数据自然也就没有了。

为了能保存数据在docker中我们就可以使用数据卷。让数据挂载到我们本地，这样数据就不会因为容器删除而丢失了。

特点：

1、数据卷可在容器之间共享或重用数据。

2、数据卷中的更改可以直接生效。

3、数据卷中的更改不会包含在镜像的更新中。

4、数据卷的生命周期一直持续到没有容器使用它为止。

4.2 数据卷使用

指令：docker run -it -v [宿主机绝对路径目录:容器内目录] [镜像名]

1） 在终端输入

“**docker run -it -v /home/pi/temp:/root/temp debian:latest /bin/bash**”，将宿主机中的/home/jetson/temp目录和容器内的/root/temp目录就可以共享数据了。

```
pi@raspberrypi:~ $ docker run -it -v /home/pi/temp:/root/temp debian:latest /bin/bash
root@46de34b3ffb7:/# ls
```

2） 在终端输入“**cd root/temp/**”，进入指定目录下。

```
root@46de34b3ffb7:/# cd root/temp/
```

3） 在终端输入“**touch text_2.txt**”，创建一个touch text_2.txt文件。

```
root@46de34b3ffb7:~/temp# touch text_2.txt
```

4） 在终端输入“**exit**”，退出容器。

```
root@46de34b3ffb7:~/temp# exit
exit
pi@raspberrypi:~ $
```

5） 在终端输入“**cd /home/pi/temp/**”，进入指定目录下。

```
pi@raspberrypi:~ $ cd /home/pi/temp/
pi@raspberrypi:~/temp $
```

6） 在终端输入“**ls**”，可以看到在容器中创建的文件已经共享到指定目录下。

```
pi@raspberrypi:~/temp $ ls
text_2.txt
```

第 6 课 容器的日常管理

Docker已经成为了现代应用程序开发和部署的核心工具之一。然而，为了确保容器环境的稳定性和可靠性，日常维护和故障排除是必不可少的任务。本文将介绍一些关键的Docker容器维护和故障排除技巧，以帮助大家应对各种常见问题。

1.定期清理无用容器和镜像

在长时间的使用过程中，Docker主机上可能会积累大量无用的容器和镜像，占用宝贵的磁盘空间。定期清理无用资源是容器环境维护的第一步。

按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker container prune**”，按下回车，输入“**y**”开始清理无用的停止容器。

```
pi@raspberrypi:~ $ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
46de34b3ffb779928582ffc909c13dd9568aff1bcbe2d56099d1029419277f76
```

按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker image prune**”，按下回车，输入“**y**”开始清理无用的镜像。

```
pi@raspberrypi:~ $ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
```

2.监控容器性能

监控容器的性能是确保容器正常运行的重要任务。使用Docker自带的stats命令可以实时查看容器的CPU、内存、网络和磁盘使用情况。

按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker stats 0cfc6a98a68c**”，按下回车， 查看ID 0cfc6a98a68c 的容器实时性能数据。

```
pi@raspberrypi:~ $ docker stats 0cfc6a98a68c
```

```
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT   MEM %     NET I/O       BLOCK I/O   PIDS
0cfc6a98a68c   jolly_edison  0.00%     0B / 0B             0.00%     4.98kB / 0B   0B / 0B     1
```

3.查看容器日志

容器的日志记录对于排查问题和诊断故障非常重要。使用docker logs命令可以查看容器的标准输出日志。

按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker logs 0cfc6a98a68c**”，按下回车， 查看ID 0cfc6a98a68c 的容器的标准输出日志。

```
pi@raspberrypi:~ $ docker logs 0cfc6a98a68c
root@0cfc6a98a68c:/# exit
exit
```

4.重启容器

容器在运行过程中可能会出现各种问题，包括应用程序崩溃或资源耗尽。在这种情况下，重启容器可能是解决问题的一种有效方法。

按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker restart 0cfc6a98a68c**”，按下回车， 重启容器。

```
pi@raspberrypi:~ $ docker restart 0cfc6a98a68c
```

5.处理容器网络问题

容器之间的网络问题可能会导致应用程序无法通信。使用docker network命令可以管理容器网络，查看容器的IP地址和端口映射情况。

按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 0cfc6a98a68c**”，

按下回车，查看容器网络信息。

```
pi@raspberrypi:~ $ docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 0cfc6a98a68c
172.17.0.2
```

