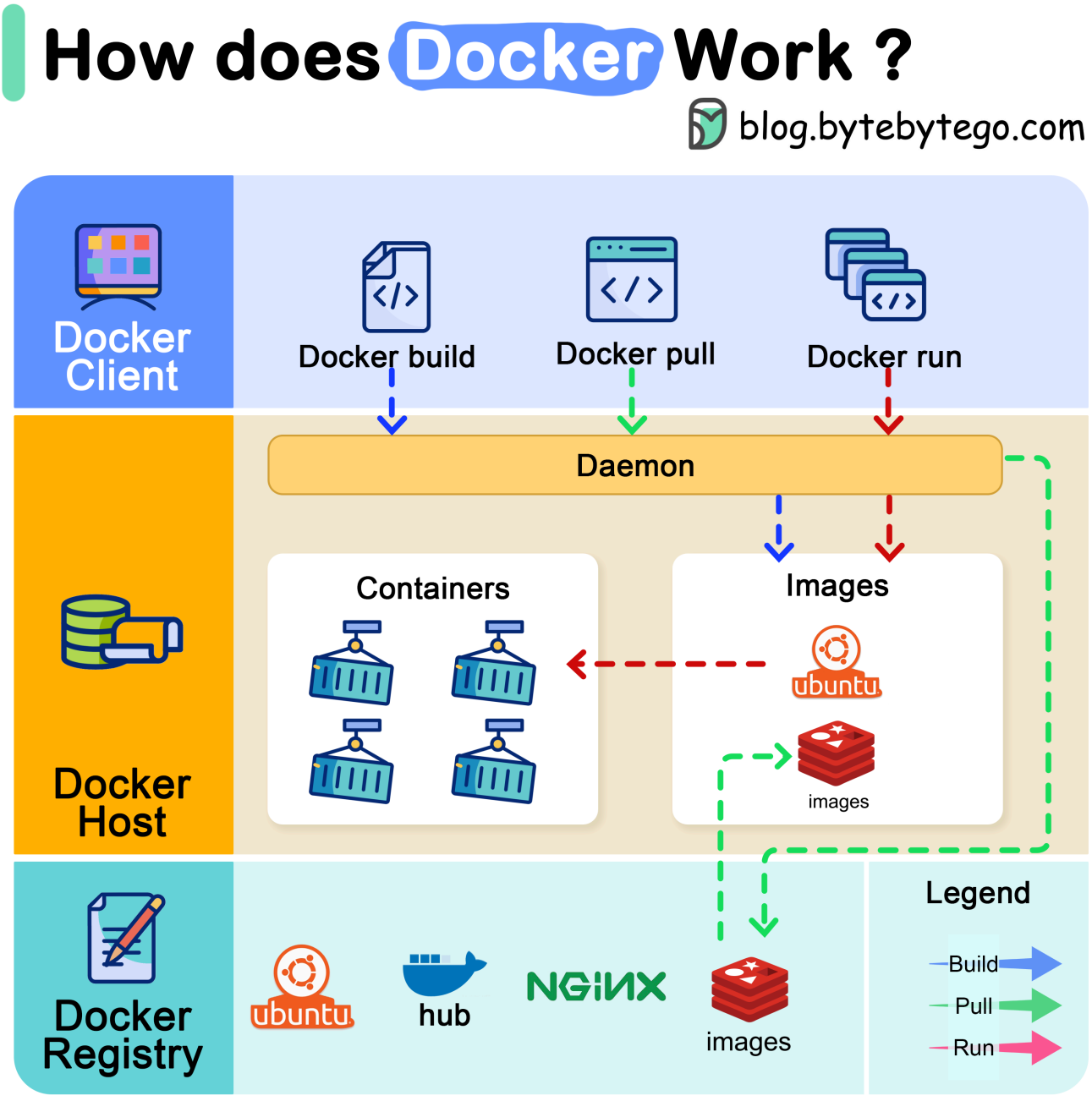
docker容器化技术的原理和实现

docker的起源与特点：

> docker首次发布于2013年3月13日，是一种操作系统层虚拟化，但是与虚拟机不同的是，docker 容器利用主机操作系统的内核，通过“隔离技术”为应用程序提供一个独立的运行环境。

> docker和虚拟机（比如virtua box,vmware workstation） 都可以虚拟运行操作系统，但是docker 容器不能像某些虚拟机那样直接运行在裸机硬件上，而必须依赖于一个宿主机操作系统。比如docker可以运行在MacOS、Windows、Linux等主流操作系统上。
>
> Docker 容器运行在宿主机操作系统之上，并共享其内核，而不是像虚拟机那样虚拟化整套硬件。



docker解决了什么问题：

一般来说，在每台电脑上的开发环境都是不同的，这就导致一个项目能运行在我的电脑，但是不能直接在你的电脑上运行。有两个常见的方法：

1. 第一个是修改自己的开发环境，但是版本依赖层层相扣，堪称依赖地狱，费时费力还可能摧毁现有的开发环境。
2. 第二个是使用虚拟机，但是使用虚拟机配置开发环境也需要大量时间，还十分消耗电脑性能，因为虚拟机的很多硬件开销是不必要的。

> 直到docker的出现，极大改善了这一现象，想象一个场景：
>
> 你发现了一个有趣但是古老的项目A，你想要继续开发维护它：
>
> 当你下载下来这个项目，发现它使用了`Python2.7.10`（我的电脑上`python`版本为`3.10.11`）

依赖使用了

- `numpy 1.11.0`（这是Python 进行科学计算的基础包）
- `pandas 0.18.1`（数据处理与分析工具）
- `scikit-learn 0.17.1`（机器学习库）

1. 由于你使用的是`python3`，但是代码是python2语法，无法运行，于是你选择安装`python 2.7.10`，还要为此管理多个版本的Python
2. 当你安装号`python 2.7.10`后，使用`pip install -r requirements.txt`

   报错发现`pandas 0.18.1`需要`numpy>1.21.0`，导致版本冲突

   `scikit-learn 0.17.1`无法使用新版C++编译器编译，安装失败

> requirements.txt是一个纯文本文件，用于列出python项目所依赖的所有第三方软件包及其**精确版本。**
>
> Pip 为 Python install package

- 或者软件包`A 1.0`依赖`lib-x 3.0`,而软件包`B 1.2`依赖`lib-x 4.0`，软件包`A` `B`无法在同一环境下共存。这是传递依赖冲突，不是直接的冲突，仍然会导致失败

