

- [docker学习](#)
 - [ubuntu安装docker](#)
 - [docker操作](#)
 - [dockerfile学习](#)
 - [dockerfile指令](#)

docker学习

ubuntu安装docker

1. 卸载自带的docker

```
apt remove docker docker-engine docker.io containerd runc
```

2. 更新Ubuntu软件包列表和已安装软件的版本

```
sudo apt update
```

3. 安装docker依赖

```
apt install ca-certificates curl gnupg lsb-release
```

4. 添加Docker官方GPG密钥

```
curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
```

5. 添加Docker的软件源

```
sudo add-apt-repository "deb [arch=amd64] http://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
```

6. 安装Docker

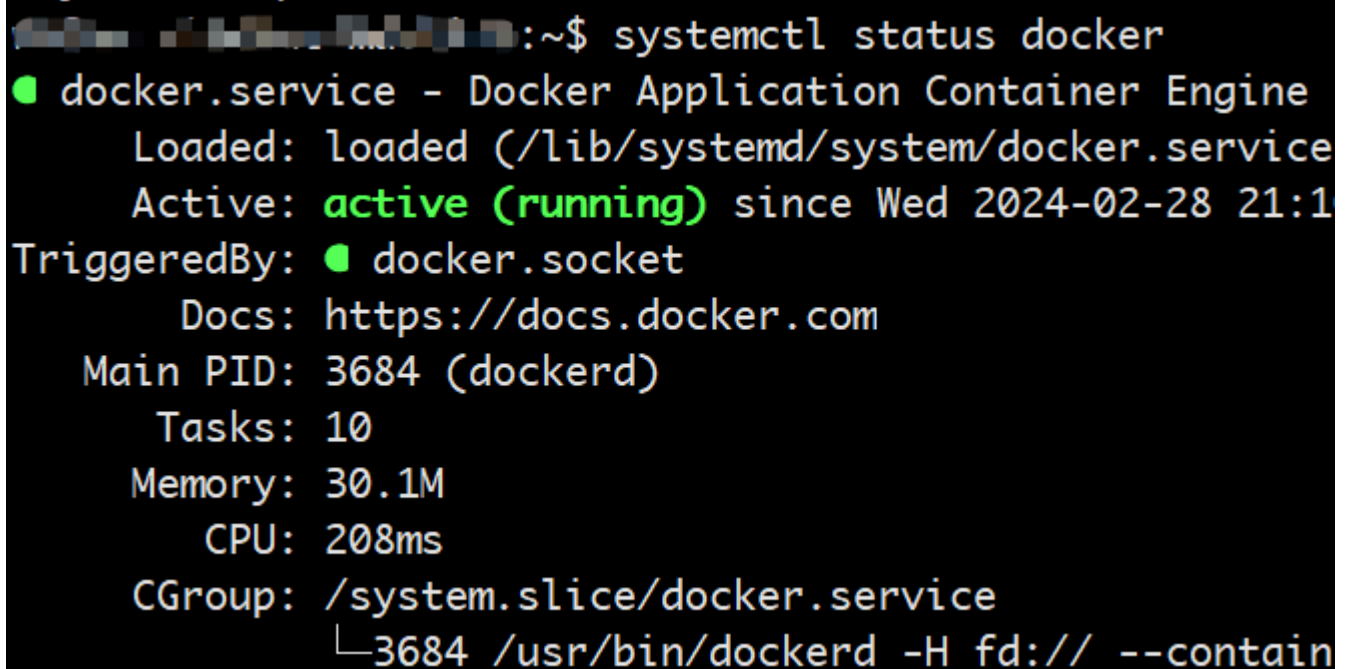
```
apt install docker-ce docker-ce-cli containerd.io
```

7. 配置用户组 默认情况下，只有root用户和docker组的用户才能运行Docker命令。我们可以将当前用户添加到docker组，以避免每次使用Docker时都需要使用sudo。命令如下：

```
sudo usermod -aG docker $USER
```

8. 运行Docker

```
systemctl start docker
```

A terminal window with a black background and white text. The prompt is a user's home directory (~\$). The command 'systemctl status docker' has been executed. The output shows that the 'docker.service' is loaded and active (running). It was triggered by 'docker.socket'. The main PID is 3684 (dockerd). It has 10 tasks, uses 30.1M of memory, and has a CPU time of 208ms. The CGroup is '/system.slice/docker.service', and the command being run is '3684 /usr/bin/dockerd -H fd:// --contain'.