6.处理容器存储问题

容器的存储问题可能包括磁盘空间耗尽或数据丢失。使用数据卷和存储驱动程序可以更好地管理容器的存储。

按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker volume create my-data**”，按下回车，创建数据卷。

```
pi@raspberrypi:~ $ docker volume create my-data
my-data
```

创建成功后，你可以将这个数据卷挂载到容器中，以实现数据的共享。

```
pi@raspberrypi:~ $ docker run -v my-data:/path/in/container debian
```

总结

Docker容器的日常维护和故障排除是确保容器环境稳定性和可靠性的关键任务。通过定期清理无用资源、监控容器性能、查看容器日志、进入运行中容器、重启容器、构建自定义镜像、备份和恢复容器数据、处理容器网络和存储问题、以及使用容器编排工具等技巧，可以更好地管理和维护容器化的应用程序。

第 7 课 Docker安装ROS

注意：我们的树莓派镜像出厂时已经安装好ROS1/ROS2环境，这里只供学习参考。

1.ROS1安装

1.1 这里以melodic版本为例（需联网）

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker pull ros:melodic**”，下载ROS1镜像，镜像下载需要一些时间，请耐心等待一会。

```
pi@raspberrypi:~ $ docker pull ros:melodic
```

1） 镜像下载完成后，在终端输入“**docker run -it --network=host -d -v=/dev:/dev -v /tmp/.X11-unix:/tmp/.X11-unix --name melodic -e DISPLAY=${DISPLAY} --restart=always ros:melodic /bin/bash** ”，运行容器并指定名称为 melodic。

```
pi@raspberrypi:~ $ docker run -it --network=host -d -v=/dev:/dev -v /tmp/.X11-unix:/tmp/.X11-unix --name melodic -e DISPLAY=${
DISPLAY} --restart=always ros:melodic /bin/bash
75b6634e622762255058569dd769ed3925f77b9976539e85d1d70cd029929f9d
```

2） 在终端输入“**xhost +**”，开启 X Server 的访问控制。

```
pi@raspberrypi:~ $ xhost +
access control disabled, clients can connect from any host
```

3） 在终端输入“**docker ps -a**”查看新创建容器的ID。

```
pi@raspberrypi:~ $ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS                          PORTS     NAMES
75b6634e6227   ros:melodic    "/ros_entrypoint.sh …"   15 seconds ago      Up 13 seconds                             melodic
a24dbc912220   debian         "bash"                   About an hour ago   Exited (0) About an hour ago              blissful_
mccarthy
f1b755532336   debian:latest  "/bin/bash"              2 hours ago         Exited (255) 26 minutes ago               jolly_bas
si
8313afaf6374   debian:latest  "/bin/bash"              2 hours ago         Exited (0) 2 hours ago                    fervent_m
orse
d2d0e43b41eb   debian         "/bin/bash"              3 hours ago         Exited (255) 26 minutes ago               crazy_bel
l
0cfc6a98a68c   debian:latest  "/bin/bash"              4 hours ago         Exited (255) 26 minutes ago               jolly_edi
son
```

4） 在终端输入“**docker exec -it 75b6634e6227 /bin/bash**”进入容器。

```
pi@raspberrypi:~ $ docker exec -it 75b6634e6227 /bin/bash
root@raspberrypi:/#
```

5） 在终端输入“**useradd -m -s /bin/bash ubuntu**”，创建一个新用户。

```
root@raspberrypi:/# useradd -m -s /bin/bash ubuntu
```

6） 在终端输入“**passwd ubuntu**”，设置“**ubuntu**”的密码，输入密码按下之后，需要重新输入一遍密码。

```
root@50e095a38fff:/# passwd ubuntu
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

7） 在终端输入“**usermod -aG sudo ubuntu**”，将新用户 ubuntu 添加sudo，使其具有超级用户权限。

```
root@raspberrypi:/# usermod -aG sudo ubuntu
```

8） 在终端输入“**sudo apt-get update -y && sudo apt-get upgrade -y**”，更新可用软件包列表并升级系统上已安装的软件包。