我曾经就因为想要自行编译`ffmpeg`，因为依赖地狱最后放弃自编译。

那么docker是如何如何解决的：

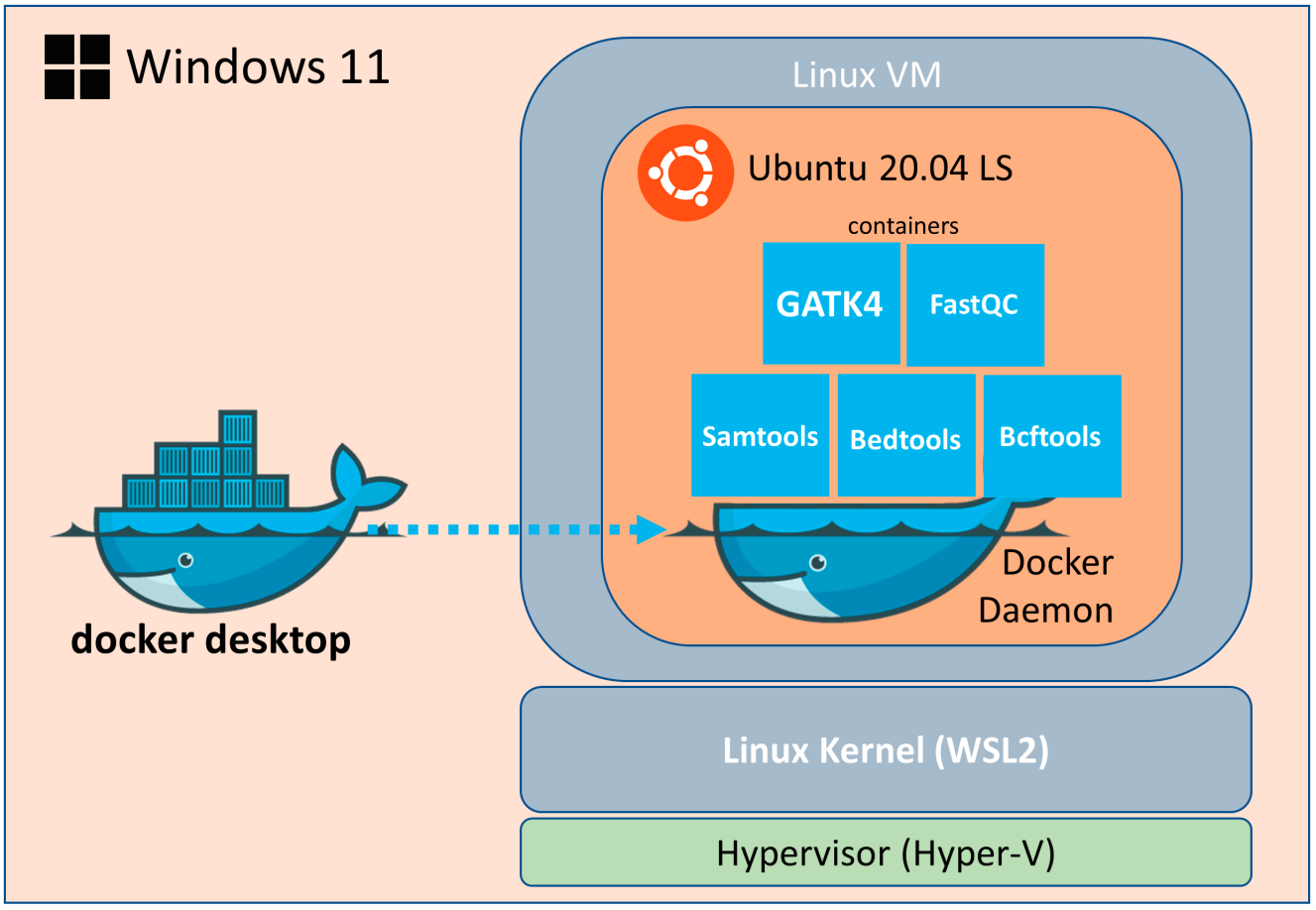
1. 使用一个安装了`python 2.7.10`的Linux镜像，这个镜像只包括最基础的系统和环境
2. 因为Python版本、编译器版本与这些古老库诞生的年代一致，所以安装成功率极高
3. 将整个系统保存下来，打包为镜像。

docker的安装

此处主要讲解Linux和Windows如何安装使用docker，两者大部分是互通的

windows的docker运行在WSL2（*Windows Subsystem for Linux 2*）上

```
PS C:\Users\jaych> wsl -v
WSL 版本： 2.5.10.0
内核版本： 6.6.87.2-1
WSLg 版本： 1.0.66
MSRDC 版本： 1.2.6074
Direct3D 版本： 1.611.1-81528511
DXCore 版本： 10.0.26100.1-240331-1435.ge-release
Windows: 10.0.26200.6899
PS C:\Users\jaych> wsl
wsl: 检测到 localhost 代理配置，但未镜像到 WSL。NAT 模式下的 WSL 不支持 localhost 代理。
jay@jay:/mnt/c/Users/jaych$ docker -v
Docker version 28.5.1, build e180ab8
```



此处主讲Linux。

docker、docker compose安装

> docker推荐从docker官方APT源下载，Debian官方的APT源内的docker一般较久，甚至还在使用`docker-compose`而非`docker compose`

以我最熟悉的Debian12为例

1. 卸载关于docker的所有过去的包，他们可能导致冲突

   `sudo apt remove $(dpkg --get-selections docker.io docker-compose docker-doc podman-docker containerd runc | cut -f1)`

   docker engine过去使用`containerd`和`runc`，现在已经捆绑为一个包`containerd.io`

   `podman`为下一代虚拟化技术

2. 设置docker官方源

```
# Add Docker's official GPG key:
sudo apt update
sudo apt install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
Types: deb
URIs: https://download.docker.com/linux/debian
Suites: $(. /etc/os-release && echo "$VERSION_CODENAME")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF

sudo apt update
```

由于Kail Linux为Debian12衍生而来，需要将`$(. /etc/os-release && echo "$VERSION_CODENAME")`替换为`bookworm`，这是debian12的代号。

3. 安装docker最新版

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
```

docker将在安装后自启动，通过`sudo systemctl status docker`验证docker是否正常启动

保险起见可以先运行`sudo systemctl start docker`启动docker，也可以验证docker是否安装

```
jay@s868537:~$ sudo systemctl status docker
[sudo] password for jay:
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-11-11 12:03:28 UTC; 22h ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 798 (dockerd)
      Tasks: 270
     Memory: 451.9M
        CPU: 39min 40.461s
     CGroup: /system.slice/docker.service
             ├─   798 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
             ├─  2874 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 10633 -container-ip 1>
             ├─  2901 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 10633 -container-ip 172.18>
             ├─  2999 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 3011 -container-ip 19>
             ├─  3007 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 3011 -container-ip 192.168>
             ├─  3134 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 5244 -container-ip 17>
             ├─  3154 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 5244 -container-ip 172.31.>
```

4. 将用户添加到docker组

切换到要添加到docker组的用户，比如jay

```
su jay
sudo usermod -aG docker $USER
exit # 推出当前用户，更新状态
su jay
id # ···994(docker)
```

5. 更新docker源

   docker需要更换国内镜像源才能正常使用，docker源主要有

   - **docker.io (docker inc)**
   - **ghcr.io (github)**
   - **quay.io (redhat)**
   - **gcr.io (k8s)**

   1. 找到一个可用的docker镜像源
      - https://docker.1panel.dev/
      - https://gallery.ecr.aws/
      - https://docker.aityp.com/s/docker.io
   2. 修改文件 /etc/docker/daemon.json

      ```
      sudo nano /etc/docker/daemon.json
      # 默认配置：
      {
          "runtimes": {
              "runsc": {
                  "path": "/usr/bin/runsc"
              }
          }
      }
      # 修改后配置：
      {
          "runtimes": {
              "runsc": {
                  "path": "/usr/bin/runsc"
              }
          },
          "registry-mirrors": ["https://docker.1panel.dev"]
      }
      ```

      重载systemd管理守护进程配置文件

      `sudo systemctl daemon-reload`

      重启docker

      `sudo systemctl restart docker`

```
jay@s868537:~$ sudo nano /etc/docker/daemon.json
jay@s868537:~$ sudo systemctl daemon-reload
jay@s868537:~$ sudo systemctl restart docker
jay@s868537:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Wed 2025-11-12 10:44:12 UTC; 16s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 715116 (dockerd)
      Tasks: 255
     Memory: 186.1M
        CPU: 4.796s
     CGroup: /system.slice/docker.service
```

5. 运行第一个docker容器

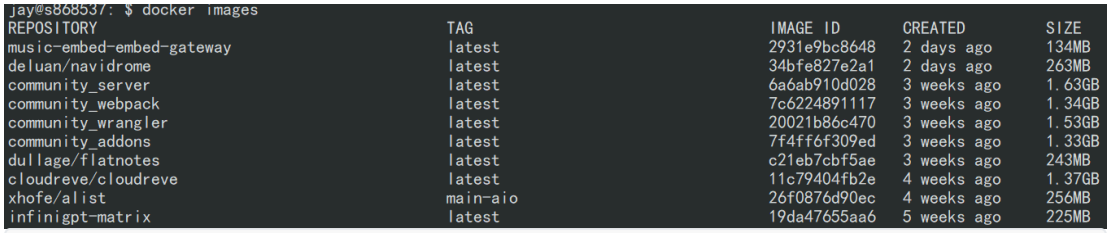
`sudo docker run hello-world`

分析输出：

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
Digest: sha256:f7931603f70e13dbd844253370742c4fc4202d290c80442b2e68706d8f33ce26
Status: Downloaded newer image for hello-world:latest

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.
    (amd64)
 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/
```