9. 安装工具

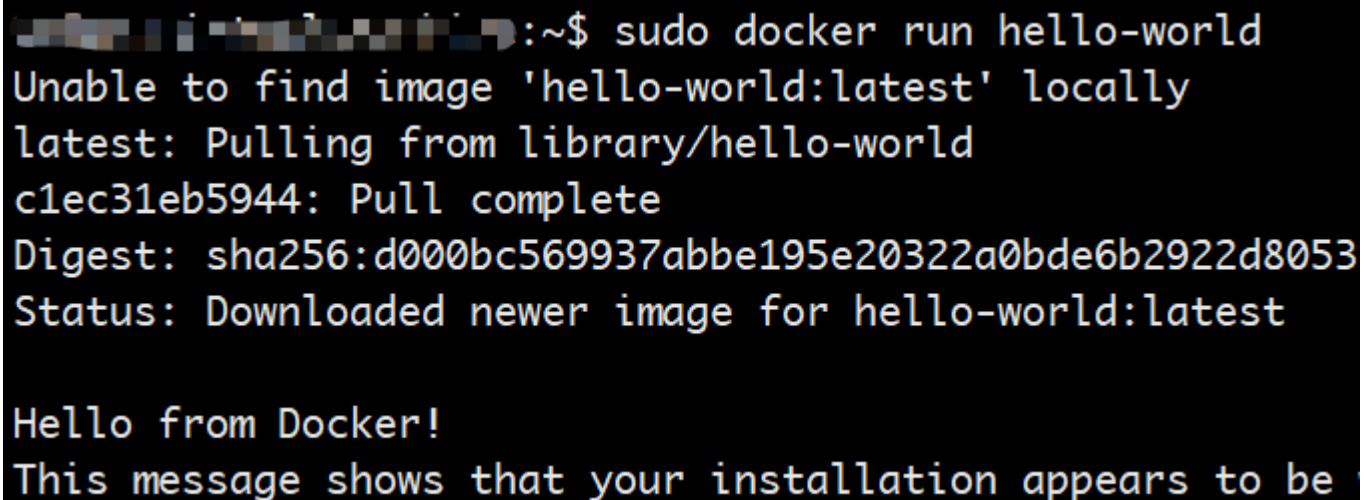
```
apt -y install apt-transport-https ca-certificates curl software-properties-common
```

10. 重启Docker

```
systemctl restart docker
```

11. 验证是否成功

```
sudo docker run hello-world
```

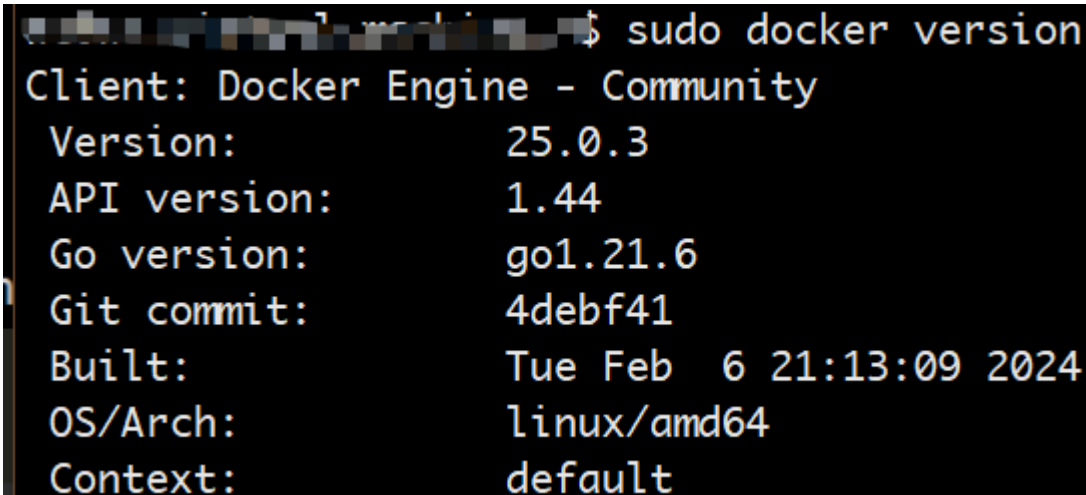
A terminal window with a black background and yellow text. The prompt is a blurred username followed by '~\$'. The command 'sudo docker run hello-world' is entered. The output shows that the image 'hello-world:latest' was not found locally, so it was pulled from the library. The pull is complete, and the digest is shown. The status indicates that a newer image was downloaded. Finally, the container outputs 'Hello from Docker!' and a message stating that the installation appears to be successful.

```
username:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:d000bc569937abbe195e20322a0bde6b2922d8053
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be
```

