Docker Compose介绍

使用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。本节课将讨论如何使用 Docker Compose来轻松、高效地管理容器。为了简单起见将 Docker Compose简称为 Compose。

Compose 是一个用于定义和运行多容器的Docker应用的工具。使用Compose，你可以在一个配置文件（yaml格式）中配置你应用的服务，然后使用一个命令，即可创建并启动配置中引用的所有服务。下面我们进入Compose的实战吧

Docker Compose的安装

Compose的安装有多种方式，例如通过shell安装、通过pip安装、以及将compose作为容器安装等等。本文讲解通过shell安装的方式。其他安装方式如有兴趣，可以查看Docker的官方文档：

https://docs.docker.com/compose/install/

```
# docker compose安装步骤
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
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

Docker Compose入门示例

Compose的使用非常简单，只需要编写一个`docker-compose.yml`，然后使用docker-compose 命令操作即可。`docker-compose.yml`描述了容器的配置，而docker-compose 命令描述了对容器的操作。我们首先通过一个示例快速入门：

还记得上节课，我们使用Dockerfile为项目microservice-eureka-server构建Docker镜像吗？我们还以此项目为例测试

- 我们在`microservice-eureka-server-0.0.1-SNAPSHOT.jar`所在目录的上一级目录，创建`docker-compose.yml` 文件。

目录树结构如下：

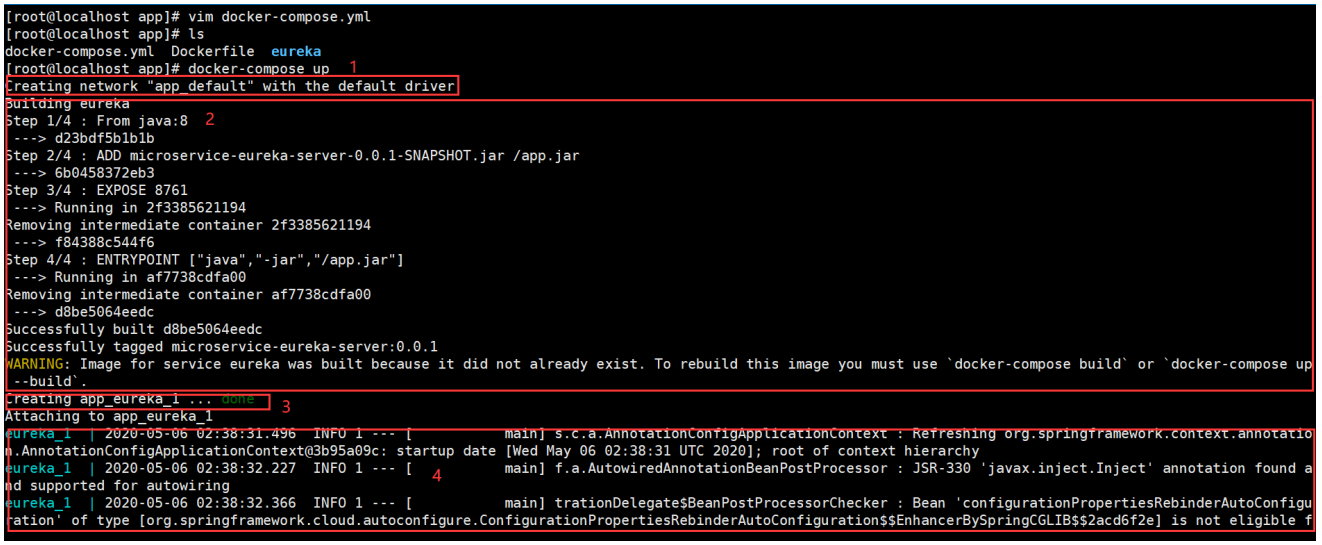
```
├── docker-compose.yml
└── eureka
    ├── Dockerfile
    └── microservice-eureka-server-0.0.1-SNAPSHOT.jar
```

- 然后在`docker-compose.yml` 中添加内容如下：