1. `Unable to find image 'hello-world:latest' locally`

   本地镜像检测，本地镜像没有查找到则需要远程拉取（docker images查找本地镜像）

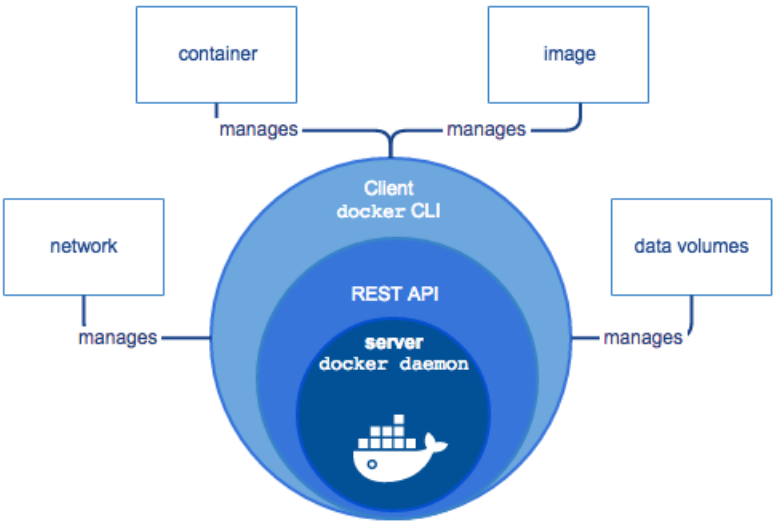


```
jay@s868537: $ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
music-embed-embed-gateway     latest    2931e9bc8648   2 days ago      134MB
deluan/navidrome              latest    34bfe827e2a1   2 days ago      263MB
community_server              latest    6a6ab910d028   3 weeks ago     1.63GB
community_webpack             latest    7c6224891117   3 weeks ago     1.34GB
community_wrangler            latest    20021b86c470   3 weeks ago     1.53GB
community_addons              latest    7f4ff6f309ed   3 weeks ago     1.33GB
dullage/flatnotes             latest    c21eb7cbf5ae   3 weeks ago     243MB
cloudreve/cloudreve           latest    11c79404fb2e   4 weeks ago     1.37GB
xhofe/alist                   main-aio  26f0876d90ec   4 weeks ago     256MB
infinigpt-matrix              latest    19da47655aa6   5 weeks ago     225MB
```

2.
```
latest: Pulling from library/hello-world
# 查找到镜像，tag为latest
17eec7bbc9d7: Pull complete
# 分层下载，此处只有一层
Digest:
sha256:f7931603f70e13dbd844253370742c4fc4202d290c80442b2e68706d8f33ce26
# SHA256校验保证镜像完整
Status: Downloaded newer image for hello-world:latest
# 镜像成功下载到本地
```

3.
```
Hello from Docker!
This message shows that your installation appears to be working correctly.
# docker安装运行正常

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 #docker-cli和docker守护进程建立了连接
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
    # docker守护进程从Docker Hub拉取了“hello-world”镜像，镜像架构为AMD64
 3. The Docker daemon created a new container from that image which runs
the
    executable that produces the output you are currently reading.
    # docker守护进程从镜像创建了容器，执行了现在看到的命令
 4. The Docker daemon streamed that output to the Docker client, which sent
it
    to your terminal.
    # docker守护进程将输出通过流式传输到docker客户端，出现在终端上
```

Docker Engine



Storage

1. Volume特点 创建 删除

   docker使用映射和Volume管理数据存储

   Volume也称卷，是docker用于容器container的持久数据存储

   由于映射绑定是由宿主机掌握的（/home/jay/Music:/music），如果/home/jay/Music路径改变了，容器就无法读写/home/jay/Music下的数据了，而Volume则由docker管理，这使

   1. 卷比绑定更容易迁移备份
   2. 新卷的内容可以通过容器或构建预先填充
   3. 卷可以在多个容器之间更安全地共享
   4. 可以使用 Docker CLI 命令或 Docker API 管理卷
   5. 有更高的I/O性能

   当然，如果你需要从主机访问添加文件，还是映射绑定更好，举例部署自托管服务Navidrome

   - 使用映射绑定的yaml文件：

```yaml
services:
  navidrome:
    image: deluan/navidrome:latest
    user: 1001:1001 # should be owner of volumes
    ports:
      - "4533:4533"
    restart: unless-stopped
    environment:
      # Optional: put your config options customization here. Examples:
      # ND_LOGLEVEL: debug
      - ND_DEFAULTSHAREEXPIRATION=8760h
      - ND_ENABLESHARING=true
      - ND_LASTFM_LANGUAGE=zh
```

```
      - ND_LASTFM_APIKEY=XXXXXXXXXXXXX
      - ND_LASTFM_SECRET=XXXXXXXXXXXXX
      - ND_SPOTIFY_ID=XXXXXXXXXXXXX
      - ND_SPOTIFY_SECRET=XXXXXXXXXXXXX
    volumes:
      - "/home/jay/navidrome/data:/data"
      - "/home/jay/Music:/music:ro"
```

- 混合映射绑定&卷的yaml文件：

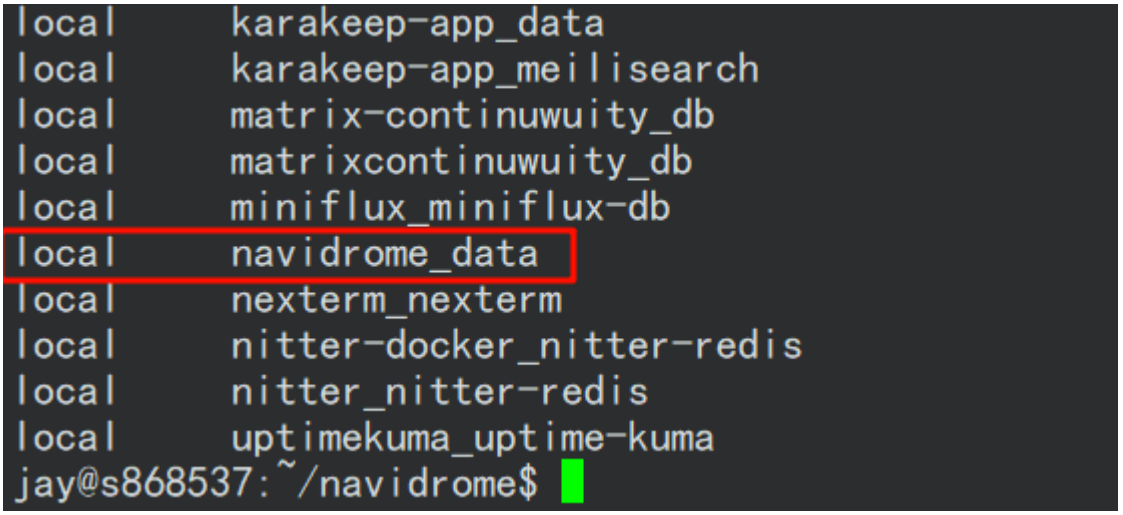
```yaml
services:
  navidrome:
    image: deluan/navidrome:latest
    user: 1001:1001
    ports:
      - "4533:4533"
    restart: unless-stopped
    environment:
      - ND_DEFAULTSHAREEXPIRATION=8760h
      - ND_ENABLESHARING=true
      - ND_LASTFM_LANGUAGE=zh
      - ND_LASTFM_APIKEY=XXXXXXXXXXXXX
      - ND_LASTFM_SECRET=XXXXXXXXXXXXX
      - ND_SPOTIFY_ID=XXXXXXXXXXXXX
      - ND_SPOTIFY_SECRET=XXXXXXXXXXXXX
    volumes:
      - "navidrome_data:/data"
      - "/home/jay/Music:/music:ro"

volumes:
  navidrome_data:
```

```
# 手动创建卷
docker volume create navidrome_data

# 查找卷再主机上的路径
docker volume inspect navidrome_data
# [
#     {
#         "CreatedAt": "2025-11-12T11:28:06Z",
#         "Driver": "local",
#         "Labels": null,
#         "Mountpoint": "/var/lib/docker/volumes/navidrome_data/_data",
#         "Name": "navidrome_data",
#         "Options": null,
#         "Scope": "local"
#     }
# ]

# 查看具体权限，默认root权限
sudo ls -la /var/lib/docker/volumes/navidrome_data
```

```
# total 12
# drwx-----x  3 root root 4096 Nov 12 11:28 .
# drwx-----x 46 root root 4096 Nov 12 11:28 ..
# drwxr-xr-x  2 root root 4096 Nov 12 11:28 _data

# 通过创建alpine虚拟机来设置数据卷权限，满足yaml文件的1001:1001
docker run --rm -it -v navidrome_data:/data alpine chown 1001:1001 /data
# Unable to find image 'alpine:latest' locally
# latest: Pulling from library/alpine
# 2d35ebdb57d9: Already exists
# Digest: sha256:4b7ce07002c69e8f3d704a9c5d6fd3053be500b7f1c69fc0d80990c2ad8dd412
# Status: Downloaded newer image for alpine:latest

# 查看具体权限,权限改动完成
sudo ls -la /var/lib/docker/volumes/navidrome_data
# total 12
# drwx-----x  3 root root 4096 Nov 12 11:28 .
# drwx-----x 46 root root 4096 Nov 12 11:28 ..
# drwxr-xr-x  2 jay  jay  4096 Nov 12 11:28 _data
```

