

Docker Tutorial

2022软院暑培 8月30日 周恒宇

注：本文基于的 Docker 版本为 20.10.17

什么是 Docker

Docker 是一个用于开发、配置、运行应用的开放平台。Docker 将应用与环境隔离，让应用能够快捷地交付。

Docker 可以将应用打包，运行在称为容器的隔离环境中。容器中包括了应用运行所需的一切环境，摆脱了对宿主机的配置需求。与 VM 不同，Docker 基于 Linux 内核的“虚拟化”，通过命名空间等机制将内核资源分割，创建独立的虚拟环境。因此，Docker 依赖于宿主机的 Linux 内核，隔离程度低于 VM，但在启动速度、占用资源等方面优于 VM。

安装 Docker

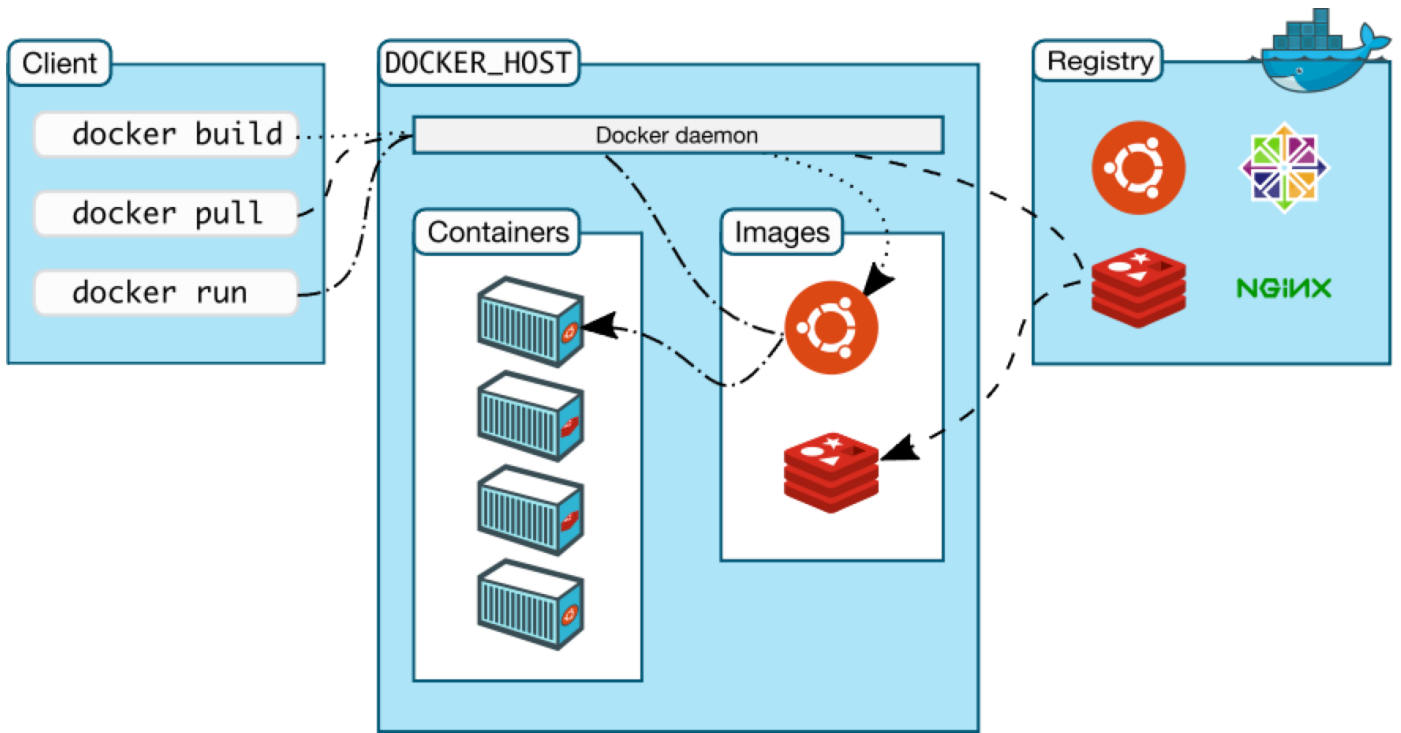
安装 Docker 存在两种选择：安装 Docker Desktop 应用或仅安装 Docker Engine。Docker 提供了用于 Windows，macOS 和 Linux 的 Docker Desktop 应用，基于虚拟机实现，并提供了宿主机上的 CLI，可以直接在宿主机的终端中使用。Docker Engine 只可以被安装在 Linux 系统下。

