**作者：诸葛老师**

**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pip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/>

1、安装python-pip

# yum -y install epel-release

# yum -y install python-pip

2、安装docker-compose

# pip install docker-compose

3、待安装完成后，执行查询版本的命令

# 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：

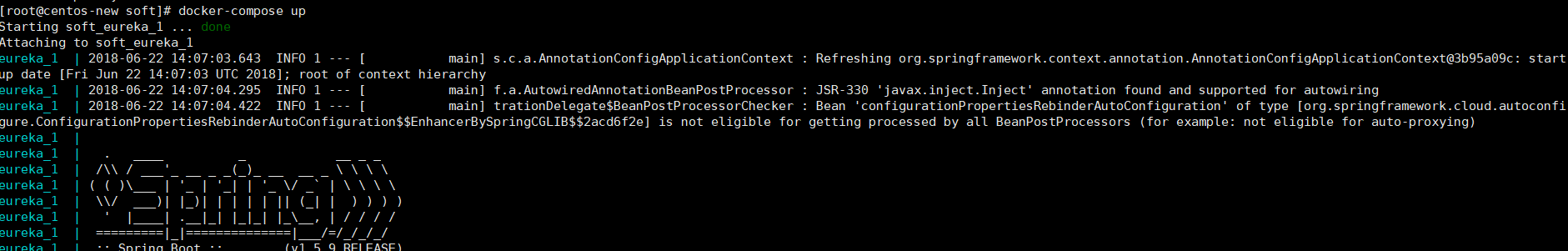
├── docker-compose.yml
└── eureka
├── Dockerfile
└── microservice-eureka-server-0.0.1-SNAPSHOT.jar

然后在docker-compose.yml 中添加内容如下：

eureka: #指定服务名
build: ./eureka #指定Dockfile所在路劲
ports:
- "8761:8761" #指定端口映射
expose:
- 8761 #容器提供服务端口

在docker-compose.yml 所在路径执行：

# docker-compose up



Compose就会自动构建镜像并使用镜像启动容器。也可使用 docker-compose up -d后台启动并运行这些容器

访问：http://宿主机IP:8761/ ，发现可以正常启动。

**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

**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**

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，本文仅挑选常用命令进行讲解，其他不不作赘述。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微服务**

如果微服务较多，则可以用docker compose来统一编排，我们打算用docker compose来统一编排三个微服务：eureka服务(项目05-ms-eureka-server)，user服务(项目05-ms-provider-user)，order服务(项目05-ms-consumer-order-ribbon)

**编排微服务**

1、在根目录创建文件夹/app

2、在app目录下新建docker-compose.yml文件和三个文件夹eureka，user，order

3、在eureka，user，order三个文件夹下分别构建eureka服务镜像，user服务镜像，order服务镜像，以构建eureka服务镜像为例，在eureka文件夹下新建dockerfile文件并且将eureka服务的可运行jar包上传到该目录(注意：需要将配置eureka.client.serviceUrl.defaultZone的值改为<http://eureka:8761/eureka/>，默认情况下Compose以服务名称作为hostname被其他容器访问)，dockerfile文件内容如下

# 基于哪个镜像

From java:8

# 将本地文件夹挂载到当前容器

VOLUME /tmp

# 复制文件到容器

ADD microservice-eureka-server-0.0.1-SNAPSHOT.jar /app.jar

# 声明需要暴露的端口

EXPOSE 8761

# 配置容器启动后执行的命令

ENTRYPOINT ["java","-jar","/app.jar"]

4、docker-compose.yml内容如下

version: '2' #docker的文件格式版本

services:

eureka: #docker服务名

image: eureka #docker镜像

ports:

- "8761:8761"

user:

image: user

ports:

- "8000:8000"

order:

image: order

ports:

- "8010:8010"

5、启动所有微服务，在命令后面加-d可以后台启动：

# docker-compose up

6、访问三个微服务是否正常

**编排高可用微服务**

1、在根目录创建文件夹/app-ha

2、在app-ha目录下新建docker-compose.yml文件和三个文件夹eureka-ha，user-ha，order-ha

3、在eureka-ha，user-ha，order-ha三个文件夹下分别构建eureka-ha服务镜像，user-ha服务镜像，order-ha服务镜像，eureka-ha服务参考项目08-ms-eureka-server-ha，（注意：需要修改user服务和order服务配置文件eureka.client.serviceUrl.defaultZone的值为<http://peer1:8761/eureka/,http://peer2:8762/eureka/>)

4、docker-compose.yml内容如下

version: '2' #docker的文件格式版本

services:

peer1: #docker微服务名称

image: eureka-ha #docker镜像

ports:

- "8761:8761"

environment:

- spring.profiles.active=peer1

peer2:

image: eureka-ha

ports:

- "8762:8762"

environment:

- spring.profiles.active=peer2

user:

image: user-ha

ports:

- "8000:8000"

order:

image: order-ha

ports:

- "8010:8010"

5、启动所有微服务，在命令后面加-d可以后台启动：

# docker-compose up

6、访问三个微服务是否正常

**动态扩容微服务**

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docker-compose.yml文件应该如下配置

docker-compose.yml内容如下

version: '2' #docker的文件格式版本

services:

peer1: #docker微服务名称

image: eureka-ha #docker镜像

ports:

- "8761:8761"

environment:

- spring.profiles.active=peer1

peer2:

image: eureka-ha

ports:

- "8762:8762"

environment:

- spring.profiles.active=peer2

user:

image: user-ha

order:

image: order-ha

执行如下扩容命令：

**# docker-compose up #必须先正常编排微服务，然后才能动态扩容**

**# docker-compose scale user=2 order=2**

注意：如果是在同一台物理机上做动态扩容，则需要在docker-compose.yml里去掉除了eureka其它微服务ports端口映射

运行完查看eureka注册中心如下图所示：