12. 查看docker版本

```
sudo docker version
```

A terminal window with a black background and yellow text. The prompt is a blurred username followed by '\$'. The command 'sudo docker version' is entered. The output displays various Docker client and engine details in a key-value format.

```
username$ sudo docker version
Client: Docker Engine - Community
Version:           25.0.3
API version:       1.44
Go version:        go1.21.6
Git commit:        4debf41
Built:             Tue Feb  6 21:13:09 2024
OS/Arch:           linux/amd64
Context:           default
```

docker操作

1. 查看镜像

```
sudo docker images
```

2. 拉取镜像

```
# 搜索是否存在nginx的镜像
sudo docker search nginx
# 拉取nginx镜像
sudo docker pull
```

```
root@██-virtual-machine:~# docker pull nginx
Using default tag: latest

^[[Alatest: Pulling from library/nginx
a2abf6c4d29d: Pull complete
a9edb18cadd1: Pull complete
589b7251471a: Pull complete
186b1aaa4aa6: Pull complete
b4df32aa5a72: Pull complete
a0bcbecc962e: Pull complete
Digest: sha256:0d17b565c37bcbd895e9d92315a05c1c3
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

3. 运行容器

```
# 运行容器
sudo docker run -d -p 81:80 nginx
# 运行并进入容器
sudo docker run -d -p 81:80 -it nginx bash
# -d 参数让容器在后台运行
# --rm 容器执行结束后自动从历史记录中删除这容器
# -p 参数让容器的80端口映射到主机的81端口 -p 宿主机端口: 容器暴露端口
```

4. 查看运行中的容器

```
sudo docker ps
```

5. 停止容器

```
sudo docker stop 容器ID
```

6. 进入一个运行着的容器

```
sudo docker exec -it 容器ID bash
```

```
wc@wc-virtual-machine:~$ docker exec -it 673977b7ca6c bash
root@673977b7ca6c:/# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 11 (bullseye)"
NAME="Debian GNU/Linux"
VERSION_ID="11"
VERSION="11 (bullseye)"
VERSION_CODENAME=bullseye
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
root@673977b7ca6c:/#
```

7. 删除镜像

```
sudo docker rmi 镜像ID
```

删除镜像之前，不能有依赖的容器记录，需要先删除容器记录可以先用 `docker ps -a` 查询容器记录

```
wc@wc-virtual-machine:~$ docker rmi d2c94e258dcb
Untagged: hello-world:latest
Untagged: hello-world@sha256:d000bc569937abbe195e20322a0bde6b2922d805332fd6d8a68
Deleted: sha256:d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c2234249495f87264ac5a
Deleted: sha256:ac28800ec8bb38d5c35b49d45a6ac4777544941199075dff8c4eb63e093aa81e
wc@wc-virtual-machine:~$
```

8. 删除容器记录

```
sudo docker rm 容器ID
```

```
wc@wc-virtual-machine:~$ docker rm e26d9e238784
e26d9e238784
```

9. 导出镜像

```
docker image save hello-world > /opt/hello-world.tar
```

10. 导入镜像

```
docker image load -i /opt/hello-world.tar
```

11. 查看容器日志

```
sudo docker logs -f 容器ID  
# -f 参数让容器日志实时输出
```

dockerfile学习

```
sudo vim Dockerfile  
  
# Dockerfile内容如下:  
``shell  
FROM nginx  
RUN echo '<meta charset="utf-8">用docker运行nginx</meta>' >  
/usr/share/nginx/html/index.html  
  