安装桌面应用可以参照 [Docker 官网](#) 的内容。安装 Docker Engine 可以参考 [Docker 官网](#) 或 [TUNA](#) 提供的帮助。

Why Docker

- 开发
 - Docker 可以作为轻量级的虚拟机，拥有独立的环境
- 部署
 - Docker 镜像将程序运行所需的环境和配置进行了打包，便于进行大规模的部署
- 交付
 - Docker 容器可以为应用提供一致的环境，非常适用于 CI/CD

Docker 基础概念



Docker 基于 Client-Server 架构。我们使用的 Docker CLI 通过 UNIX socket 与 Docker daemon 通讯，后者负责容器的创建与管理。

在 Docker 中有两个最重要的概念：镜像 (image) 与容器 (container)。

前面说到，我们会使用 Docker 将应用和所需的环境打包，这个打包就是镜像。镜像运行在容器中，每个容器都是一个独立的实例。镜像本身是只读的，其作为容器创建的模板存在。因此，我们可以使用同一个镜像运行多个容器。

由于镜像本身就包含了应用程序所需的一切依赖，可以直接复制到安装了 Docker 的其他计算机上运行，大大简化了部署的流程。

通常，我们会基于其他镜像创建新的镜像。比如，我们想要部署一个 Python 应用，我们就可以从 `python` 镜像开始，加入我们应用的代码，设置启动命令，形成一个新的镜像。

当我们有了一个镜像，就可以使用 `docker run` 命令启动一个容器。

```
1 | docker run hello-world
```

这条命令就会基于 `hello-world` 镜像创建一个容器并运行。这个镜像通过 Docker Hub 下载，类似于 Node 的 npm 或者 Python 的 pypi，用户可以在这里上传并分享自己创建的镜像。

接下来我们通过一个简单的例子介绍如何创建自己的镜像

使用 Dockerfile 创建一个镜像

我们用 Flask 写一个简单的应用

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello_world():
6     return 'Hello, Docker!'
```

将其命名为 `app.py`，再创建一个 `requirements.txt`，包括我们的依赖库 (Flask)

```
1 Flask==2.2.2
```

现在，我们来创建一个 Dockerfile。Dockerfile 告诉 Docker 如何构建我们的镜像。

```
1 # syntax=docker/dockerfile:1
2
3 FROM python:3.10-slim-buster
4
5 WORKDIR /app
6
7 COPY requirements.txt requirements.txt
8 RUN pip3 install -r requirements.txt
9
10 COPY . .
11
12 CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

`# syntax=docker/dockerfile:1` 这一行指定了 syntax 版本，目前最新的是 `docker/dockerfile:1`

`FROM` 命令指定了我们所基于的镜像。我们所选用的 `python:3.10-slim-buster` 包括了 Python 运行时。

`WORKDIR` 命令指定了后续命令的工作目录。

`COPY` 命令将文件从外部复制到镜像中，第一个参数相对于镜像构建的目录，即和 Dockerfile 同级，第二个参数指要被复制到镜像文件系统中的什么位置，由于前面指定了工作目录，会被复制到 `/app/requirements.txt`。

`RUN` 命令在构建镜像时运行，这里我们调用 `pip3` 安装依赖库。

`COPY . .` 将当前目录的所有文件复制到镜像的 `/app` 目录下

`CMD` 指定了容器的启动命令，在 `docker run` 启动容器时运行。

现在，我们的目录下有三个文件：

```
1 .
2 |— Dockerfile
3 |— app.py
4 |— requirements.txt
```

一切都就绪了，我们运行 `docker build` 命令来构建镜像

```
1 | docker build -t docker-tutorial-backend:latest .
```

`-t` 是 `--tag` 的缩写，指定镜像的标签。

运行 `docker images`，我们就可以看到新创建的镜像了

```
1 | REPOSITORY          TAG          IMAGE ID      CREATED      SIZE
2 | docker-tutorial-backend  latest      cf34efab2d8e  8 seconds ago  130MB
```

我们使用 `docker run docker-tutorial-backend` 运行之

```
1 | rhacoal@rhacoal-mac backend % docker run docker-tutorial-backend
2 | * Debug mode: off
3 | WARNING: This is a development server. Do not use it in a production deployment. Use
4 | a production WSGI server instead.
5 | * Running on all addresses (0.0.0.0)
6 | * Running on http://127.0.0.1:5000
7 | * Running on http://172.17.0.2:5000
8 | Press CTRL+C to quit
```