```
version: '3'
services:
 eureka: #指定服务名
 image: microservice-eureka-server:0.0.1 #指定镜像名称
 build: ./eureka #指定Dockfile所在路径
 ports:
 - "8761:8761" #指定端口映射
 expose:
 - 8761 #声明容器对外暴露的端口
```

- 在`docker-compose.yml` 所在路径执行：

```
docker-compose up （后面加-d可以后台启动）
```



如上图，compose启动会做几件事：

1、创建一个默认的网络app_default，默认以compose所在文件目录名加"_default"命名，compose内的所有容器都会加入此网络，可以相互用服务名访问。

2、如果镜像 microservice-eureka-server:0.0.1 不存在先构建镜像，如果镜像存在则不构建，加上 --build 参数可以强制先构建镜像，如果镜像之前构建过且构建文件没有变化或构建的内容没有变化，就算加上 --build 参数也不会重新构建。

3、根据构建的镜像创建一个名称叫 app_eureka_1 的容器。

4、启动容器。

- 访问：`http://宿主机IP:8761/`，发现可以正常访问eureka主页。

Docker Compose管理容器的结构

Docker Compose将所管理的容器分为三层，**分别是工程（project），服务（service）以及容器（container）。**

Docker Compose运行目录下的所有文件（docker-compose.yml、 extends文件或环境变量文件等）组成一个工程（默认为 docker-compose.yml所在目录的目录名称）。一个工程可包含多个服务，每个服务中定义了容器运行的镜像、参数和依赖，一个服务可包括多个容器实例。

上节示例里工程名称是 docker-compose.yml 所在的目录名。该工程包含了1个服务，服务名称是 eureka，执行 docker-compose up时，启动了eureka服务的1个容器实例。

同一个docker compose内部的容器之间可以用服务名相互访问，**服务名就相当于hostname，可以直接 ping 服务名，得到的就是服务对应容器的ip，如果服务做了扩容，一个服务对应了多个容器，则 ping 服务名 会轮询访问服务对应的每台容器ip，docker底层用了LVS等技术帮我们实现这个负载均衡。**

docker-compose.yml常用指令

image

指定镜像名称或者镜像id，如果该镜像在本地不存在，Compose会尝试pull下来。

示例：

```
image: java
```

build

指定Dockerfile文件的路径。可以是一个路径，例如：

```
build: ./dir
```

也可以是一个对象，用以指定Dockerfile和参数，例如：

```
build:
  context: ./dir
```

```
  dockerfile: Dockerfile-alternate
  args:
    buildno: 1
```

command

覆盖容器启动后默认执行的命令。

示例：

```
command: bundle exec thin -p 3000
```

也可以是一个list，类似于Dockerfile总的CMD指令，格式如下：

```
command: [bundle, exec, thin, -p, 3000]
```

links

显示链接到其他服务中的容器。可以指定服务名称和链接的别名使用`SERVICE:ALIAS`的形式，或者只指定服务名称，示例：

```
web:
  links:
    - db
    - db:database
    - redis
```

external_links

表示链接到docker-compose.yml外部的容器，甚至并非Compose管理的容器，特别是对于那些提供共享容器或共同服务。格式跟links类似，示例：

```
external_links:
 - redis_1
 - project_db_1:mysql
 - project_db_1:postgresql
```

ports

暴露端口信息。使用宿主端口:容器端口的格式，或者仅仅指定容器的端口（此时宿主机将会随机指定端口），类似于`docker run -p`，示例：

```
ports:
 - "3000"
 - "3000-3005"
 - "8000:8000"
 - "9090-9091:8080-8081"
 - "49100:22"
 - "127.0.0.1:8001:8001"
 - "127.0.0.1:5000-5010:5000-5010"
```

expose

暴露端口，只将端口暴露给连接的服务，而不暴露给宿主机，示例：