```
root@raspberrypi:/# sudo apt-get update -y && sudo apt-get upgrade -y
Get:1 http://snapshots.ros.org/melodic/final/ubuntu bionic InRelease [13.8 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports bionic InRelease [242 kB]
Get:3 http://snapshots.ros.org/melodic/final/ubuntu bionic/main arm64 Packages [1017 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports bionic-updates InRelease [88.7 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports bionic-backports InRelease [83.3 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports bionic-security InRelease [88.7 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports bionic/universe arm64 Packages [11.0 MB]
Get:8 http://ports.ubuntu.com/ubuntu-ports bionic/main arm64 Packages [1285 kB]
Get:9 http://ports.ubuntu.com/ubuntu-ports bionic/restricted arm64 Packages [572 B]
```

9） 在终端输入“**sudo apt-get install vim -y**”，安装 Vim 文本编辑器。

```
root@raspberrypi:/# sudo apt-get install vim -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libgpm2 vim-common vim-runtime xxd
Suggested packages:
  gpm ctags vim-doc vim-scripts
The following NEW packages will be installed:
```

10）在终端输入“**sudo apt-get install ros-melodic-desktop-full -y**”，安装 ROS Melodic 版本的完整桌面环境，下载需要一些时间，请耐心等待一会。

```
root@raspberrypi:/# sudo apt-get install ros-melodic-desktop-full -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  autoconf automake autopoint autotools-dev avahi-daemon bind9-host blt bsdmainutils curl cython dbus d
  dh-autoreconf dh-strip-nondeterminism dmsetup file fltk1.3-doc fluid fonts-lato fonts-liberation font
```

1.2 测试ROS1环境

1） 输入“**docker exec -it -u ubuntu -w /home/ubuntu 75b6 /bin/bash**”指令，进入容器。（注意：**75b6**是装有ROS1环境的容器ID）

```
pi@raspberrypi:~ $ docker exec -it -u ubuntu -w /home/ubuntu 75b6 /bin/bash
```

2） 输入“**source /opt/ros/melodic/setup.bash**”指令，手动配置ROS1环境。

```
ubuntu@raspberrypi:~$ source /opt/ros/melodic/setup.bash
ubuntu@raspberrypi:~$
```

3） 每一次打开终端都执行2）步骤加载工作空间，可以在终端输入“echo “**source /opt/ros/melodic/setup.bash**” **>> ~/.bashrc**”指令，将该指令写入.bashrc文件中。

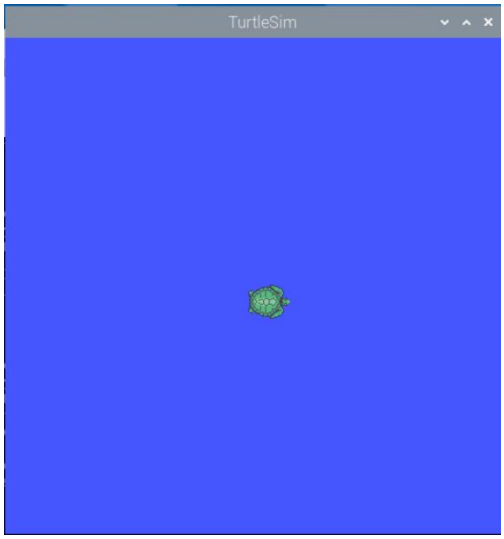
```
ubuntu@raspberrypi:~$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
ubuntu@raspberrypi:~$
```

4） 然后输入“**source ~/.bashrc**”指令，让**.bashrc**文件生效。这样就不需要每一次加载工作空间环境。

```
ubuntu@raspberrypi:~$ source ~/.bashrc
ubuntu@raspberrypi:~$
```

5） 输入“**rosrun turtlesim turtlesim_node**”，启动小海龟GUI界面，启动成功即说明ROS1安装成功。

```
ubuntu@raspberrypi:~$ rosrun turtlesim turtlesim_node
```



2.ROS2安装

2.1 这里以humble版本为例（需联网）

2） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker pull ros:humble**”，下载ROS2镜像，镜像下载需要一些时间，请耐心等待一会。

```
pi@raspberrypi:~ $ docker pull ros:humble
```

3） 镜像下载完成后，在终端输入“**docker run -it --network=host -d -v=/dev:/dev -v /tmp/.X11-unix:/tmp/.X11-unix --name humble -e DISPLAY=${DISPLAY} --restart=always ros:humble /bin/bash**”，运行容器并指定名称为 humble。

