Docker Compose

思考

前面我们使用 Docker 的时候，定义 Dockerfile 文件，然后**使用 docker build、docker run -d --name -p等命令**操作容器。然而微服务架构的应用系统一般包含若干个微服务，每个微服务一般都会部署多个实例，如果每个微服务都要手动启停，那么效率之低，维护量之大可想而知

使用 Docker Compose 可以轻松、高效的管理容器，它是一个用于定义和运行多容器 Docker 的应用程序工具

简介

`Docker Compose` 是 Docker 官方编排（Orchestration）项目之一，**负责快速的部署分布式应用。**

描述

`Compose` 项目是 Docker 官方的开源项目，负责实现对 Docker 容器集群的快速编排。从功能上看，跟 `OpenStack` 中的 `Heat` 十分类似。

其代码目前在 https://github.com/docker/compose 上开源。

`Compose` 定位是 「定义和运行多个 Docker 容器的应用（Defining and running multi-container Docker applications）」，其前身是开源项目 Fig。

通过第一部分中的介绍，我们知道使用一个 `Dockerfile` 模板文件，可以让用户很方便的定义一个单独的应用容器。然而，在日常工作中，经常会碰到需要多个容器相互配合来完成某项任务的情况。例如要实现一个 Web 项目，除了 Web 服务容器本身，往往还需要再加上后端的数据库服务容器，甚至还包括负载均衡容器等。

`Compose` 恰好满足了这样的需求。它允许用户通过一个单独的 `docker-compose.yml` 模板文件（YAML 格式）来定义一组相关联的应用容器为一个项目（project）。

`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。

`Compose` 的默认管理对象是项目，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。

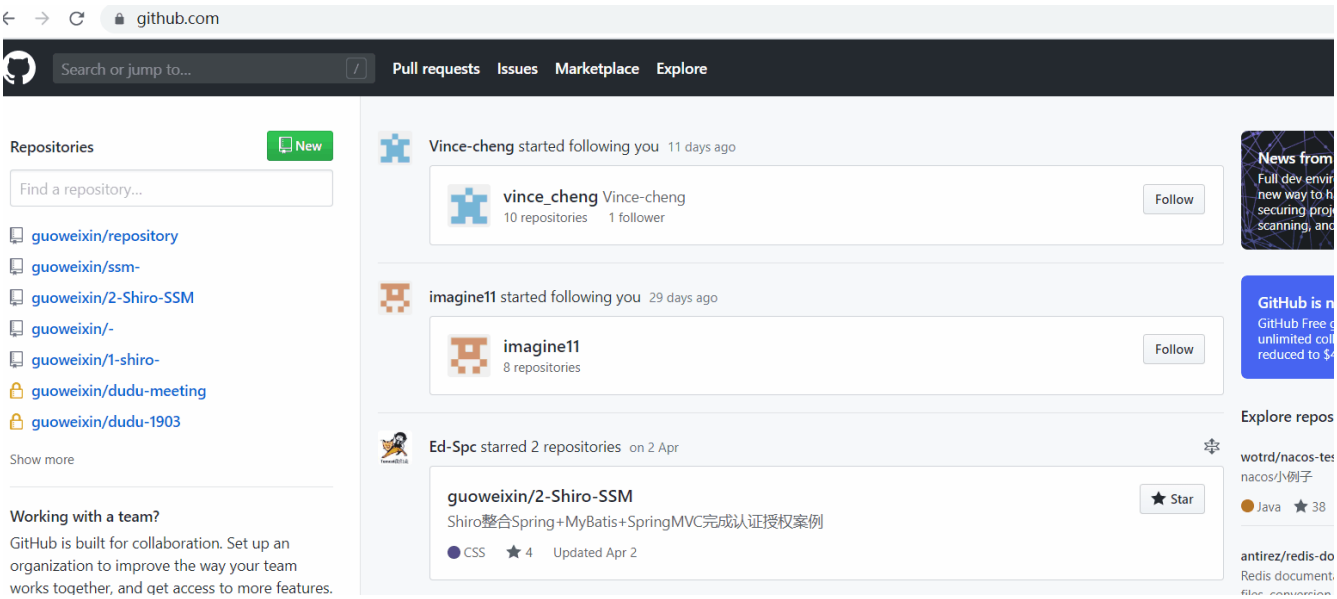
`Compose` 项目由 Python 编写，实现上调用了 Docker 服务提供的 API 来对容器进行管理。因此，只要所操作的平台支持 Docker API，就可以在其上利用 `Compose` 来进行编排管理。