```
expose:
 - "3000"
 - "8000"
```

volumes

卷挂载路径设置。可以设置宿主机路径 （`HOST:CONTAINER`） 或加上访问模式 （`HOST:CONTAINER:ro`）。示例：

```
volumes:
  # Just specify a path and let the Engine create a volume
  - /var/lib/mysql

  # Specify an absolute path mapping
  - /opt/data:/var/lib/mysql
```

```
  # Path on the host, relative to the Compose file
  - ./cache:/tmp/cache

  # User-relative path
  - ~/configs:/etc/configs/:ro

  # Named volume
  - datavolume:/var/lib/mysql
```

volumes_from

从另一个服务或者容器挂载卷。可以指定只读或者可读写，如果访问模式没有指定，则默认是可读写。示例：

```
volumes_from:
 - service_name
 - service_name:ro
 - container:container_name
 - container:container_name:rw
```

environment

设置环境变量。可以使用数组或者字典两种方式。只有一个key的环境变量可以在运行Compose的机器上找到对应的值，这有助于加密的或者特殊主机的值。示例：

```
environment:
  RACK_ENV: development
  SHOW: 'true'
  SESSION_SECRET:

environment:
  - RACK_ENV=development
  - SHOW=true
  - SESSION_SECRET
```

env_file

从文件中获取环境变量，可以为单独的文件路径或列表。如果通过 `docker-compose -f FILE` 指定了模板文件，则 `env_file` 中路径会基于模板文件路径。如果有变量名称与 `environment` 指令冲突，则以`envirment` 为准。示例：

```
env_file: .env

env_file:
  - ./common.env
  - ./apps/web.env
  - /opt/secrets.env
```

extends

继承另一个服务，基于已有的服务进行扩展。

net

设置网络模式。示例：

```
net: "bridge"
net: "host"
net: "none"
net: "container:[service name or container name/id]"
```

dns

配置dns服务器。可以是一个值，也可以是一个列表。示例：

```
dns: 8.8.8.8
dns:
  - 8.8.8.8
  - 9.9.9.9
```

dns_search

配置DNS的搜索域，可以是一个值，也可以是一个列表，示例：

```
dns_search: example.com
dns_search:
  - dc1.example.com
  - dc2.example.com
```

其他

docker-compose.yml 还有很多其他命令，这里仅挑选常用命令进行讲解，其它不作赘述。如果感兴趣的，可以参考docker-compose.yml文件官方文档：https://docs.docker.com/compose/compose-file/

用Docker Compose编排Spring Cloud电商项目微服务

如果微服务较多，则可以用docker compose来统一编排，接下来我们用docker compose来统一编排电商项目的五个微服务：tulingmall-authcenter，tulingmall-gateway，tulingmall-member，tulingmall-order，tulingmall-product

编排电商项目依赖环境

1、创建一个空目录docker-mall

2、在docker-mall目录下新建一个编排文件docker-compose-env.yml，内容如下：

```yaml
version: '3'
services:
 mysql:
 image: mysql:5.7
 container_name: mysql
 command: mysqld --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci #覆盖容器启动后默认执行的启动mysql命令
 restart: always #关机或者重启机器时，docker同时重启容器，一般mysql服务可以这么设置，保持服务一直都在
 environment:
 MYSQL_ROOT_PASSWORD: root #设置root帐号密码
 ports:
 - 3306:3306
 volumes:
 - /mydata/mysql/data/db:/var/lib/mysql #数据文件挂载
 - /mydata/mysql/data/conf:/etc/mysql/conf.d #配置文件挂载
 - /mydata/mysql/log:/var/log/mysql #日志文件挂载
 redis:
 image: redis:5.0
 container_name: redis
 command: redis-server --appendonly yes
 volumes:
 - /mydata/redis/data:/data #数据文件挂载
 ports:
 - 6379:6379
 nginx:
 image: nginx:1.10
 container_name: nginx
```