但如果我们想要尝试在浏览器中打开 `http://127.0.0.1:5000`，并不能正常访问



无法访问此网站

127.0.0.1 拒绝了我们的连接请求。

请试试以下办法：

- 检查网络连接
- 检查代理服务器和防火墙

ERR_CONNECTION_REFUSED

详情

重新加载

这是因为 Docker 容器有着自己独立的网络，并不能从外部直接访问。我们需要使用端口映射，将容器内的端口暴露到外部：

```
1 | docker run -d -p 8080:5000 docker-tutorial-backend:latest
```

这个命令将容器内的 5000 端口暴露在宿主机的 8080 端口上，此时我们就可以使用 `http://127.0.0.1:8080` 访问到容器中的 Flask 应用。在这个命令中我们还使用了 `-d` 参数，指定容器应在后台启动。运行中的容器可以通过 `docker ps` 查看。

保存状态

前面我们提到，Docker 的镜像是只读的。实际上，每个镜像是由多个层 (layer) 组成的，Dockerfile 中每个修改文件系统的命令都会新建一层，每层的内容是该层相对于上一层文件系统的变动。Docker 使用存储驱动 (storage driver) 将所有层映射为一个完整的文件系统，默认值为 `overlay2`，即使用 OverlayFS，这是一个目前的 Linux 发行版都支持的文件系统。

当我们创建一个新的容器，Docker 会在所使用的镜像上附加一个可读写的层，构成容器的文件系统。在这一层上的读写仅限于当前容器，对其他容器不可见。如果我们重建这个容器，所有的更改就会丢失，包括应用的设置和数据。

因此，Docker 为我们提供了一种持久化机制：卷 (volume)。我们可以通过 `docker run` 的 `-v` 参数，将卷挂载于容器的指定路径。

```
1 docker run -v mysql-data:/var/lib/mysql \  
2     -e MYSQL_ROOT_PASSWORD=change_password \  
3     -d \  
4     mysql:latest
```

(`-e` 参数用于指定环境变量，可以参考容器在 Docker Hub 上的页面，了解有哪些可以设置的环境变量)

我们在这里使用的是具名卷 (named volume)，创建一个名为 `mysql-data` 的卷并挂载在容器文件系统的 `/data` 路径。我们也可以使用绑定挂载 (bind mount)：

```
1 docker run -v ./data:/var/lib/mysql \  
2     -e MYSQL_ROOT_PASSWORD=change_password \  
3     -d \  
4     mysql:latest
```

将当前目录下的 `./data` 目录挂载到容器中。这个路径可以是相对或绝对的，但需要包含至少一个斜杠，用于区分具名卷和挂载卷。

卷中的数据在容器删除后不会被清除，因此，我们只需要在启动容器时指定映射的卷，就可以继续容器之前的状态。

进入容器进行调试

在创建 Dockerfile 的过程中，我们难免会遇到很多问题。一些在宿主机上可以正常运行的命令，可能在镜像中就无法完成。在这种情况下，我们可以通过 `docker exec`，在镜像中执行命令，检查命令的错误提示。命令参数如下：

```
1 docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

我们使用 `docker ps` 确定正在运行的容器名称或 ID

1	CONTAINER ID	IMAGE	COMMAND	CREATED
	STATUS	PORTS	NAMES	
2	06be87496e69	docker-tutorial-backend	"python3 -m flask ru..."	40 minutes ago
	Up 40 minutes	wonderful_ganguly		

比如通过 `docker exec 06 ls /app`，就可以列出容器中 `/app` 目录下的文件（CONTAINER 参数可以使用容器的完整名称，或者容器 ID 的前缀，只要这个前缀是唯一的）

我们还可以指定 OPTIONS 为 `-it` (`--interactive --tty`) 分配一个 pseudo tty，运行交互式程序，比如 bash：

```
1 | docker exec -it 06 bash
```

我们就获得了一个容器中的终端，帮助我们调试。使用 `exit` 或 `Ctrl-D` 退出 bash。

使用 Docker Compose 管理应用

当我们的应用逐渐复杂，启动参数越来越多，或者需要同时运行多个容器，直接使用 `docker` 命令可能不再是一个好的选择。

Docker 为我们提供了一套解决方案：Docker Compose。我们只需要编写一个 YAML 文件定义我们的服务，就可以用一条命令启动所需的所有容器。

安装 Docker Compose 请参考官方教程：[Install Docker Compose | Docker](#)

这里我们以 HackMD 为例（来自 [Docker Deployment | HackMD](#)）

