DockerCompose

1. 概念

1.1 基本概念

> **Note**
>
> 是`docker compose`工具使用的配置文件。`docker compose`是一个用于定义和运行多容器Docker应用程序的工具。
>
> 使用Docker Compose，你可以通过一个YAML文件来配置你的应用服务，然后使用一个简单的命令来启动和停止所有服务。

- 可以将 `docker-compose.yaml` 文件理解为一种**自动化的脚本**，这个脚本定义了如何使用 Docker 容器来部署和管理一个多容器应用程序。
- 与dockerfile的关系——将docker-compose.yaml 视作脚本，可以调用dockerfile同时创建多个容器，从而不需要命令行手动创建容器

1.2 作用

`docker-compose.yaml`文件（自动化脚本）的作用包括：

1. **定义服务**：
   - 在 `docker-compose.yaml` 文件中，您可以定义多个服务，每个服务可以基于一个 Docker 镜像或 Dockerfile 构建的镜像运行。
2. **使用 Dockerfile**：
   - 如果您在项目中使用 Dockerfile 来构建自定义镜像，可以在 `docker-compose.yaml` 文件中通过 `build` 或 `image` 指令引用这个 Dockerfile，以确保使用正确的镜像来启动容器。
3. **创建容器**：
   - 通过运行 `docker-compose up` 命令，Docker Compose 会根据 `docker-compose.yaml` 文件中定义的服务创建容器。如果服务指定了基于 Dockerfile 构建的镜像，Docker Compose 会先构建这个镜像，然后使用它来创建容器。

4. **管理容器**：
    - Docker Compose 提供了命令来管理这些容器的生命周期，包括启动（`up`）、停止（`down`）、重启（`restart`）和更多。
5. **网络和卷**：
    - `docker-compose.yaml` 文件还允许您定义网络和卷，这些网络和卷可以被不同的服务共享，从而实现容器间的通信和数据持久化。
6. **环境变量和配置**：
    - 您可以在 `docker-compose.yaml` 文件中设置环境变量和配置，这些设置会传递给容器，从而允许您根据不同的环境（开发、测试、生产）调整应用程序的行为。
7. **扩展性**：
    - Docker Compose 还支持服务的扩展，例如，您可以指定服务的副本数量，以便于在测试或生产环境中扩展应用程序。

1.3 简单示例

- 利用 dockerfile 构建容器：

```
# 假设dockerfile文件内容是简单的web应用
FROM node:14
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 3000
CMD ["npm", "start"]
```

```
# yaml 文件内容
# build . 即使用当前目录下的dockerfile来构建(build)容器
version: '3'
services:
  web:
    build: .
    ports:
      - "3000:3000"
    volumes:
      - .:/app
```

- 更复杂的示例：

```
# 假设你有一个Web应用，它由一个Web服务器和一个数据库组成。你可以使用docker-
compose.yaml来定义这个多容器应用。

version: '3.8'  # 使用的Docker Compose文件版本

services:
  web:
    image: nginx:latest  # 使用最新的Nginx镜像
    ports:
      - "80:80"  # 将容器的80端口映射到主机的80端口
    volumes:
      - web-data:/usr/share/nginx/html  # 挂载数据卷到Nginx的静态文件目录
    depends_on:
      - db  # 确保数据库服务先启动

  db:
    image: postgres:latest  # 使用最新的Postgres镜像
    environment:
      POSTGRES_DB: mydatabase  # 设置环境变量，定义数据库名
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - db-data:/var/lib/postgresql/data  # 挂载数据卷到Postgres的数据目录

volumes:
  web-data:  # 定义web服务的数据卷
  db-data:  # 定义db服务的数据卷
```

2. docker compose 相关配置命令

1. **version**：
    - 指定使用的Docker Compose文件格式版本。这有助于确保文件的语法与Docker Compose的版本兼容。
    - 例如：`version: '3.8'`

2. **services**:
   - 定义了一组服务，每个服务代表一个容器。
   - 例如：

```
services:
  web:
    image: nginx # 使用nginx镜像
```

3. **image**:
   - 指定容器使用的镜像。
   - 例如：`image: nginx:latest`
4. **build**:
   - 指定构建上下文和Dockerfile路径，用于构建镜像。
   - 例如：

```
build: ./dir

# 如果是使用当前目录下的dockerfile
build: .
```

5. **ports**:
   - 映射容器端口到宿主机。
   - 例如：`ports: - "5000:5000"`
6. **volumes**:
   - 挂载一个卷到容器，用于数据持久化。
   - 例如：

```
volumes:
  - mydata:/var/data
```

7. **volumes_from**:
   - 从另一个服务挂载卷。
   - 例如：`volumes_from: - service_name`
8. **networks**:
   - 定义服务连接的网络。
   - 例如：

```
networks:
  - mynet
```

9. **environment**:
   - 设置环境变量。
   - 例如：

```
environment:
  - DEBUG=1
  - NODE_ENV=production
```