# 构建dockerfile镜像  
dcoker build .  
  
# 修改镜像名字  
docker tag 镜像ID my-nginx  
  
# 运行该镜像  
docker run -d -p 80:80 my-nginx
```

dockerfile指令

Dockerfile 指令	说明
FROM	指定基础镜像，用于后续的指令构建。
MAINTAINER	指定Dockerfile的作者/维护者。（已弃用，推荐使用LABEL指令）
LABEL	添加镜像的元数据，使用键值对的形式。
RUN	在构建过程中在镜像中执行命令。
CMD	指定容器创建时的默认命令。（可以被覆盖）
ENTRYPOINT	设置容器创建时的主要命令。（不可被覆盖）
EXPOSE	声明容器运行时监听的特定网络端口。
ENV	在容器内部设置环境变量。
ADD	将文件、目录或远程URL复制到镜像中。
COPY	将文件或目录复制到镜像中。
VOLUME	为容器创建挂载点或声明卷。
WORKDIR	设置后续指令的工作目录。
USER	指定后续指令的用户上下文。
ARG	定义在构建过程中传递给构建器的变量，可使用 "docker build" 命令设置。
ONBUILD	当该镜像被用作另一个构建过程的基础时，添加触发器。
STOPSIGNAL	设置发送给容器以退出的系统调用信号。
HEALTHCHECK	定义周期性检查容器健康状态的命令。
SHELL	覆盖Docker中默认的shell，用于RUN、CMD和ENTRYPOINT指令。

FROM: 定制的镜像都是基于 FROM 的镜像，这里的 nginx 就是定制需要的基础镜像

RUN: 用于执行后面跟着的命令行命令。**RUN <命令行命令>**或者**RUN ["可执行文件", "参数1", "参数2"]**

注意：Dockerfile 的指令每执行一次都会在 docker 上新建一层。所以过多无意义的层，会造成镜像膨胀过大。例如：

```
FROM centos
RUN yum -y install wget
RUN wget -O redis.tar.gz "http://download.redis.io/releases/redis-5.0.3.tar.gz"
RUN tar -xvf redis.tar.gz
```

以上执行会创建 3 层镜像。可简化为以下格式：

```
FROM centos
RUN yum -y install wget \
    && wget -O redis.tar.gz "http://download.redis.io/releases/redis-5.0.3.tar.gz" \
    && tar -xvf redis.tar.gz
```

COPY：复制指令，从上下文目录中复制文件或者目录到容器里指定路径。

```
COPY [--chown=<user>:<group>] <源路径1>... <目标路径>
COPY [--chown=<user>:<group>] ["<源路径1>",... "<目标路径>"]
COPY hom* /mydir/
COPY hom?.txt /mydir/
```

ADD：ADD 指令和 COPY 的使用格类似（同样需求下，官方推荐使用 COPY）

- ADD 的优点：在执行 <源文件> 为 tar 压缩文件的话，压缩格式为 gzip, bzip2 以及 xz 的情况下，会自动复制并解压到 <目标路径>。
- ADD 的缺点：在不解压的前提下，无法复制 tar 压缩文件。会令镜像构建缓存失效，从而可能会令镜像构建变得比较缓慢。具体是否使用，可以根据是否需要自动解压来决定。

CMD：类似于 RUN 指令，用于运行程序，但二者运行的时间点不同：

- CMD 在 docker run 时运行。
- RUN 是在 docker build。

```
CMD <shell 命令>
CMD ["<可执行文件或命令>","<param1>","<param2>"...]
CMD ["<param1>","<param2>"...] # 该写法是为 ENTRYPOINT 指令指定的程序提供默认参数
```

ENTRYPOINT：类似于 CMD 指令，但其不会被 docker run 的命令行参数指定的指令所覆盖，而且这些命令行参数会被当作参数送给 ENTRYPOINT 指令指定的程序。

WORKDIR：指定工作目录。用 WORKDIR 指定的工作目录，会在构建镜像的每一层中都存在。以后各层的当前目录就被改为指定的目录，如该目录不存在，WORKDIR 会帮你建立目录。

`docker build` 构建镜像过程中的，每一个 `RUN` 命令都是新建的一层。只有通过 `WORKDIR` 创建的目录才会一直存在。

格式: `WORKDIR <工作目录路径>`

EXPOSE: 仅仅是声明端口。

作用:

- 帮助镜像使用者理解这个镜像服务的守护端口，以方便配置映射。
- 在运行时使用随机端口映射时，也就是 `docker run -P` 时，会自动随机映射 `EXPOSE` 的端口。 格式: `EXPOSE <端口1> [<端口2>...]`

ENV: 设置环境变量，定义了环境变量，那么在后续的指令中，就可以使用这个环境变量。 格式:

```
ENV <key> <value>
ENV <key1>=<value1> <key2>=<value2>...
```