Docker Compose安装与卸载

> Compose 支持 Linux、macOS、Windows 10 三大平台。
>
> Compose 可以通过 Python 的包管理工具 pip 进行安装，也可以直接下载编译好的二进制文件使用，甚至能够直接在 Docker 容器中运行。
>
> Docker for Mac、Docker for Windows自带 docker-compose。
>
> 二进制文件，安装 Docker 之后可以直接使用

官方安装

```
#1 github官方网站 搜索Docker compose
#2 根据如下动态图片示例，找到下载好的二进制文件
https://github.com/docker/compose/releases/download/1.25.5/docker-compose-Linux-x86_64
#2.1将下载好的文件拖入Linux 并剪切到 /usr/local目录下
mv docker-compose-Linux-x86_64 /usr/local
#2.2 修改名称（为后面方便调用） 并 修改其为可执行文件
mv docker-compose-Linux-x86_64  docker-compose
chmod 777 docker-compose
mv docker-compose  /usr/local/bin/
```



或通过 curl的方式下载。 https://docs.docker.com/compose/install/

```
#3 运行以下命令以下载Docker Compose的当前稳定版本：

sudo curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
# 4将可执行权限应用于二进制文件：
sudo chmod +x /usr/local/bin/docker-compose
```

```
#5测试是否安装
$ docker-compose --version
docker-compose version 1.25.5, build 1110ad01
```

卸载

```
#如果是 二进制包方式安装的，删除二进制文件即可
sudo rm /usr/local/bin/docker-compose

#如果是通过 pip 安装的，则执行如下命令即可删除。
sudo pip uninstall docker-compose
```

Docker Compose使用

术语

> 首先介绍几个术语。
>
> - 服务（`service`）：一个应用容器，实际上可以运行多个相同镜像的实例。
> - 项目（`project`）：由一组关联的应用容器组成的一个完整业务单元。
>
> 可见，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。

Docker-compose创建容器

> 通过一个单独的 `docker-compose.yml` 模板文件（YAML 格式）来定义一组相关联的应用容器为一个项目（project）。

yml格式描述：

```
1、yml文件以缩进代表层级关系
2、缩进不允许使用tab只能使用空格
3、空格的个数不重要，只要相同层级的元素左对齐即可（建议2个）
4、大小写敏感
5、数据格式为，名称:(空格)值
```

```
#一、 k: (空格)v:表示一对键值对(空格不能省略),以空格控制层级关系,只要是左对齐的数据,都是同一级别;
server:
  port: 8083
  path: /helloBoot
#二、数组(用-表示数组中的一个元素):
animal:
- cat
- dag
```