删除Volume

```
docker volume prune # 删除未使用的卷（谨慎）
docker volume rm navidrome_data # 删除特定的卷，可以通过docker volume list查找volume
```



Networking

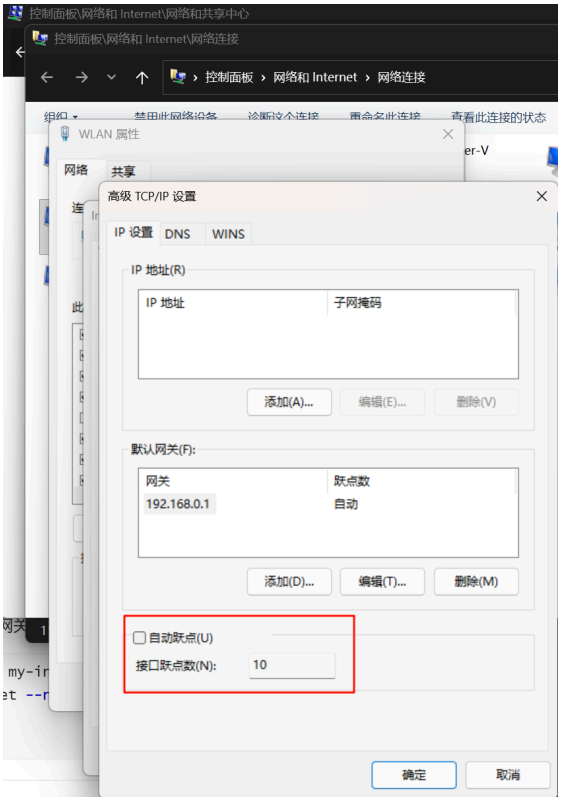
1. docker容器默认启用网络功能，为桥接网络（bridge）
2. 通过 --network 可以连接到用户自定义网络

```
# 创建自定义桥接网络
docker network create -d bridge my-net

# 在自定义网络中运行容器
docker run --network=my-net -it busybox
# -it busybox启用终端输出
```

`-d bridge`,以及`-d host`最为常用

驱动程序	描述	使用场景
`bridge`	默认网络驱动程序	单机环境容器通信
`host`	移除容器与主机的网络隔离	高性能应用
`none`	完全隔离容器网络	安全敏感应用
`overlay`	连接多个 Docker 守护进程	Swarm 集群
`ipvlan`	连接到外部 VLAN	企业网络集成
`macvlan`	容器作为主机网络设备	需要真实 MAC 地址的应用

创建内部网络，容器连接多个网络，设立网关优先级（就像Windows上的接口跃点数）




```
docker network create --internal my-internal-net
docker run --network=my-bridge-net --network=my-internal-net my-app # 两个网络，my-app为容器
docker run --network name=gwnet,gw-priority=1 --network anet1 --name myctr myimage # gw-priority=0,优先级最高的网络网关成为默认网关
```

3. 端口映射

```
    ......
    ports:
      - "4533:4533"
    ......
```

第一个4533：主机上的端口，可以，推荐，最好更改

第二个4533：容器的端口，不推荐更改

端口绑定的访问范围（外网访问，内网访问）

- 外网可访问（外网可访问，内网可访问，本机可访问，**容器间可访问**）

```
ports:
  - "4533:4533"

ports:
  - "0.0.0.0:4533:4533"
```

- 仅本机访问（**外网不可访问**，除非使用反向代理，**内网不可访问**，容器间可访问）

```
ports:
  - "127.0.0.1:4533:4533"
```

- 内网可访问（**外网不可访问**，除非使用反向代理，**本机可能能访问**，容器间可访问）

```
ports:
  - "192.168.1.100:4533:4533" # 本机ip为192.168.1.100就能访问
```

- 端口动态绑定（如果要使用NGINX等web服务器实现外网访问，不推荐此做法，除非使用docker网络，nginx处proxy_pass http://navidrome:4533;）适用于开发环境，安全考虑，通过docker的网络反向代理

```
ports:
  - "4533" # 此为容器端口
```

4. 网络管理：

```
docker network ls
docker network rm XXX-net
docker network inspect XXX-net # 显示网络详细情况
docker network connect XXX-net XXX-container # 连接运行中的容器到网络
docker network disconnect XXX-net XXX-container # 从网络断开连接
```

```
jay@s868537: /alist$ docker network ls
NETWORK ID     NAME                          DRIVER    SCOPE
a51fe855caa2   alist_default                 bridge    local
9ad98f02715d   blinko_blinko-network         bridge    local
59d4d039f390   bridge                        bridge    local
35b539e6196b   cadvisor_default              bridge    local
2de0d9487681   cloudreve_default             bridge    local
cf181fb587aa   flatnotes_default             bridge    local
709837fe4ee3   glances_default               bridge    local
c71f9474c6e4   host                          host      local
bebd99c401df   lemmy_default                 bridge    local
42438d68341c   linkwarden_default            bridge    local
bc5fce6b94f0   matrixcontinuwuity_default    bridge    local
bea8e8cbe84a   matterbridge_default          bridge    local
a96682b45e1f   miniflux_default              bridge    local
75ffc9c178b1   music-embed_default           bridge    local
b9c34f0b4c6e   navidrome_default             bridge    local
d4cad07c509c   none                          null      local
96ab57a8c31e   portracker_default            bridge    local
6eb18784d38e   semaphore_default             bridge    local
81840d5d389d   uptimekuma_default            bridge    local
```

Containers

1. 容器重启策略

策略	命令示例	描述	适用场景
`no`	`--restart no`	**不自动重启**	临时测试
`on-failure`	`--restart on-failure`	**失败时重启**	错误恢复
`on-failure:5`	`--restart on-failure:5`	**限制重启次数**	避免无限重启
`always`	`--restart always`	**总是重启**	关键服务
`unless-stopped`	`--restart unless-stopped`	**除非手动停止**	生产环境

docker-compose.yaml内配置方法（与image、container_name、volumes、ports、environment等同级）

```yaml
services:
  alist:
    image: 'xhofe/alist:main-aio'
    container_name: alist
    volumes:
      - '/etc/alist:/opt/alist/data'
      - '/home/jay/alist/storage:/opt/alist/storage'
    ports:
      - '5244:5244'
    environment:
      - PUID=1001
      - PGID=1001
      - UMASK=022
    restart: unless-stopped
```