```
pi@raspberrypi:~ $ docker run -it --network=host -d -v=/dev:/dev -v /tmp/.X11-unix:/tmp/.X11-unix --name melodic -e DISPLAY=${
DISPLAY} --restart=always ros:melodic /bin/bash
75b6634e622762255058569dd769ed3925f77b9976539e85d1d70cd029929f9d
```

4） 在终端输入“**xhost +**”，开启 X Server 的访问控制。

```
pi@raspberrypi:~ $ xhost +
access control disabled, clients can connect from any host
```

5） 在终端输入“**docker ps -a**”查看新创建容器的ID。

```
pi@raspberrypi:~ $ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                      PORTS     NAMES
76a092503f2e   ros:humble     "/ros_entrypoint.sh …"   14 seconds ago   Up 13 seconds                         humble
75b6634e6227   ros:melodic    "/ros_entrypoint.sh …"   18 hours ago     Up 12 minutes                         melodic
a24dbc912220   debian         "bash"                   20 hours ago     Exited (0) 20 hours ago               blissful_mccar
hy
f1b755532336   debian:latest  "/bin/bash"              20 hours ago     Exited (255) 19 hours ago             jolly_bassi
8313afaf6374   debian:latest  "/bin/bash"              20 hours ago     Exited (0) 20 hours ago               fervent_morse
d2d0e43b41eb   debian         "/bin/bash"              22 hours ago     Exited (255) 19 hours ago             crazy_bell
0cfc6a98a68c   debian:latest  "/bin/bash"              22 hours ago     Exited (255) 19 hours ago             jolly_edison
```

6） 在终端输入“**docker exec -it 76a092503f2e /bin/bash**”进入容器。

```
pi@raspberrypi:~ $ docker exec -it 76a092503f2e /bin/bash
```

7） 在终端输入“**useradd -m -s /bin/bash ubuntu**”，创建一个新用户。

```
root@raspberrypi:/# useradd -m -s /bin/bash ubuntu
```

8） 在终端输入“**passwd ubuntu**”，设置“**ubuntu**”的密码，这里我们设置为“**ubuntu**”输入密码按下之后，需要重新输入一遍密码。

```
root@50e095a38fff:/# passwd ubuntu
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

9） 在终端输入“**usermod -aG sudo ubuntu**”，将新用户 ubuntu 添加sudo，使其具有超级用户权限。

```
root@raspberrypi:/# usermod -aG sudo ubuntu
```

10）在终端输入“**sudo apt-get update -y && sudo apt-get upgrade -y**”，更新可用软件包列表并升级系统上已安装的软件包。

```
root@raspberrypi:/# sudo apt-get update -y && sudo apt-get upgrade -y
Get:1 http://snapshots.ros.org/melodic/final/ubuntu bionic InRelease [13.8 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports bionic InRelease [242 kB]
Get:3 http://snapshots.ros.org/melodic/final/ubuntu bionic/main arm64 Packages [1017 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports bionic-updates InRelease [88.7 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports bionic-backports InRelease [83.3 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports bionic-security InRelease [88.7 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports bionic/universe arm64 Packages [11.0 MB]
Get:8 http://ports.ubuntu.com/ubuntu-ports bionic/main arm64 Packages [1285 kB]
Get:9 http://ports.ubuntu.com/ubuntu-ports bionic/restricted arm64 Packages [572 B]
```

在终端输入“**sudo apt-get install vim -y**”，安装 Vim 文本编辑器。

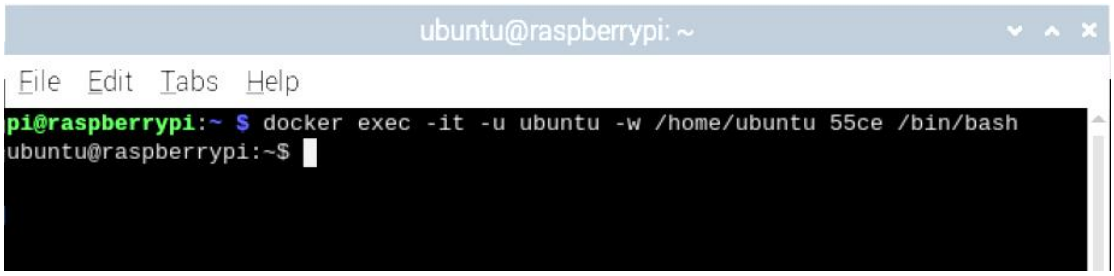
```
root@raspberrypi:/# sudo apt-get install vim -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libgpm2 vim-common vim-runtime xxd
Suggested packages:
  gpm ctags vim-doc vim-scripts
The following NEW packages will be installed:
```

