Docker(四)：Docker 三剑客之 Docker Compose

原创： 纯洁的微笑 纯洁的微笑 2018-03-22

前两篇文章我们介绍了 Dockerfile 的使用Docker(二)：Dockerfile 使用介绍，我们知道使用一个 Dockerfile 模板文件可以定义一个单独的应用容器，如果需要定义多个容器就需要服务编排。服务编排有很多种技术方案，今天给大家介绍 Docker 官方产品 Docker Compose 。

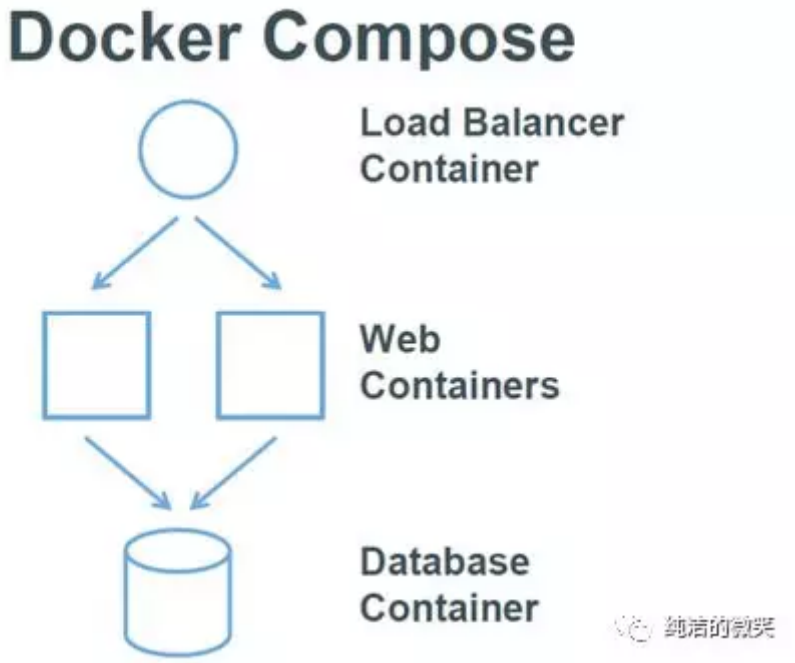
Dockerfile 可以让用户管理一个单独的应用容器；而 Compose 则允许用户在一个模板（YAML 格式）中定义一组相关联的应用容器（被称为一个 project，即项目），例如一个 Web 服务容器再加上后端的数据库服务容器等。

Docker Compose 介绍

Docker-Compose 是 Docker 的一种编排服务，是一个用于在 Docker 上定义并运行复杂应用的工具，可以让用户在集群中部署分布式应用。

通过 Docker-Compose 用户可以很容易地用一个配置文件定义一个多容器的应用，然后使用一条指令安装这个应用的所有依赖，完成构建。Docker-Compose 解决了容器与容器之间如何管理编排的问题。

Docker Compose 工作原理图



Compose 中有两个重要的概念：

- 服务 (service) ：一个应用的容器，实际上可以包括若干运行相同镜像的容器实例。
- 项目 (project) ：由一组关联的应用容器组成的一个完整业务单元，在 docker-compose.yml 文件中定义。

一个项目可以由多个服务（容器）关联而成，Compose 面向项目进行管理，通过子命令对项目中的一组容器进行便捷地生命周期管理。

Compose 项目由 Python 编写，实现上调用了 Docker 服务提供的 API 来对容器进行管理。因此，只要所操作的平台支持 Docker API，就可以在其上利用 Compose 来进行编排管理。

Docker Compose 安装

Docker Compose 是 Docker 的独立产品，因此需要安装 Docker 之后在单独安装 Docker Compose .

方法一：

```
curl -L https://github.com/docker/compose/releases/download/1.19.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
#查看版本
docker-compose version
```

方法二：

```
#安装pip
yum -y install epel-release
yum -y install python-pip
#确认版本
pip --version
#更新pip
pip install --upgrade pip
#安装docker-compose
pip install docker-compose
#查看版本
docker-compose version
```

推荐使用方法一进行安装，安装成功后输入 `docker-compose version` 会返回 docker-compose 的版本信息，如下：

```
[root@localhost ~]# docker-compose version
docker-compose version 1.19.0, build 9e633ef
docker-py version: 2.7.0
CPython version: 2.7.13
```

```
5. OpenSSL version: OpenSSL 1.0.1t 3 May 2016
```

出现以上信息，表明 docker-compose 安装成功

安装补全工具(可选)

为了方便我们输入命令，也可以安装 Docker 的补全提示工具帮忙我们快速输入命令

```
1. #安装
2. yum install bash-completion
3. #下载docker-compose脚本
4. curl -L https://raw.githubusercontent.com/docker/compose/$(docker-compose version --short)/contrib/completion/bash/docker-compose > /etc/bash_completion.d/docker-compose
```