2. 部分命令

```
docker run -d --restart unless-stopped XXX-container
docker run -d --restart on-failure:3 XXX-container
docker stop XXX-container
docker start XXX-container
systemctl restart docker
```

3. 进入容器：docker exec

   案例：

```
6bfa9714d798   deluan/navidrome:latest                "/app/navidrome"         6
hours ago   Up 2 hours               0.0.0.0:4533->4533/tcp, [::]:4533->4533/tcp
```

```
# 启动bash shell
docker exec -it <容器名或容器ID> /bin/bash
# 使用 sh
docker exec -it <容器名或容器ID> /bin/sh
# 查找（所有:包括停止的）容器
docker ps (-a)
# 退出容器
exit
# 启动容器时进入容器
docker run -it <容器名或容器ID> /bin/bash(sh)
```

```
jay@s868537:~$ docker exec -it 6bfa9714d798 /bin/bash
OCI runtime exec failed: exec failed: unable to start container process: exec: "/bin/bash": stat /bin/bas
h: no such file or directory: unknown
jay@s868537:~$ docker exec -it 6bfa9714d798 /bin/sh
/app $ cat /etc/os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.19.9
PRETTY_NAME="Alpine Linux v3.19"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://gitlab.alpinelinux.org/alpine/aports/-/issues"
/app $ ps aux
PID   USER     TIME  COMMAND
    1 1001      0:01 /app/navidrome
   37 1001      0:00 /bin/sh
   44 1001      0:00 ps aux
/app $ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if92: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 2e:9f:c0:91:f2:66 brd ff:ff:ff:ff:ff:ff
    inet 172.26.0.2/16 brd 172.26.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/app $
```

4. 附加到容器（查看输出）

```
jay@s868537:~/navidrome$ docker attach 6bfa9714d798
time="2025-11-12T12:32:50Z" level=info msg="Now Playing" artist="Marshmello/Jonas Brothers" player="Arpeg
gi [Arpeggi]" position=0 requestId=6bfa9714d798/HEQFbkcP20-000006 title="Leave Before You Love Me (with J
onas Brothers)" user=jay
time="2025-11-12T12:32:51Z" level=info msg="Streaming file" artist=Eminem bitRate=0 cached=false format=r
aw originalBitRate=320 originalFormat=mp3 requestId=6bfa9714d798/HEQFbkcP20-000007 title="Never Love Agai
n" transcoding=false user=jay
time="2025-11-12T12:33:05Z" level=info msg="Now Playing" artist=Marshmello player="Arpeggi [Arpeggi]" pos
ition=0 requestId=6bfa9714d798/HEQFbkcP20-000009 title=Shockwave user=jay
```

CLI

1. Docker过滤命令 `filter`

```
docker COMMAND --filter "KEY=VALUE"
docker COMMAND --filter "KEY!=VALUE"
```

查找状态为退出的容器 `docker ps --filter "status=exited"`

以及 `created, running, paused, restarting, removing, dead`

```
jay@s868537:~/navidrome$ docker ps --filter "status=exited"
CONTAINER ID   IMAGE                              COMMAND                  CREATED       STATUS
          PORTS     NAMES
100e5ad7d814   hello-world                        "/hello"                 2 hours ago   Exited (0) 2 hou
rs ago              nostalgic_wiles
6bfa9714d798   deluan/navidrome:latest            "/app/navidrome"         7 hours ago   Exited (0) 2 min
utes ago            navidrome-navidrome-1
9afab8e7ef34   gcr.io/cadvisor/cadvisor:v0.52.1   "/usr/bin/cadvisor -…"   5 weeks ago   Exited (0) 4 wee
ks ago              cadvisor
4e1e7c7af646   miniflux/miniflux:latest           "/usr/bin/miniflux"      5 weeks ago   Exited (1) 4 wee
ks ago              miniflux-miniflux-1
588ca602b531   postgres:17-alpine                 "docker-entrypoint.s…"   5 weeks ago   Exited (0) 4 wee
ks ago              miniflux-db-1
jay@s868537:~/navidrome$
```

按名称 `"name=XXX(navidrome_container)"`

按镜像 `"ancestor=XXX(nginx)"`

按退出码 `"exited=0(成功)/1(失败)"`

```
jay@s868537:~/navidrome$ docker ps -a --filter "exited=0"
CONTAINER ID   IMAGE                              COMMAND                  CREATED       STATUS
          PORTS     NAMES
100e5ad7d814   hello-world                        "/hello"                 2 hours ago   Exited (0) 2 hou
rs ago              nostalgic_wiles
6bfa9714d798   deluan/navidrome:latest            "/app/navidrome"         7 hours ago   Exited (0) 6 min
utes ago            navidrome-navidrome-1
9afab8e7ef34   gcr.io/cadvisor/cadvisor:v0.52.1   "/usr/bin/cadvisor -…"   5 weeks ago   Exited (0) 4 wee
ks ago              cadvisor
588ca602b531   postgres:17-alpine                 "docker-entrypoint.s…"   5 weeks ago   Exited (0) 4 wee
ks ago              miniflux-db-1
```

按标签 `"label=..."`

查找标签：`docker inspect <容器名或ID> | grep -A 10 "Labels"`

```
jay@s868537:~/navidrome$ docker inspect navidrome-navidrome-1 | grep -A 10 "Labels"
            "Labels": {
                "com.docker.compose.config-hash": "a4281a7b6f7fca4260da5f51e699cbdad7f21659196a4e8e3a110d
69a916f8df",
                "com.docker.compose.container-number": "1",
                "com.docker.compose.depends_on": "",
                "com.docker.compose.image": "sha256:34bfe827e2a1778a5a6f1bf7ac4d8d13ddba9e8c038be6c8d214a
40b383bffa4",
                "com.docker.compose.oneoff": "False",
                "com.docker.compose.project": "navidrome",
                "com.docker.compose.project.config_files": "/home/jay/navidrome/docker-compose.yml",
                "com.docker.compose.project.working_dir": "/home/jay/navidrome",
                "com.docker.compose.service": "navidrome",
                "com.docker.compose.version": "2.39.4",
jay@s868537:~/navidrome$
```

等等

2. 格式化输出（扩展）利于查看docker输出

```
docker COMMAND --format '{{模板表达式}}'
```

常用的格式化函数：

`docker inspect <容器/镜像>` - 查看详细信息

```
# 表格输出

# 自定义镜像列表列
docker image ls --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}\t{{.Size}}"
# 自定义容器列表列
docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Status}}\t{{.Ports}}"
# 网络列表格式化
docker network ls --format "table {{.ID}}\t{{.Name}}\t{{.Driver}}\t{{.Scope}}"

# JSON格式输出

# 以 JSON 格式输出挂载信息
docker inspect --format '{{json .Mounts}}' container_name
# 输出完整容器信息为 JSON
docker inspect --format '{{json .}}' container_name
# 输出环境变量为 JSON
docker inspect --format '{{json .Config.Env}}' container_name

# join - 连接字符串列表

# 连接容器的启动参数
docker inspect --format '{{join .Args " , "}}' container_name
# 连接环境变量（用换行符）
docker inspect --format '{{join .Config.Env "\n"}}' container_name
# 连接端口绑定信息
docker inspect --format '{{range $p, $conf := .NetworkSettings.Ports}}{{$p}} {{join
$conf ","}}{{end}}' container_name
```

Deamon

1. Docker 守护进程配置

   通过修改JSON配置文件

系统配置	配置文件路径
Linux 常规安装	`/etc/docker/daemon.json`
Linux Rootless 模式	`~/.config/docker/daemon.json`
Windows	`C:\ProgramData\docker\config\daemon.json`

```
sudo nano /etc/docker/daemon.json
sudo docker info | grep -A 10 "Docker Root Dir" # 输出检测配置语法
sudo systemctl restart docker
```

```
jay@s868537:~/navidrome$ sudo docker info | grep -A 10 "Docker Root Dir"
[sudo] password for jay:
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Experimental: false
 Insecure Registries:
  ::1/128
  127.0.0.0/8
 Registry Mirrors:
  https://docker.1panel.dev/
 Live Restore Enabled: false
```