11）在终端输入“**sudo apt-get install ros-humble-desktop-full -y**”，安装ROS Humble 版本的完整桌面环境。

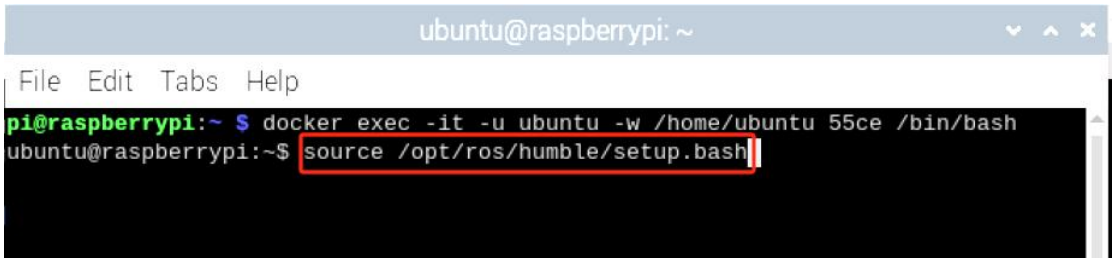
```
root@raspberrypi:/# sudo apt-get install ros-melodic-desktop-full -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  autoconf automake autopoint autotools-dev avahi-daemon bind9-host blt bsdmainutils curl cython dbus d
  dh-autoreconf dh-strip-nondeterminism dmsetup file fltk1.3-doc fluid fonts-lato fonts-liberation font
```

2.2 测试ROS2环境

1） 输入“**docker exec -it -u ubuntu -w /home/ubuntu 55ce /bin/bash**”指令，进入容器。（注意：55ce是装有ROS2环境的容器ID）



2） 输入“**source /opt/ros/humble/setup.bash**”指令，手动配置ROS2环境。



3） 每一次打开终端都执行2）步骤加载工作空间，可以在终端输入“**echo “source /opt/ros/humble/setup.bash” >> ~/.bashrc**”指令，将该指令写入.bashrc文件中。

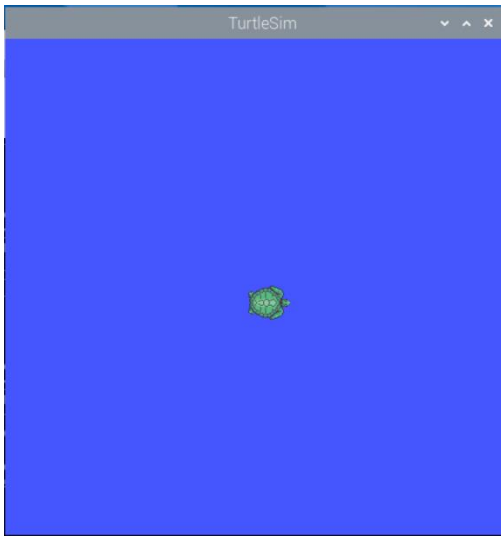
```
ubuntu@raspberrypi:~$ echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
```

4） 然后输入“**source ~/.bashrc**”指令，让**.bashrc**文件生效。这样就不需要每一次加载工作空间环境。

```
ubuntu@raspberrypi:~$ source ~/.bashrc
ubuntu@raspberrypi:~$
```

5） 输入“**ros2 run turtlesim turtlesim_node**”，启动小海龟GUI界面，启动成功即说明ROS2安装成功。

```
ubuntu@raspberrypi:~$ ros2 run turtlesim turtlesim_node
```



第 8 课 Docker仓库

docker仓库（repository）是集中存放镜像文件的场所。最大的公开仓库是docker hub(https://hub.docker.com/)，存放了数量庞大的镜像供用户下载。国内的公开仓库包括阿里云 、网易云等。