```
  volumes:
  - /mydata/nginx/nginx.conf:/etc/nginx/nginx.conf #配置文件挂载，docker对单个文件的挂载需要先在宿主机建好对应文件才能挂载成功，可以先启动nginx容器，将容器里的nginx.conf文件复制出来修改好再挂载
  - /mydata/nginx/html:/usr/share/nginx/html #静态资源根目录挂载
  - /mydata/nginx/log:/var/log/nginx #日志文件挂载
  ports:
  - 80:80
  rabbitmq:
  image: rabbitmq:3.7.25-management
  container_name: rabbitmq
  volumes:
  - /mydata/rabbitmq/data:/var/lib/rabbitmq #数据文件挂载
  - /mydata/rabbitmq/log:/var/log/rabbitmq #日志文件挂载
  ports:
  - 5672:5672
  - 15672:15672
  elasticsearch:
  image: elasticsearch:6.4.0
  container_name: elasticsearch
  environment:
  - "cluster.name=elasticsearch" #设置集群名称为elasticsearch
  - "discovery.type=single-node" #以单一节点模式启动
  - "ES_JAVA_OPTS=-Xms512m -Xmx512m" #设置使用jvm内存大小，稍微配置大点，不然有可能启动不成功
  volumes:
  - /mydata/elasticsearch/plugins:/usr/share/elasticsearch/plugins #插件文件挂载
  - /mydata/elasticsearch/data:/usr/share/elasticsearch/data #数据文件挂载
  ports:
  - 9200:9200
  - 9300:9300
  kibana:
  image: kibana:6.4.0
  container_name: kibana
  links: #同一个compose文件管理的服务可以直接用服务名访问，如果要给服务取别名则可以用links实现，如下面的es就是elasticsearch服务的别名
  - elasticsearch:es #可以用es这个域名访问elasticsearch服务
  depends_on:
  - elasticsearch #kibana在elasticsearch启动之后再启动
  environment:
  - "elasticsearch.hosts=http://es:9200" #设置访问elasticsearch的地址
  ports:
  - 5601:5601
  logstash:
  image: logstash:6.4.0
  container_name: logstash
  volumes:
  - /mydata/logstash/logstash-springboot.conf:/usr/share/logstash/pipeline/logstash.conf #挂载logstash的配置文件，docker对单个文件的挂载需要先在宿主机建好对应文件才能挂载成功
  depends_on:
  - elasticsearch #kibana在elasticsearch启动之后再启动
  links:
  - elasticsearch:es #可以用es这个域名访问elasticsearch服务
  ports:
  - 4560:4560
```

```
  mongo:
  image: mongo:3.2
  container_name: mongo
  volumes:
  - /mydata/mongo/db:/data/db #数据文件挂载
  ports:
  - 27017:27017
  nacos:
  image: nacos/nacos-server:1.1.4
  container_name: nacos
  environment:
  - MODE=standalone
  volumes:
  - /mydata/nacos/logs/:/home/nacos/logs
  ports:
  - "8848:8848"
  zookeeper:
  image: zookeeper:3.5
  ports:
  - 2181:2181
  volumes:
  - /mydata/zookeeper/data:/data
  - /mydata/zookeeper/conf:/conf
```

3、启动compose所有容器，在docker-mall目录执行如下命令：