示例1

用compose的方式管理一个Tomcat容器和MySQL

```
#1 管理文件夹，创建相应的目录
mkdir -p /opt/docker_mysql_tomcat/
```

```
#2 在如上目录中 编写创建 docker-compose.yml配置文件
编写 docker-compose.yml 文件，这个是 Compose 使用的主模板文件。
```

```
version: '3.1'
services:
  mysql:                                        # 服务的名称
    restart: always                             # 只要docker启动，容器会随着启动
    image: daocloud.io/library/mysql:5.7.6      # 指定镜像路径信息（默认官方镜像地址）
    container_name: mysql-3306                   # 指定容器名称 --name
    ports:
      - 3306:3306                              #指定端口号映射
    environment:
      MYSQL_ROOT_PASSWORD: root                 #指定MYSQL ROOT用户的密码
      TZ: Asiz/Shanghai                         #指定时区
    volumes:
      - /opt/docker_mysql_tomcat/mysql/data:/var/lib/mysql #映射mysql的数据目录到宿主机，保存数据
      - /opt/docker_mysql_tomcat/mysql/conf/mysqld.cnf:/etc/mysql/mysql.conf.d/mysqld.cnf
      #把mysql的配置文件映射到容器的相应目录
  tomcat:
    restart: always
    image: daocloud.io/library/tomcat:8.5.15-jre8
    container_name: tomcat-8080
    ports:
      - 8080:8080
    environment:
      TZ: Asiz/Shanghai
    volumes:
      - /opt/docker_mysql_tomcat/tomcat/webapps:/usr/local/tomcat/webapps
      - /opt/docker_mysql_tomcat/tomcat/logs:/usr/local/tomcat/logs
```

`小提示: docker stop $(docker ps -qa) 用于停掉所有的docker容器`

```
# 3启动(执行命令创建容器)(在)
docker-compose up -d

#默认执行的文件名：docker-compose.yml(且需要在当前上下文路径中) 。如果说文件名不是默认的需要使用下面的指令：
docker-compose -f 文件名.后缀名 up -d
```

连接访问测试Tomcat

连接访问测试MySQL

```
#1 Bash进入mysql容器：
sudo docker exec -it 7f /bin/bash
mysql -u root -p

use mysql;
grant all privileges on *.* to 'root'@'%';
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'root';
flush privileges;
```

```
#1 基于docker-compose.yml启动管理容器
docker-compose up -d
```

```
#2 关闭或删除容器
docker-compose down  #关闭并删除容器
docker-compose stop  #关闭容器
```

```
#3 开启容器
docker-compose start
```

```
#4 查看由docker-compose管理的容器
docker-compose ps
```

```
#5 查看日志
docker-compose logs -f
```

Docker-compose 常用命令

> build 构建或重建服务
>
> help 命令帮助
>
> kill 杀掉容器
>
> logs 显示容器的输出内容
>
> port 打印绑定的开放端口
>
> ps 显示容器

pull 拉取服务镜像

restart 重启服务

rm 删除停止的容器

run 运行一个一次性命令

scale 设置服务的容器数目

start 开启服务

stop 停止服务

up 创建并启动容器

down

常用命令示例：

docker-compose up -d nginx 构建启动nignx容器

docker-compose exec nginx bash 登录到nginx容器中

docker-compose down 删除所有nginx容器,镜像

docker-compose ps 显示所有容器

docker-compose restart nginx 重新启动nginx容器

docker-compose run --no-deps --rm php-fpm php -v 在php-fpm中不启动关联容器，并容器执行php -v 执行完成后删除容器

docker-compose build nginx 构建镜像 。

docker-compose build --no-cache nginx 不带缓存的构建。

docker-compose logs nginx 查看nginx的日志

docker-compose logs -f nginx 查看nginx的实时日志

docker-compose config -q 验证（docker-compose.yml）文件配置，当配置正确时，不输出任何内容，当文件配置错误，输出错误信息。

docker-compose events --json nginx 以json的形式输出nginx的docker日志

docker-compose pause nginx 暂停nignx容器

docker-compose unpause nginx 恢复ningx容器

docker-compose rm nginx 删除容器（删除前必须关闭容器）

docker-compose stop nginx 停止nignx容器

docker-compose start nginx 启动nignx容器

Docker仓库

1. 国内 Docker 仓库

阿里云

网易云

时速云

DaoCloud

2. 国外 Docker 仓库

Docker Hub

Quay

Tomcat搭建集群

```
1、创建指定管理目录
mkdir /opt/docker-tomcat-cluster
cd /opt/docker-tomcat-cluster  #进入该目录
```

```
2、编写docker-compose.yml
```

```
version: '3.1'
services:
   tomcat-8080:
    restart: always
    image: tomcat
    container_name: tomcat-8080
    ports:
      - 8080:8080
    environment:
      TZ: Asiz/Shanghai
    volumes:
      - /opt/docker-tomcat-cluster/tomcat-8080/webapps:/usr/local/tomcat/webapps
      - /opt/docker-tomcat-cluster/tomcat-8080/logs:/usr/local/tomcat/logs

.....................................
```

```
3、测试 在每一个tomcat下创建ROOT/index.html。用于访问即可。
cp -r ROOT/ /opt/docker-tomcat-cluster/tomcat-8081/webapps
```