1．发布镜像到docker hub

① 地址：https://hub.docker.com/ ，先注册账号

② 保证账号可以正常登录

③ 使用 tag命令修改镜像名称

发布镜像到docker hub的规范是：docker push 注册用户名/镜像名

这里的注册用户名是:tao082，那就要先修改镜像名称

1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker images**”，查看主机上的镜像。

```
pi@raspberrypi:~ $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
debian        1.1       30555f5a4f74   28 minutes ago   139MB
test/debian   v1        7f96d767c7be   18 hours ago     139MB
mysql         latest    b068ea59c677   4 days ago       638MB
debian        latest    60e0ade36b1a   12 days ago      139MB
ros           melodic   3365d517511a   6 weeks ago      1.2GB
hello-world   latest    ee301c921b8a   8 months ago     9.14kB
```

2） 在终端输入“**docker tag 30555f5a4f74 tao082/debian:1.1**”，修改镜像名。

```
pi@raspberrypi:~ $ docker tag 30555f5a4f74 tao082/debian:1.1
```

3） 在终端输入“**docker images**”，可以看到镜像名称已经修改成功了。

```
pi@raspberrypi:~ $ docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
debian              1.1       30555f5a4f74   35 minutes ago   139MB
tao082/debian       1.1       30555f5a4f74   35 minutes ago   139MB
```

④ 登录docker hub发布镜像：

1） 在终端输入“**docker login -u tao082**”，按下回车，输入密码，进行登录。

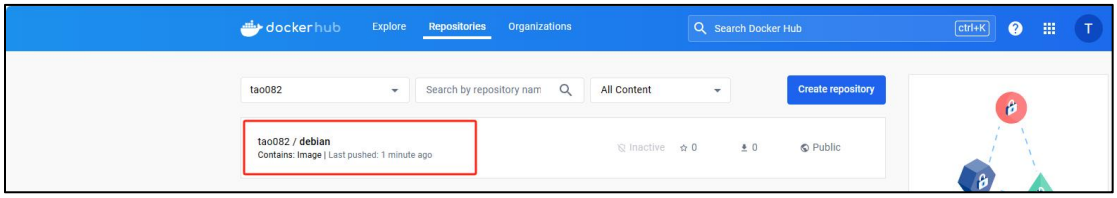
```
pi@raspberrypi:~ $ docker login -u tao082
Password:
WARNING! Your password will be stored unencrypted in /home/pi/.docker/config.jso
n.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

2） 在终端输入“docker push tao082/debian:1.1 ”，按下回车，发布镜像。

```
pi@raspberrypi:~ $ docker push tao082/debian:1.1
The push refers to repository [docker.io/tao082/debian]
72c13eb508ab: Layer already exists
688a80a87f8f: Layer already exists
39ce3f073d94: Layer already exists
bf7f4fd8fe90: Layer already exists
77ad615d04d4: Layer already exists
1.1: digest: sha256:509237074c04151d36b3e082a709edf9a57cf54924f24c4913b5ecc1a350
6271 size: 1358
```

⑤ 访问docker hub可查看到已经发布成功



2．发布镜像生成tar压缩包

以下方法都是在宿主机中操作，而非docker里面操作

方式一：docker export 和 docker import（从容器中导出压缩包）

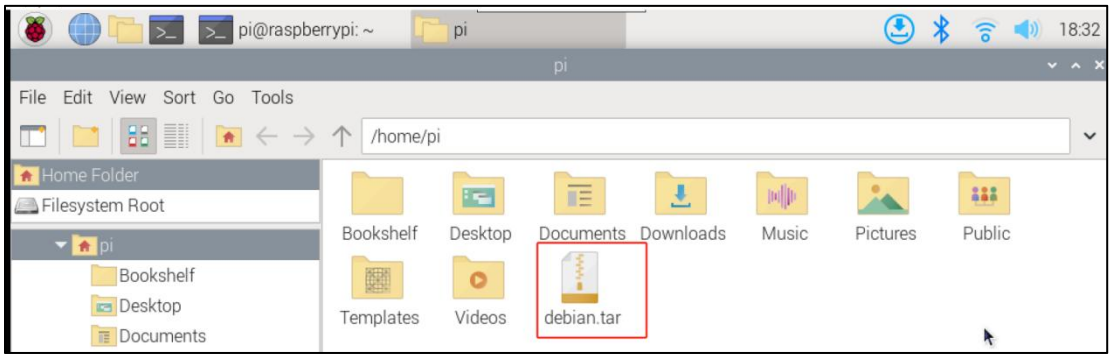
1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker ps**”，显示当前主机中的容器。

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
efe969b704a7   debian    "/bin/bash"   8 seconds ago    Up 5 seconds              fervent_williams
cd1df225f0bf   debian    "/bin/bash"   8 minutes ago    Up 7 minutes              nostalgic_black
```