```
docker-compose -f docker-compose-env.yml up -d
```

常用的一些docker-compose命令：

```
# 查看compose内的容器
docker-compose -f docker-compose-env.yml ps
# 关闭或启动或重启compose内的某个容器
docker-compose -f docker-compose-env.yml stop/start/restart <服务名>
# 关闭或重启compose所有容器
docker-compose -f docker-compose-env.yml stop/restart
# 查看compose所有容器的运行日志
docker-compose -f docker-compose-app.yml logs -f
# 查看compose下某个容器的运行日志
 docker-compose -f docker-compose-app.yml logs -f <服务名>
 # 也可以把compose的容器日志输出到日志文件里去，然后用tail -f 随时查看
 docker-compose -f docker-compose-app.yml logs -f >> myDockerCompose.log &
 # 重新构建有变化的镜像并更新到容器再启动
 docker-compose up --build -d
 # 重新创建docker-compose.yml配置有变化的容器并启动
 docker-compose up --force-recreate -d
```

编排电商微服务

1、在docker-mall目录下分别创建tulingmall-authcenter，tulingmall-gateway，tulingmall-member，tulingmall-order，tulingmall-product目录。

2、修改电商项目上面这几个微服务配置文件里的环境配置为上面docker compose里的服务名，并打好jar包放入上面对应的文件夹。

以tulingmall-order服务为例，对应修改后的配置文件如下(注意：大家按照自己下载项目的配置文件去修改，不要直接用我这里的配置，有可能版本不对)

```
server:
```

```yaml
  port: 8844

spring:
  application:
  name: tulingmall-order
  cloud:
  nacos:
  discovery:
   server-addr: nacos:8848 #修改nacos的ip为compose的服务名
   datasource:
  url: jdbc:mysql://db:3306/micromall?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8 #修改mysql的ip为compose的服务名
  username: root
  password: root
  druid:
  initial-size: 5 #连接池初始化大小
  min-idle: 10 #最小空闲连接数
  max-active: 20 #最大连接数
  web-stat-filter:
  exclusions: "*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*" #不统计这些请求数据
  stat-view-servlet: #访问监控网页的登录用户名和密码
  login-username: druid
  login-password: druid


  redis:
  host: redis # Redis服务器地址 #修改redis的ip为compose的服务名
  database: 0 # Redis数据库索引（默认为0）
  port: 6379 # Redis服务器连接端口
  password:
  jedis:
  pool:
  max-active: 8 # 连接池最大连接数（使用负值表示没有限制）
  max-wait: -1ms # 连接池最大阻塞等待时间（使用负值表示没有限制）
  max-idle: 8 # 连接池中的最大空闲连接
  min-idle: 0 # 连接池中的最小空闲连接
  timeout: 3000ms # 连接超时时间（毫秒）

  rabbitmq: #注意：rabbitmq启动后记得用guest账号登录进去添加admin用户并设置为admin管理员和/mall这个virtual-host以及在/mall下创建一个队列mall.order.cancel，并将/mall分配给admin用户
  host: rabbitmq #修改rabbitmq的ip为compose的服务名
  port: 5672
  virtual-host: /mall
  username: admin
  password: admin
  publisher-confirms: true #如果对异步消息需要回调必须设置为true
  main:
  allow-bean-definition-overriding: true

feign:
  client:
  config:
  default:
```

```
  loggerLevel: full
  requestInterceptors:
  - com.tuling.tulingmall.feignapi.interceptor.HeaderInterceptor
  readTimeout: 3000
  connectTimeout: 3000

logging:
  level:
  com:
  tuling:
  tulingmall:
  feignapi:
  ums:
  UmsMemberReceiveAddressFeignApi: debug

# 自定义redis键值
redis:
  key:
  prefix:
  authCode: "portal:authCode:"
  orderId: "portal:orderId:"
  expire:
  authCode: 90 # 验证码超期时间

# 自定义消息队列名称
rabbitmq:
  queue:
  name:
  cancelOrder: cancelOrderQueue

#支付-当面付qrcode存储与访问路径设置
trade:
  zhifu:
  qrcode:
  aliPayPath: /alipay
  weChatPath: /wechat
  storePath: C:/temp/qr-code
  httpBasePath: /static/qrcode
  paySuccessCallBack: http://yangguo.natapp1.cc/order/paySuccess

seata:
  config:
  nacos:
  server-addr: nacos:8848 #修改nacos的ip为compose的服务名
  type: nacos
  registry:
  type: nacos
   tx-service-group: prex_tx_group
   client:
   support:
   spring:
   datasource-autoproxy: true

```