检测特定配置项

```
docker info | grep -i "registry mirror"
docker info | grep -i "storage driver"
```

验证JSON语法无误

```
sudo python3 -m json.tool /etc/docker/daemon.json
```

```
jay@s868537:~/navidrome$ sudo python3 -m json.tool /etc/docker/daemon.json
{
    "runtimes": {
        "runsc": {
            "path": "/usr/bin/runsc"
        }
    },
    "registry-mirrors": [
        "https://docker.1panel.dev"
    ]
}
jay@s868537:~/navidrome$
```

查看docker详细日志

```
sudo journalctl -u docker.service -f
```

查看docker数据存放目录

```
docker info | grep "Docker Root Dir"
```

Docker Build

以部署misskey为例

与docker有关的文件

```
Dockerfile
.dockerignore
compose.local-db.yml
compose_example.yml
```

1. Dockerfile:

```
ARG NODE_VERSION=24.10.0-bookworm
FROM --platform=$BUILDPLATFORM node:${NODE_VERSION} AS native-builder
RUN --mount=type=cache,target=/var/cache/apt,sharing=locked \
    --mount=type=cache,target=/var/lib/apt,sharing=locked \
    rm -f /etc/apt/apt.conf.d/docker-clean \
    ; echo 'Binary::apt::APT::Keep-Downloaded-Packages "true";' >
/etc/apt/apt.conf.d/keep-cache \
    && apt-get update \
    && apt-get install -yqq --no-install-recommends \
    build-essential

......

ENV LD_PRELOAD=/usr/local/lib/libjemalloc.so
ENV NODE_ENV=production
HEALTHCHECK --interval=5s --retries=20 CMD ["/bin/bash", "/misskey/healthcheck.sh"]
ENTRYPOINT ["/usr/bin/tini", "--"]
CMD ["pnpm", "run", "migrateandstart"]
```

- ARG 构建时参数
- FROM 选择一个基础镜像作为起点
- WORKDIR 在容器内创建并进入工作目录
- COPY 从主机复制文件到镜像内
- RUN 执行命令，对镜像进行修改
- ENV 设置环境变量
- USER 切换运行用户
- CMD 指定容器启动时运行的默认命令
- ENTRYPOINT 设置入口点

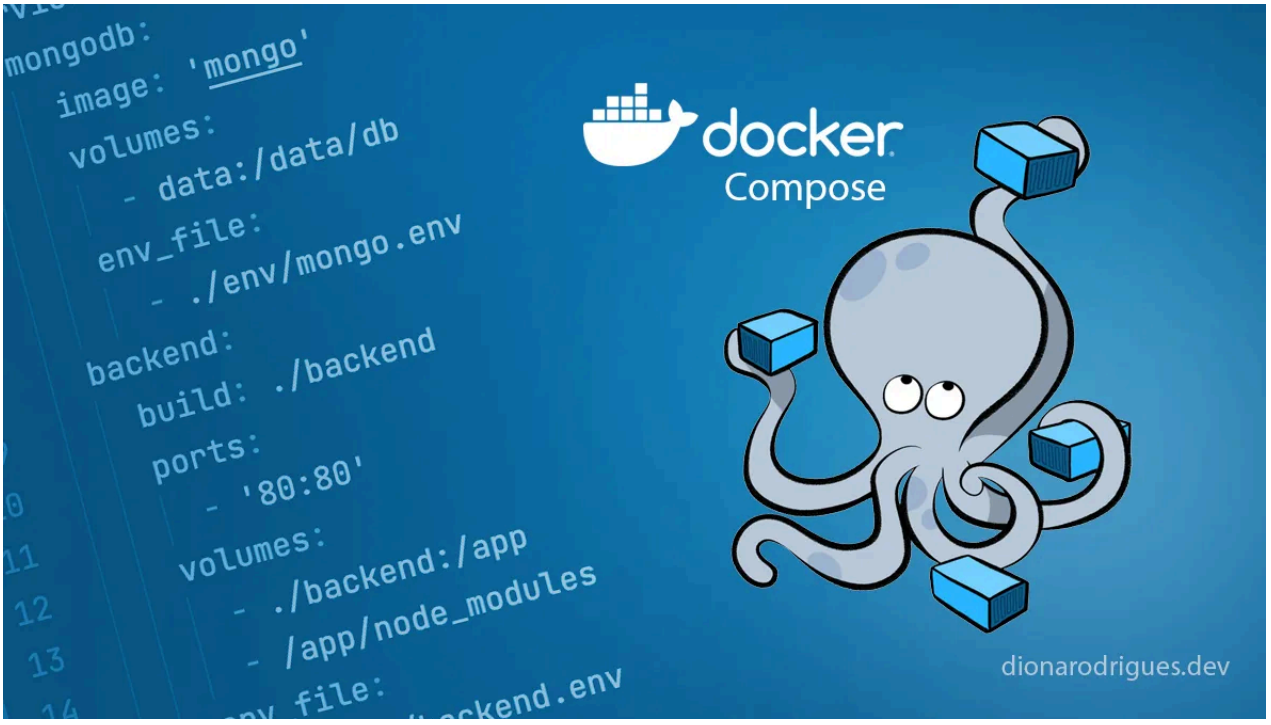
- **每个指令创建一个新的镜像层**

2. `.dockerignore`

   当运行`docker build .`时，会把整个目录发送给Docker守护进程，包括一些没用的文件.git 日志文件...

   `.dockerignore`告诉Docker构建时候忽略哪些文件，让`docker build .`构建快镜像小

```
.autogen
.github
.travis
.vscode
.config
Dockerfile

......

!.yarn/releases
!.yarn/sdks
!.yarn/versions

.pnpm-store

.idea/
packages/*/.vscode/
packages/backend/test/compose.yml
```



3. `compose.local-db.yml`

   仅数据库服务

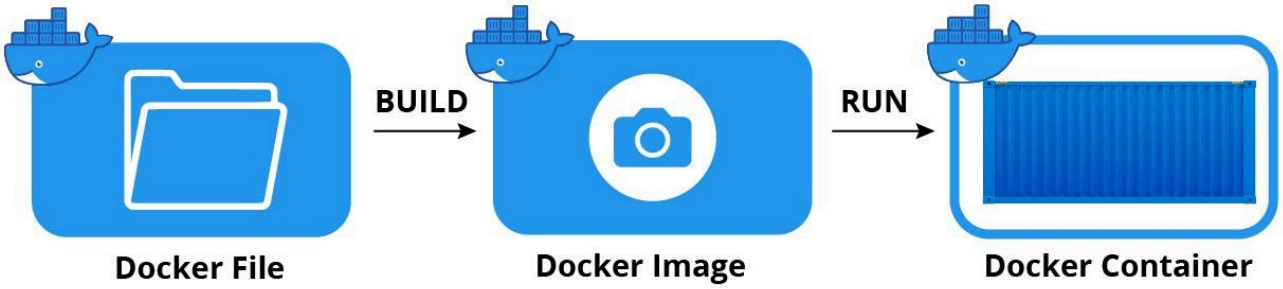
4. `compose_example.yml`

   完整应用栈，修改完后需要改为`compose.yml`再运行

```
services:
```

```yaml
  web:
    build: .
    restart: always
    links:
      - db
      - redis
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_healthy
    ports:
      - "3000:3000"
    networks:
      - internal_network
      - external_network
    env_file:
      - .config/docker.env
    volumes:
      - ./files:/misskey/files
      - ./.config:/misskey/.config:ro
  redis:
    restart: always
    image: redis:7-alpine
    networks:
      - internal_network
    volumes:
      - ./redis:/data
    healthcheck:
      test: "redis-cli ping"
      interval: 5s
      retries: 20
  db:
    restart: always
    image: postgres:15-alpine
    networks:
      - internal_network
    env_file:
      - .config/docker.env
    volumes:
      - ./db:/var/lib/postgresql/data
    healthcheck:
      test: "pg_isready -U $$POSTGRES_USER -d $$POSTGRES_DB"
      interval: 5s
      retries: 20
networks:
  internal_network:
    internal: true
  external_network:
```