2） 在终端输入“**docker export efe969b704a7 >debian.tar**”，将“**efe969b704a7**”容器导出为“**debian.tar**”。

```
pi@raspberrypi:~ $ docker export efe969b704a7 >debian.tar
```

可以在当前路径下看到导出的容器。



3） 在终端输入“**cat /home/pi/debian.tar | docker import - test/debian:v1**”，为“**/home/pi/debian.tar**”导入容器所在路径，“**test/debian**”为镜像名，“**v1**”为标签名。

```
pi@raspberrypi:~ $ cat /home/pi/debian.tar | docker import - test/debian:v1
sha256:7f96d767c7be0fc55a3ab368058590df2245b1bae477cdf1a5fc39f2d46a72b0
```

4） 在终端输入“**docker images**”，按下回车，查看主机中的容器信息。

```
pi@raspberrypi:~ $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
test/debian   v1        7f96d767c7be   5 minutes ago    139MB
debian        latest    60e0ade36b1a   11 days ago      139MB
hello-world   latest    ee301c921b8a   8 months ago     9.14kB
```

方式二：docker save 和 docker load（从镜像中导出压缩包）

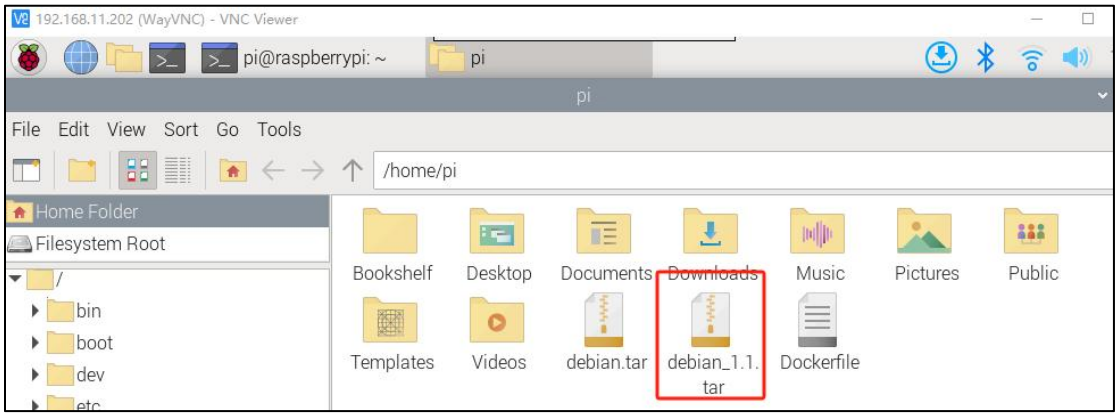
1） 按下“**Ctrl+Alt+T**”，打开命令行终端，在终端输入“**docker images**”，查看当前主机中的镜像。

```
pi@raspberrypi:~ $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
test/debian   v1        7f96d767c7be   5 minutes ago    139MB
debian        latest    60e0ade36b1a   11 days ago      139MB
hello-world   latest    ee301c921b8a   8 months ago     9.14kB
```

2） 在终端输入“**docker save -o debian_1.1.tar debian:latest**”，按下回车，将镜像生成压缩包。

```
pi@raspberrypi:~ $ docker save -o debian_1.1.tar debian:latest
```

可以在当前路径下看到导出的容器。



3） 在终端输入“**docker load -i debian_1.1.tar**”，**将**生成压缩包后使用docker load导入到镜像库

```
pi@raspberrypi:~ $ docker load -i debian_1.1.tar
Loaded image: debian:latest
```

两种方式的区别：

docker export 是从容器（container)中生成，docker save 是镜像（image）中生成。docker export比 docker save 保存的包要小，原因是 save 保存的是一整个分层的文件系统，export 导出的只是容器一层文件系统。docker import 和 docker load 导入都会生成镜像。docker import 可以对镜像指定新名称及版本号，docker load 无法对镜像重命名。