Redis集群搭建

redis集群需要至少要三个master节点，我们这里搭建三个master节点，并且给每个master再搭建一个slave节点，总共6个redis节点，这里用一台机器（可以多台机器部署，修改一下ip地址就可以了）部署6个redis实例，三主三从，搭建集群的步骤如下：

主	服务名称	端口号	从	服务名称	端口号
master1	127.0.0.1	6381	slave1	127.0.0.1	6384
master1	127.0.0.1	6382		127.0.0.1	6385
master1	127.0.0.1	6383		127.0.0.1	6386

1、创建Redis节点安装目录

```
mkdir docker-redis-cluster
```

2、依次创建文件夹。并依次修改配置文件

```
创建文件夹 6381-6386
mkdir redis-6381

复制redis.conf到此目录下
cp redis.conf  /opt/docker-redis-cluster/redis-6381/
```

```
# 关闭保护模式 用于公网访问
protected-mode no
port 6381
# 开启集群模式
cluster-enabled yes
#不改也可
#cluster-config-file nodes-6381.conf
#cluster-node-timeout 5000
# 日志文件
pidfile /var/run/redis_6381.pid
# 此处绑定ip 可以是阿里内网ip 和 本地ip 可以直接注释掉该项
#bind 127.0.0.1
#用于连接主节点密码
masterauth guoweixin
#设置redis密码 各个节点请保持密码一致
requirepass guoweixin
```

3、依次创建文件夹。并依次修改配置文件

按如上方式。修改6份。分别为 6381-6386。

4、编写docker-compose文件内容

```
version: '3'
services:
 redis-6381:
  container_name: redis-6381
  image: redis
  command: redis-server /etc/usr/local/redis.conf
  network_mode: "host"
  volumes:
   - /opt/docker-redis-cluster/redis-6381/redis.conf:/etc/usr/local/redis.conf
   - /opt/docker-redis-cluster/redis-6381/data:/data

 redis-6382:
  container_name: redis-6382
  image: redis
  command: redis-server /etc/usr/local/redis.conf
  network_mode: "host"
  volumes:
   - /opt/docker-redis-cluster/redis-6382/redis.conf:/etc/usr/local/redis.conf
   - /opt/docker-redis-cluster/redis-6382/data:/data

 redis-6383:
  container_name: redis-6383
  image: redis
  command: redis-server /etc/usr/local/redis.conf
  network_mode: "host"
  volumes:
   - /opt/docker-redis-cluster/redis-6383/redis.conf:/etc/usr/local/redis.conf
   - /opt/docker-redis-cluster/redis-6383/data:/data

 redis-6384:
  container_name: redis-6384
  image: redis
  command: redis-server /etc/usr/local/redis.conf
  network_mode: "host"
  volumes:
   - /opt/docker-redis-cluster/redis-6384/redis.conf:/etc/usr/local/redis.conf
   - /opt/docker-redis-cluster/redis-6384/data:/data

 redis-6385:
  container_name: redis-6385
  image: redis
  command: redis-server /etc/usr/local/redis.conf
  network_mode: "host"
  volumes:
   - /opt/docker-redis-cluster/redis-6385/redis.conf:/etc/usr/local/redis.conf
   - /opt/docker-redis-cluster/redis-6385/data:/data

 redis-6386:
  container_name: redis-6386
  image: redis
  command: redis-server /etc/usr/local/redis.conf
```

```
  network_mode: "host"
  volumes:
   - /opt/docker-redis-cluster/redis-6386/redis.conf:/etc/usr/local/redis.conf
   - /opt/docker-redis-cluster/redis-6386/data:/data
```