开始构建misskey镜像：



1. git到本地，切换到了 master 分支

```
git clone -b master https://github.com/misskey-dev/misskey.git
cd misskey
git checkout master
```

2. 拉取配置文件的示例文件

```
cp .config/docker_example.yml .config/default.yml
cp .config/docker_example.env .config/docker.env
cp ./compose_example.yml ./compose.yml
```

修改三个文件

3. 构建镜像，数据库设置初始化

```
sudo docker compose build
sudo docker compose run --rm web pnpm run init
```

部分docker compose build输出

```
 => [runner  3/17] COPY ./package.json ./package.json
              0.1s
 => [runner  4/17] RUN node -e
"console.log(JSON.parse(require('node:fs').readFileSync('./package.  4.1s
 => [runner  5/17] WORKDIR /misskey
              0.0s
 => [native-builder 19/22] COPY --link . ./
              1.5s
 => [native-builder 20/22] RUN git submodule update --init
             2.3s
 => [native-builder 21/22] RUN pnpm build
             60.0s
 => [runner  6/17] COPY --chown=misskey:misskey --from=target-builder
/misskey/node_modules ./nod  20.4s
 => [runner  7/17] COPY --chown=misskey:misskey --from=target-builder
/misskey/packages/backend/no  0.1s
 => [runner  8/17] COPY --chown=misskey:misskey --from=target-builder
/misskey/packages/misskey-js  0.0s
```

```
 => [runner  9/17] COPY --chown=misskey:misskey --from=target-builder
/misskey/packages/misskey-re  0.0s
 => [runner 10/17] COPY --chown=misskey:misskey --from=target-builder
/misskey/packages/misskey-bu  0.0s
 => [native-builder 22/22] RUN rm -rf .git/
              0.3s
 => [runner 11/17] COPY --chown=misskey:misskey --from=native-builder /misskey/built
./built        1.9s
 => [runner 12/17] COPY --chown=misskey:misskey --from=native-builder
/misskey/packages/misskey-js  0.0s
 => [runner 13/17] COPY --chown=misskey:misskey --from=native-builder
/misskey/packages/misskey-re  0.0s
 => [runner 14/17] COPY --chown=misskey:misskey --from=native-builder
/misskey/packages/misskey-bu  0.0s
 => [runner 15/17] COPY --chown=misskey:misskey --from=native-builder
/misskey/packages/backend/bu  0.2s
 => [runner 16/17] COPY --chown=misskey:misskey --from=native-builder /misskey/fluent-
emojis /miss  0.3s
 => [runner 17/17] COPY --chown=misskey:misskey . ./
              1.1s
 => exporting to image
             18.4s
 => => exporting layers
              18.4s
 => => writing image
sha256:77795cb799f1683ac78cb7132c9b97a7103ee7665617789ea10bbbac24a2beb7        0.0s
 => => naming to docker.io/library/misskey-web
              0.0s
 => resolving provenance for metadata file
              0.0s
[+] Building 1/1
 ✓ misskey-web  Built
```

构建成功，输出`misskey-web`

使用`docker images | grep misskey`快速查找镜像

```
docker images | grep misskey
misskey-web   latest  77795cb799f1   6 minutes ago   2.65GB
```

构建镜像会消耗大量内存与CPU性能

`sudo docker compose run --rm web pnpm run init` **初始化 Misskey 数据库和设置**

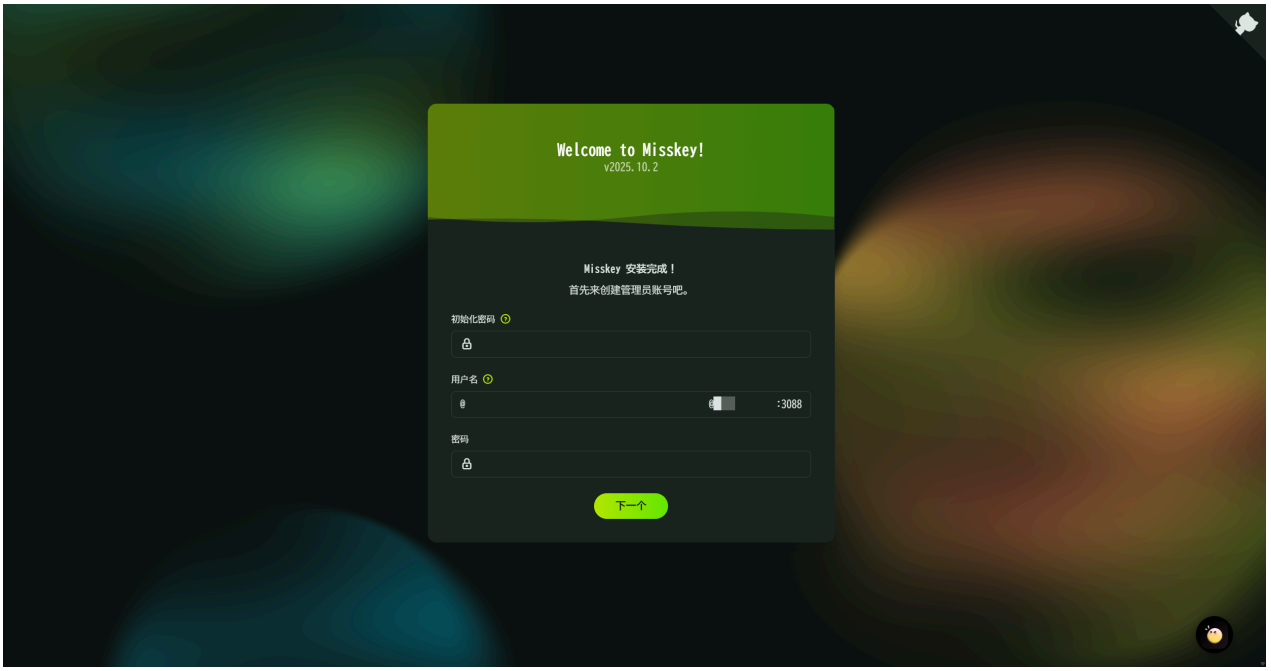
`--rm`为命令执行后自动删除容器

`web`为`compose.yml`的服务名称

`pnpm run init`为在容器内执行的命令

```
# 初始化 Misskey 数据库和设置
sudo docker compose run --rm web pnpm run init
[+] Running 21/21
 ✓ redis Pulled
              8.2s # 拉取redis
   ✓ f637881d1138 Pull complete
              3.4s
......
   ✓ d75b3becd998 Pull complete
              5.5s
 ✓ db Pulled
             7.8s # 拉取PostgreSQL
   ✓ 2d35ebdb57d9 Already exists
             0.0s
......
   ✓ 9229d0c336e2 Pull complete
              5.0s
[+] Creating 4/4
 ✓ Network misskey_internal_network  Created
             0.0s # 创建容器内部网络
 ✓ Network misskey_external_network  Created
             0.1s # 创建容器外部网络
 ✓ Container misskey-redis-1         Created
             0.4s # 创建redis容器
 ✓ Container misskey-db-1            Created
             0.4s # 创建postgrespl容器
[+] Running 2/2
 ✓ Container misskey-db-1     Started
              0.2s
 ✓ Container misskey-redis-1  Started
              0.3s

> misskey@2025.10.2 init /misskey

# 后台启动docker容器
docker compose up -d
# 流式输出容器logs
docker compose logs -f
```