```

1 | services:
2 |   database:
3 |     image: postgres:11.6-alpine
4 |     environment:
5 |       - POSTGRES_USER=codimd
6 |       - POSTGRES_PASSWORD=change_password
7 |       - POSTGRES_DB=codimd
8 |     volumes:
9 |       - "database-data:/var/lib/postgresql/data"
10 |    restart: always
11 |  codimd:
12 |    image: hackmdio/hackmd:2.4.2-cjk
13 |    environment:
14 |      - CMD_DB_URL=postgres://codimd:change_password@database/codimd
15 |      - CMD_USECDN=false
16 |    depends_on:
17 |      - database
18 |    ports:
19 |      - "3000:3000"
20 |    volumes:
21 |      - upload-data:/home/hackmd/app/public/uploads
22 |    restart: always

```

```
23 volumes:
24     database-data: {}
25     upload-data: {}
```

我们将其保存为 `docker-compose.yml`。

```
1 # 启动
2 docker compose up -d
3 # 停止
4 docker compose down
```

在这个 compose 文件中，我们指定了两个容器：`database` 和 `codimd`，分别运行 `postgres` 和 `hackmdio/hackmd` 镜像，设置了环境变量 (environment)、卷 (volumes)、端口映射 (ports)、自动重启策略 (restart)。如果我们要使用自己创建的镜像，可以用 `build: <Dockerfile 所在的路径>` 设置。比如当 Dockerfile 和 `docker-compose.yml` 在同一目录下时，可以用这样的方式构建并启动容器：

```
1 services:
2     backend:
3         build: .
4         ports:
5             - "8080:5000"
```

本例中，我们并没有将数据库的 5432 端口暴露在宿主机上，这是因为 Docker Compose 会为文件中的所有服务默认创建一个内部网络。在这个网络中，可以使用服务名作为主机名访问其他服务，比如 `codimd` 服务的 `CMD_DB_URL` 环境变量中，使用 `database` 作为主机名就可以访问到运行在 `database` 服务中的 PostgreSQL 数据库。如果不使用 Docker Compose，我们也可以使用 `docker network` 命令管理网络。

分发 Docker 镜像

我们可以通过将 Docker 镜像或容器导出的方式，分发我们的镜像。Docker 提供了一组命令用于完成这一系列操作：

`docker save` 与 `docker load`

`docker save` 用于将镜像导出为 tar 包，我们以前面创建的 `docker-tutorial-backend` 镜像为例：

```
1 docker save docker-tutorial-backend:latest > backend.tar
```

打包默认输出到标准输出流，可以通过 pipe operator 重定向到文件，或者使用 `-o backend.tar` 指定输出文件。

对应的，`docker load` 可以用于加载一个镜像。我们先移除原本的镜像：

```
1 docker image rm docker-tutorial-backend:latest
2 docker load < backend.tar
```

这样，我们就成功的导入了镜像 `docker-tutorial-backend:latest`。

docker export 与 docker import

`docker export` 可以将容器导出为 tar 包。该命令会将容器的文件系统打包。

```
1 | docker export 9f > backend-fs.tar
```

此时的 `backend-fs.tar` 只包含文件系统，不包含镜像的元数据。

对于这种包，我们通过 `docker import` 将其导入为新的镜像：

```
1 | docker import backend-fs.tar backend2:latest
```

我们可以在 `docker images` 中看到新导入的镜像。但如果我们通过 `docker run backend2` 尝试运行这一镜像：

```
1 | docker: Error response from daemon: No command specified.
```

这是正常的：使用 `export` 导出的包只包括了文件系统，并不包括工作目录或启动命令等信息。

我们可以在 `import` 时通过 `--change (-c)` 参数，将 Dockerfile 命令作用在导入的镜像上：

```
1 | docker import -c "WORKDIR /app" \  
2 |     -c "CMD python3 -m flask run --host 0.0.0.0" \  
3 |     backend-fs.tar backend2:latest
```

此时，我们创建了一个可以正常运行的 Docker 镜像。

Docker Hub

在上述方法之外，我们还可以将镜像推送到 Docker Hub 上进行分发，具体的操作可以参考 Docker 提供的教程 [Docker Hub Quickstart](#)。

References

1. [Docker Overview | Docker Documentation](#)
2. [Docker Deployment - HackMD](#)