5、启动并测试

```
在/opt/docker-redis-cluster目录下 启动:
docker-compose up -d

查看是否全部启动成功。如未成功，需要排查错误（再执行下一步）
docker-compose ps

可通过远程工具连接任意一节点，看是否密码有效。
```

6、创建集群

Redis 5版本后 通过redis-cli 客户端命令来创建集群。

> 创建集群的命令
>
> --cluster create
>
> --cluster-replicas 1

```
docker exec -it redis-6381 redis-cli --cluster  create  -a guoweixin 127.0.0.1:6381 127.0.0.1:6382
127.0.0.1:6383 127.0.0.1:6384 127.0.0.1:6385 127.0.0.1:6386 --cluster-replicas 1
```

7、 Redis Cluster集群验证

在某台机器上（或）连接集群的6381端口的节点:

```
普通Redis方式:
  redis-cli -h 127.0.0.1  -c -p 6381 -a guoweixin  :加参数 -c 可连接到集群

Docker方式:
 docker exec -it redis-6381  redis-cli -h 127.0.0.1  -c -p 6381 -a guoweixin
```

redis cluster在设计的时候，就考虑到了去中心化，去中间件，也就是说，集群中的每个节点都是平等的关系，都是对等的，每个节点都保存各自的数据和整个集群的状态。每个节点都和其他所有节点连接，而且这些连接保持活跃，这样就保证了我们只需要连接集群中的任意一个节点，就可以获取到其他节点的数据

基本命令

info replication通过Cluster Nodes命令和Cluster Info命令来看看集群效果

```
[root@localhost src]# redis-cli -h 127.0.0.1 -c -p 7001  -a guoweixin
127.0.0.1:7001> keys *
(empty list or set)
127.0.0.1:7001> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=127.0.0.1,port=7004,state=online,offset=1344,lag=0
master_replid:d84426dd8ac124eef42bbf1cbea5f3c7b3a7c85e
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:1344
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:1344
127.0.0.1:7001>
```