4. 成功启动，访问ip:port



5. 将本地镜像迁移到其他服务器

`docker save` & `docker load`

准备两台服务器：
在**目标服务器**上运行(ssh端口为55520)

```
sudo apt install rsync
```

在**本服务器**上运行：

```
docker images

misskey-web  latest    77795cb799f1   34 minutes ago   2.65GB
```

```
docker compose down

docker save -o misskey.tar misskey-web:latest

# compose.yml内有三个服务：web redis db
# docker save -o misskey.tar misskey-web:latest只会导出web的镜像misskey-web:latest

ls -la misskey.tar

# -rw------- 1 jay jay 2751643136 Nov 12 14:49 misskey.tar
```

```
sudo apt install rsync
rsync -avP -e "ssh -p 55520" misskey.tar jay@47.79.89.127:~/misskey/
```

```
jay@s868537:~/misskey$ rsync -avP -e "ssh -p 55520" misskey.tar jay@47.79.89.127:~/misskey/
jay@47.79.89.127's password:
sending incremental file list
created directory /home/jay/misskey
misskey.tar
  2,751,643,136 100%   24.08MB/s    0:01:48 (xfr#1, to-chk=0/1)

sent 2,752,315,026 bytes  received 75 bytes  23,829,567.97 bytes/sec
total size is 2,751,643,136  speedup is 1.00
jay@s868537:~/misskey$
```

传输成功

在**目标服务器**上：

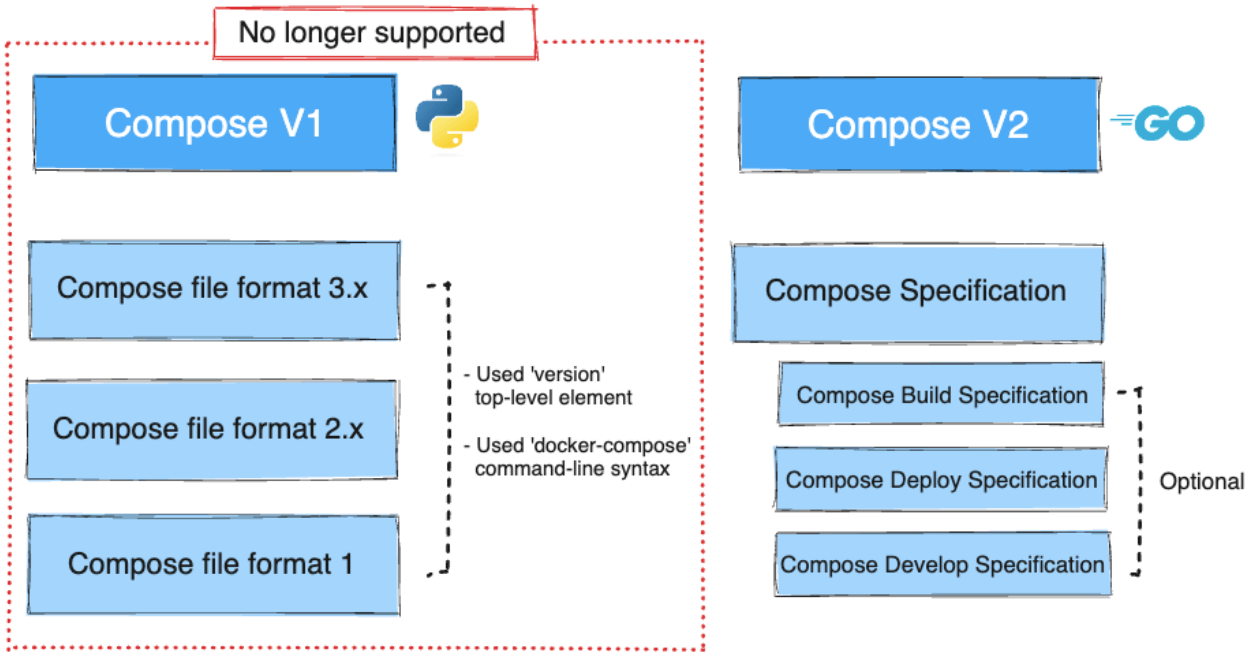
```
jay@s32796:~$ ls
DVWA-master  bbb.gemini.request.py  beszel            misskey  nginx.conf
antiGFW      bbb.gemini_nodejs      infinigpt-matrix  myenv    xftp-server
jay@s32796:~$ cd misskey
jay@s32796:~/misskey$ ls -la
total 2687164
drwxr-xr-x  2 jay jay       4096 Nov 12 15:00 .
drwx------ 19 jay jay       4096 Nov 12 14:58 ..
-rw-------  1 jay jay 2751643136 Nov 12 14:49 misskey.tar
jay@s32796:~/misskey$
```

导入镜像：

```
docker load -i misskey.tar

docker images
```

```
jay@s32796:~/misskey$ docker load -i misskey.tar
6c4c763d22d0: Loading layer   77.9MB/77.9MB
34f63e9295b6: Loading layer   21.5kB/21.5kB
16248767172b: Loading layer  144.4MB/144.4MB
bf9d82a144a9: Loading layer  7.212MB/7.212MB
9a1d79c05eac: Loading layer  3.584kB/3.584kB
001718a0d4b2: Loading layer  476.5MB/476.5MB
870a9e25b0b7: Loading layer   5.12kB/5.12kB
e6606fb8d48c: Loading layer   29.7MB/29.7MB
5f70bf18a086: Loading layer  1.024kB/1.024kB
3cd53c376dfe: Loading layer  1.543GB/1.543GB
3bf0a21c9214: Loading layer    171kB/171kB
8b70774aa6e0: Loading layer  36.86kB/36.86kB
60ad9d1f1049: Loading layer  20.99kB/20.99kB
6fdf066684a1: Loading layer   21.5kB/21.5kB
5084fe41ac4d: Loading layer  192.5MB/192.5MB
0217241dee4d: Loading layer  152.1kB/152.1kB
73185975070b: Loading layer  52.74kB/52.74kB
8baf87ec9b58: Loading layer  77.82kB/77.82kB
487ed0782e47: Loading layer  13.28MB/13.28MB
36c3873cd634: Loading layer  16.07MB/16.07MB
41727a87ca4a: Loading layer  250.5MB/250.5MB
Loaded image: misskey-web:latest
jay@s32796:~/misskey$ docker images
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
misskey-web                 latest    77795cb799f1   54 minutes ago   2.65GB
```

Docker Compose



- **V1**: 原始的Python实现，已不再支持
- **V2**: 基于Go语言重写，性能更好，功能更丰富

而debian12的默认源使用的就是过时的`V1 Compose`，启动命令为`docker-compose up -d`

新版`V2 Compose`，启动命令为`docker compose up -d`

Compose从V1的docker二级命令变为V2的模块插件

查看docker compose 版本

```
docker compose version

Docker Compose version v2.39.4
```

手动安装Docker Compose插件

```
DOCKER_CONFIG=${DOCKER_CONFIG:-$HOME/.docker}
mkdir -p $DOCKER_CONFIG/cli-plugins
curl -SL https://github.com/docker/compose/releases/download/v2.40.3/docker-compose-linux-
x86_64 -o $DOCKER_CONFIG/cli-plugins/docker-compose

chmod +x $DOCKER_CONFIG/cli-plugins/docker-compose
```

将docker run的命令变为docker-compose.yaml格式

相关网站：https://www.composerize.com/

```
docker run -p 80:80 -v /var/run/docker.sock:/tmp/docker.sock:ro --restart always --log-opt
max-size=1g nginx
```

```
version: "3"
services:
    nginx:
        ports:
            - 80:80
        volumes:
            - /var/run/docker.sock:/tmp/docker.sock:ro
        restart: always
        logging:
            options:
                max-size: 1g
        image: nginx
```

其他docker技术&容器化/虚拟化学习路径

docker registery

LXC

https://linuxcontainers.org/zh-cn/lxc/introduction/

Podman

https://podman.io/

Kubernetes

https://kubernetes.io/zh-cn/docs/home/

Proxmox

https://www.proxmox.com/en/

参考资料

> https://docs.docker.com/
>
> https://zh.wikipedia.org/wiki/Docker
>
> https://misskey-hub.net/cn/docs/for-admin/install/guides/docker/
>
> https://docker.1panel.dev/