10. **depends_on**:
    - 定义服务启动的依赖关系。
    - 例如：`depends_on: - db`
11. **command**:
    - 覆盖容器启动后默认执行的命令。
      如果在dockerfile中使用`CMD`来编写容器启动后的执行的命令，会被docker compose中的command覆盖
    - 例如：`command: ["./run.sh", "arg1", "arg2"]`
12. **entrypoint**:
    - 覆盖容器的入口点。
      入口点：即容器启动后先执行什么命令，可以执行脚本
    - 例如：`entrypoint: /code/entrypoint.sh`
    - 如果同时使用entrypoint 和 command：

```
# 同时使用entrypoint command
# entrypoint 指定了要执行的脚本文件， command中的参数将被传递给entrypoint 指定的脚本
# 即 command起到输入参数的作用
version: '3'
services:
  web:
    build: .
    entrypoint: /app/start.sh
    command: ["arg1", "arg2"] # 将arg1 arg2 两个参数输入到start.sh中
```

```
# 假设脚本内容如下
#!/bin/bash
# start.sh
echo "Starting with arguments: $@"
exec npm start "$@"
```

13. **links**:
    - 定义服务之间的连接关系，允许使用容器名作为主机名进行网络通信。
    - 例如：`links: - db:database`
14. **restart**:
    - 指定容器的重启策略。
    - 例如：`restart: always`
15. **cap_add** 和 **cap_drop**:
    - 分别添加或删除容器的Linux能力。
    - 例如：`cap_add: - SYS_ADMIN`
16. **ulimits**:
    - 设置Linux ulimit配置。
    - 例如：

```
ulimits:
  nproc: 65535
  nofile:
    soft: 20000
    hard: 40000
```

17. **logging**:
    - 配置容器的日志。
    - 例如：

```
logging:
  driver: syslog
  options:
    syslog-address: "tcp://192.168.0.42:123"
```

18. **deploy**:

- 用于指定服务的部署配置，仅在使用Docker Swarm模式时有效。
- 例如：

```yaml
deploy:
  replicas: 3
  resources:
    limits:
      cpus: '0.5'
      memory: 50M
  restart_policy:
    condition: on-failure
```

3. 常用命令

更多命令看该网站

Docker Compose常用命令_dockercompose命令-CSDN博客

3.1 restart, start, stop -- 启动和停止服务

```
# 前台启动，启动项目中的所有服务。
docker-compose up

# 后台启动，启动所有服务并在后台运行。
docker-compose up -d

# 停止所有服务。
docker-compose stop

restart
docker-compose restart # 重启服务容器。
docker-compose restart # 重启工程中所有服务的容器
docker-compose restart nginx # 重启工程中指定服务的容器

start
docker-compose start # 启动服务容器。
docker-compose start # 启动工程中所有服务的容器
docker-compose start nginx # 启动工程中指定服务的容器
```

```
stop
docker-compose stop停止服务容器。
docker-compose stop # 停止工程中所有服务的容器
docker-compose stop nginx # 停止工程中指定服务的容器
```

3.2 build -- 构建和重构服务

```
# 构建服务的镜像
docker-compose build

# 如果服务镜像不存在，则构建镜像并启动服务。
docker-compose up -build

# 重构服务。
docker-compose up --force-recreate
```

3.3 ps, logs -- 查看服务信息

```
# 查看项目中所有服务的信息。
docker-compose ps

# 查看容器的日志。
docker-compose logs
# docker-compose logs 查看服务容器的输出日志。
# 默认情况下，docker-compose将对不同的服务输出使用不同的颜色来区分。
# 可以通过--no-color来关闭颜色。
# 输出日志，不同的服务输出使用不同的颜色来区分
docker-compose logs
# 跟踪日志输出
docker-compose logs -f
# 关闭颜色
docker-compose logs --no-color

# 查看日志
docker-compose logs web  # 参考 1.9 docker-compose.yml 文件内容
```

```
# 在服务镜像的容器中执行命令。
docker-compose exec service_name commandv
```

3.4 down -- 删除所有容器

```
#  删除服务容器(容器)
docker-compose down
```

3.5 run -- 指定容器上执行命令

```
# run
docker-compose run # 在指定服务容器上执行一个命令。
docker-compose run nginx echo "helloworld" # 在工程中指定服务的容器上执行 echo "helloworld"
```

3.6 exec命令 -- 进入指定容器

```
# exec
docker-compose exec # 进入服务容器。
docker-compose exec nginx bash # 进入工程中指定服务的容器
docker-compose exec --index=1 nginx bash # 当一个服务拥有多个容器时，可通过 --index 参数进入到该服务下的任何容器

sudo docker-compose exec jobmanager ./bin/sql-client.sh -f sql/flink_kafka1.sql
sudo docker-compose exec jobmanager ./bin/flink list
sudo docker logs -f -t --since="2023-05-08" --tail=200 flink_taskmanager_1
sudo docker-compose logs -f taskmanager

sudo docker-compose exec jobmanager bash
sudo docker-compose exec jobmanager ./bin/flink cancel 8d8cc94d73f7bd0c4cdc557264553a04
```