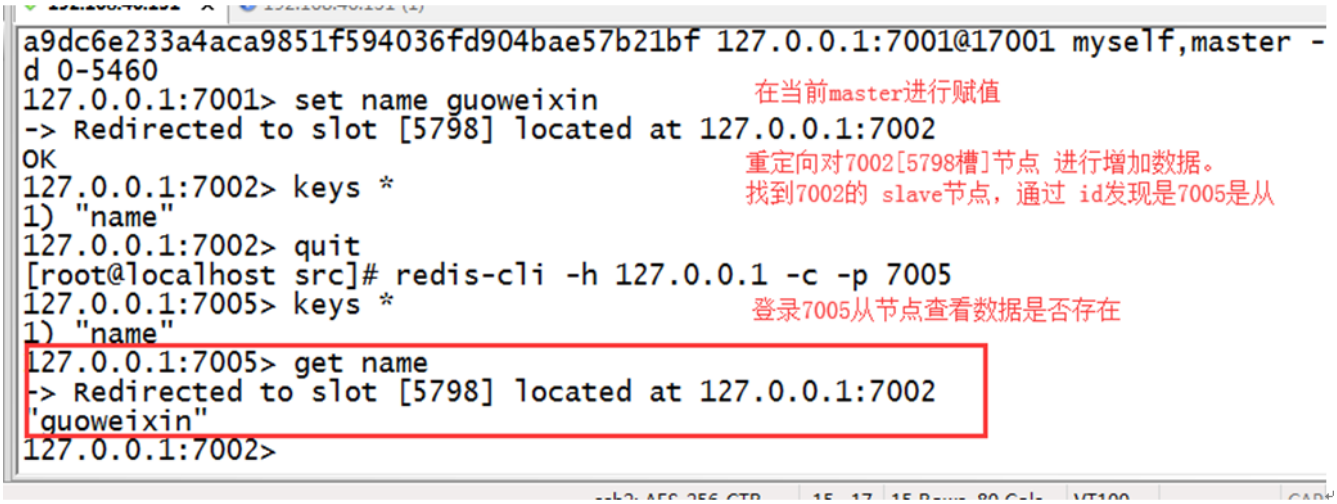
输入命令 cluster nodes

```
127.0.0.1:7001> cluster nodes
07102ad2b3d40d8153dbb826d22cf3477e8a885a 127.0.0.1:7006@17006 slave 01ee02664dd7362e7a78bbea12e
7a7c 0 1555674614954 6 connected
01ee02664dd7362e7a78bbea12e5e67e08e27a7c 127.0.0.1:7003@17003 master - 0 1555674614000 3 connec
-16383
e4620a3828e295aa592e74b579ee77ace4be711e 127.0.0.1:7002@17002 master - 0 1555674612000 2 connec
10922
02a163c8742e30872da2f2c0009aaf025876e191 127.0.0.1:7004@17004 slave a9dc6e233a4aca9851f594036fd
21bf 0 1555674613944 4 connected
b66ac338d54d22b485ef5dc46b28cd749f031ec3 127.0.0.1:7005@17005 slave e4620a3828e295aa592e74b579e
711e 0 1555674615963 5 connected
a9dc6e233a4aca9851f594036fd904bae57b21bf 127.0.0.1:7001@17001 myself,master - 0 1555674615000 1
d 0-5460
127.0.0.1:7001> set name guoweixin
-> Redirected to slot [5798] located at 127.0.0.1:7002
```

```
07102ad2b3d40d8153dbb826d22cf3477e8a885a 127.0.0.1:7006@17006 slave 01ee02664dd7362e7a78bbea12e5e67e08e27a7c 0 1555674614
01ee02664dd7362e7a78bbea12e5e67e08e27a7c 127.0.0.1:7003@17003 master - 0 1555674614000 3 connected 10923-16383
e4620a3828e295aa592e74b579ee77ace4be711e 127.0.0.1:7002@17002 master - 0 1555674612000 2 connected 5461-10922
02a163c8742e30872da2f2c0009aaf025876e191 127.0.0.1:7004@17004 slave a9dc6e233a4aca9851f594036fd904bae57b21bf 0 1555674613
b66ac338d54d22b485ef5dc46b28cd749f031ec3 127.0.0.1:7005@17005 slave e4620a3828e295aa592e74b579ee77ace4be711e 0 1555674615
a9dc6e233a4aca9851f594036fd904bae57b21bf 127.0.0.1:7001@17001 myself,master - 0 1555674615000 1 connected 0-5460
```

每个Redis的节点都有一个ID值，此ID将被此特定redis实例永久使用，以便实例在集群上下文中具有唯一的名称。每个节点都会记住使用此ID的每个其他节点，而不是通过IP或端口。IP地址和端口可能会发生变化，但唯一的节点标识符在节点的整个生命周期内都不会改变。我们简单地称这个标识符为节点ID。

测试数据


a9dc6e233a4aca9851f594036fd904bae57b21bf 127.0.0.1:7001@17001 myself,master -
d 0-5460
127.0.0.1:7001> set name guoweixin
-> Redirected to slot [5798] located at 127.0.0.1:7002
OK
127.0.0.1:7002> keys *
1) "name"
127.0.0.1:7002> quit
[root@localhost src]# redis-cli -h 127.0.0.1 -c -p 7005
127.0.0.1:7005> keys *
1) "name"
127.0.0.1:7005> get name
-> Redirected to slot [5798] located at 127.0.0.1:7002
"guoweixin"
127.0.0.1:7002>
在当前master进行赋值
重定向对7002[5798槽]节点 进行增加数据。
找到7002的 slave节点，通过 id发现是7005是从
登录7005从节点查看数据是否存在