```yaml
#rocketmq配置
rocketmq:
 name-server: rocketmq:9876 #连接超时时间 #修改rocketmq的ip为compose的服务名
 producer:
 send-message-timeout: 30000 #发送消息超时时间
 group: order-group
 tulingmall:
 scheduleTopic: order-status-check #定时任务
 cancelGroup: cancel-order #消费组业务逻辑,取消超时未支付订单
 transGroup: cart-delete #事务消息群组
 transTopic: order-cart #订单-购物车主题
 asyncOrderTopic: async-order #异步订单topic
 asyncOrderGroup: async-order-group #异步下单消息消费

mybatis:
 mapper-locations:
 - classpath:dao/*.xml
 - classpath*:com/**/mapper/*.xml

management: #开启SpringBoot Admin的监控
 endpoints:
 promethus:
 enable: true
 web:
 exposure:
 include: '*'
 endpoint:
 health:
 show-details: always
```

3、在每个微服务目录下新建一个Dockerfile，内容如下，以tulingmall-order服务为例，其它微服务都类似修改：

```
# 基于哪个镜像
From java:8
# 复制文件到容器
ADD tulingmall-order-0.0.1-SNAPSHOT.jar /app.jar
# 配置容器启动后执行的命令
ENTRYPOINT ["java","-jar","/app.jar"]
```

4、在docker-mall目录下新建微服务编排文件docker-compose-app.yml，内容如下：

```yaml
version: '3'
services:
 tulingmall-authcenter:
 image: mall/tulingmall-authcenter:0.0.1 #指定镜像名称
 build: ./tulingmall-authcenter #指定Dockfile所在路径
 container_name: tulingmall-authcenter #指定启动容器名称
 ports:
 - 9999:9999
 volumes:
 - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
 external_links: #访问不在同一个compose文件管理的服务需要用external_links，前提是这些服务都在同一个网络下才能正常访问
 - nacos:nacos #可以用nacos这个域名访问nacos服务
 - mysql:db #可以用db这个域名访问mysql服务
 tulingmall-gateway:
```

```
image: mall/tulingmall-gateway:0.0.1
build: ./tulingmall-gateway
container_name: tulingmall-gateway
ports:
- 8888:8888
volumes:
- /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
depends_on:
- tulingmall-authcenter #gateway在authcenter启动之后再启动
external_links:
- nacos:nacos
tulingmall-member:
image: mall/tulingmall-member:0.0.1
build: ./tulingmall-member
container_name: tulingmall-member
ports:
- 8877:8877
volumes:
- /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
external_links:
- nacos:nacos
- mysql:db #可以用db这个域名访问mysql服务
- mongo
- redis
- rabbitmq
tulingmall-product:
image: mall/tulingmall-product:0.0.1
build: ./tulingmall-product
container_name: tulingmall-product
ports:
- 8866:8866
volumes:
- /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
external_links:
- nacos:nacos
- mysql:db #可以用db这个域名访问mysql服务
- redis
- zookeeper
tulingmall-order:
image: mall/tulingmall-order:0.0.1
build: ./tulingmall-order
container_name: tulingmall-order
ports:
- 8844:8844
volumes:
- /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
external_links:
- nacos:nacos
- mysql:db #可以用db这个域名访问mysql服务
- redis
- rabbitmq
- rockermq
```

5、启动compose的所有微服务容器，在docker-mall目录执行如下命令：

```
#这里启动的微服务跟上面启动的mysql，redis这些中间件服务因为都在docker-mall目录下，即都是同一个工程下，默认都在相同的网络下，可以相互访问
docker-compose -f docker-compose-app.yml up -d
```

6、访问下微服务的api看是否都正常，访问接口参数参看视频，不一定访问我列的这几个接口，其它的接口也行