快速上手

没有什么比来一个小例子练练手更好的学习方法了，我们以官网上的简单示例来看看 docker compose 的使用方法。

我们设计这么一个场景，使用 Python 启动一个 Web 服务，输出一个 `hello()`方法，每次访问的时候在 Redis 缓存中进行计数，并且将统计的结果打印到页面中。

第一步，创建 Python 服务

创建项目路径:

```
1. mkdir composetest
2. cd composetest
```

在目录下创建 `app.py`文件

```
1. import time
2. import redis
3. from flask import Flask
4. app = Flask(__name__)
5. cache = redis.Redis(host='redis', port=6379)
6. def get_hit_count():
7.   retries = 5
8.   while True:
9.   try:
10.   return cache.incr('hits')
11.   except redis.exceptions.ConnectionError as exc:
12.   if retries == 0:
13.   raise exc
14.   retries -= 1
15.   time.sleep(0.5)
16. @app.route('/')
17. def hello():
```

```
    count = get_hit_count()
    return 'Hello World! I have been seen {} times.\n'.format(count)
if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

在这个例子中，redis 使用了容器内的网络默认端口是6379。这段 Python 程序的内容就是，启动后连接 Redis 并且输出 `hello()`方法，当每次访问的时候累计访问次数并且将结果放回到页面。

在同目录下创建 `requirements.txt`文件，添加项目依赖的python包：

```
flask
redis
```

Flask 是 Python 中一个微型的 Web 开发框架。

第二步，创建 Dockerfile

我们来写一个 Dockerfile 来定义 Docker 镜像，此镜像包含了 Python 的依赖包和 Python 环境。

同样在此目录下，我们创建一个 Dockerfile 文件。

```
FROM python:3.4-alpine
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

这段代码表示:

- 使用基础镜像 Python 3.4
- 将当前目录映射到镜像 `/code`目录下
- 设置工作目录为 `/code`
- 安装 Python 依赖包
- 启动 `app.py` 程序

第三步，使用 Compose 文件定义一个服务

在当期目录下，我们创建一个 docker-compose.yml 文件，内容如下：

```
version: '2'
services:
  web:
  build: .
  ports:
  - "5000:5000"
  redis:
  image: "redis:alpine"
```

这个 Compose 文件定义了两个服务，一个 Pyhon Web 服务和 redis 服务。

- Pyhon Web 服务：使用 Dockerfile 构建了当前镜像。将 Web 容器内部的5000端口映射到 host 的5000端口；并将 Web 容器与 redis 容器连接。
- redis服务：该容器直接由官方的 redis 镜像创建。

第四步，使用 Compose 编译启动应用

使用命令 `docker-compose up`启动

```
version: '2'
services:
  web:
  build: .
  command: python app.py
  ports:
  - "5000:5000"
  volumes:
  - .:/code
  redis:
  image: "redis:alpine"
```

启动成功之后，在浏览器访问： `http://ipaddress:5000/` ，返回如下：

```
Hello World! I have been seen 1 times.
```

刷新再次访问返回

1. Hello World! I have been seen 2 times.

不断的刷新数字会不断的增长。

Docker Compose 常用命令

使用 `docker-compose up -d`在后台启动服务

```
[root@localhost composetest]# docker-compose up -d
Starting composetest_web_1 ...
Starting composetest_web_1 ... done
```

使用 `docker-compose ps`命令查看启动的服务

```
[root@localhost composetest]# docker-compose ps
  Name  Command  State  Ports
------------------------------------------------------------------------------------
composetest_redis_1 docker-entrypoint.sh redis ... Up 6379/tcp
composetest_web_1 python app.py Up 0.0.0.0:5000->5000/tcp
```

使用 `docker-compose stop`停止服务。

```
[root@localhost composetest]# docker-compose stop
```

```
Stopping composetest_web_1 ... done
Stopping composetest_redis_1 ... done
```

其它常用命令

```
#查看帮助
docker-compose -h
# -f 指定使用的 Compose 模板文件，默认为 docker-compose.yml，可以多次指定。
docker-compose -f docker-compose.yml up -d
#启动所有容器，-d 将会在后台启动并运行所有的容器
docker-compose up -d
#停用移除所有容器以及网络相关
docker-compose down
#查看服务容器的输出
docker-compose logs
#列出项目中目前的所有容器
docker-compose ps
#构建（重新构建）项目中的服务容器。服务容器一旦构建后，将会带上一个标记名，例如对于 web 项目中的一个 db 容器，可能是 web_db。可以随时在项目目录下运行 docker-compose build 来重新构建服务
docker-compose build
#拉取服务依赖的镜像
docker-compose pull
#重启项目中的服务
docker-compose restart
#删除所有（停止状态的）服务容器。推荐先执行 docker-compose stop 命令来停止容器。
docker-compose rm
#在指定服务上执行一个命令。
docker-compose run ubuntu ping docker.com
#设置指定服务运行的容器个数。通过 service=num 的参数来设置数量
docker-compose scale web=3 db=2
#启动已经存在的服务容器。
docker-compose start
#停止已经处于运行状态的容器，但不删除它。通过 docker-compose start 可以再次启动这些容器。
docker-compose stop
```