```
1、通过网关访问登录接口获取token，post方式：
http://192.168.50.60:8888/sso/login?username=test&password=test
2、通过网关访问添加购物车接口，post方式：
http://192.168.50.60:8888/cart/add?memberId=1&nickName=windir
3、通过网关访问查询购物车接口，get方式：
http://192.168.50.60:8888/cart/list?memberId=1
4、通过网关访问创建订单接口，post方式：
http://192.168.50.60:8888/order/generateOrder?memberId=1
```

动态扩容微服务(单物理机内扩容)

有时我们需要扩容微服务，比如我们想把用户和订单微服务各部署两个微服务，则需要将docker-compose.yml里的服务的端口映射和容器名称都注释掉，因为不可能两个订单服务的容器映射到宿主机的同一个端口，修改之后的docker-compose-app.yml内容如下：

```
version: '3'
services:
 tulingmall-authcenter:
 image: mall/tulingmall-authcenter:0.0.1 #指定镜像名称
 build: ./tulingmall-authcenter #指定Dockfile所在路径
 container_name: tulingmall-authcenter #指定启动容器名称
 ports:
 - 9999:9999
 volumes:
 - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
 external_links: #访问不在同一个compose文件管理的服务需要用external_links，前提是这些服务都在同一个网络下才能正常访问
 - nacos:nacos #可以用nacos这个域名访问nacos服务
 - mysql:db #可以用db这个域名访问mysql服务
 tulingmall-gateway:
 image: mall/tulingmall-gateway:0.0.1
 build: ./tulingmall-gateway
 container_name: tulingmall-gateway
 ports:
 - 8888:8888
 volumes:
 - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
 depends_on:
 - tulingmall-authcenter #gateway在authcenter启动之后再启动
 external_links:
 - nacos:nacos
 tulingmall-member:
 image: mall/tulingmall-member:0.0.1
 build: ./tulingmall-member
 container_name: tulingmall-member
 ports:
 - 8877:8877
 volumes:
```

```
  - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
  external_links:
  - nacos:nacos
  - mysql:db #可以用db这个域名访问mysql服务
  - mongo
  - redis
  - rabbitmq
  tulingmall-product:
  image: mall/tulingmall-product:0.0.1
  build: ./tulingmall-product
# container_name: tulingmall-product
# ports:
# - 8866:8866
  volumes:
  - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
  external_links:
  - nacos:nacos
  - mysql:db #可以用db这个域名访问mysql服务
  - redis
  - zookeeper
  tulingmall-order:
  image: mall/tulingmall-order:0.0.1
  build: ./tulingmall-order
# container_name: tulingmall-order
# ports:
# - 8844:8844
  volumes:
  - /etc/localtime:/etc/localtime:ro #同步宿主机与容器时间
  external_links:
  - nacos:nacos
  - mysql:db #可以用db这个域名访问mysql服务
  - redis
  - rabbitmq
  - rockermq
```

执行如下扩容命令，**服务一旦扩容对应了多个容器，则访问服务名docker会自动帮我们负载均衡去访问服务对应的每台容器**：

```
docker-compose -f docker-compose-app.yml up -d #必须先正常编排微服务，然后才能动态扩容
docker-compose -f docker-compose-app.yml scale tulingmall-order=2 tulingmall-member=2
#如果要缩容执行如下操作
docker-compose -f docker-compose-app.yml scale tulingmall-order=1 tulingmall-member=1
```

注意：docker compose主要用在单物理机内扩容的情况，要做多机扩容还需自己在多个机器上做很多定制化配置，当然，要做多物理机扩容一般都会用docker swarm或kubernetes。

文档：02-用Docker Compose编排电商微服务项?..
链接：http://note.youdao.com/noteshare?id=b808b0736cf90cde53a6be1c6813f047&sub=8FABAB9EA4B34D98B15E583164210C73