Docker Compose

编排（Orchestration）描述了自动配置、协作和管理服务的过程。在Docker中，Orchestration用来描述一组实践过程，管理运行在多个Docker容器中的应用，而这些Docker容器有可能运行在多个宿主机上。Docker Compose是一个简单的开源Docker编配工具，其使用YAML文件定义一组要启动的容器，以及容器运行时的属性。

# 安装

使用curl命令来完成安装，命令如下：

*sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose*

修改docker-compose执行权限：

*chmod +x /usr/local/bin/docker-compose*

安装测试：

*docker-compose --version*

安装成功后，测试结果如下：

*docker-compose version 1.18.0, build 8dd22a9*

# 简单使用

创建一个Python应用，使用Flask，将数值记入Redis

1. 初始化

创建composetest目录

*$ mkdir composetest*

*$ cd composetest*

创建app.py，内容

*import time*

*import redis*

*from flask import Flask*

*app = Flask(\_\_name\_\_)*

*cache = redis.Redis(host='redis', port=6379)*

*def get\_hit\_count():*

*retries = 5*

*while True:*

*try:*

*return cache.incr('hits')*

*except redis.exceptions.ConnectionError as exc:*

*if retries == 0:*

*raise exc*

*retries -= 1*

*time.sleep(0.5)*

*@app.route('/')*

*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)*

创建文件requirements.txt，内容如下：

*flask*

*redis*

配置下面要安装的软件，分别是flask和redis

1. 创建Dockerfile

在composetest目录中创建Dockerfile，内容如下：

*FROM python:3.4-alpine*

*ADD . /code*

*WORKDIR /code*

*RUN pip install -r requirements.txt*

*CMD ["python", "app.py"]*

该镜像，将当前的目录复制到镜像/code中，然后安装python依赖，最后是镜像启动是启动的命令: *python app.py*

1. 在Compose file中定义服务

创建文件docker-compose.yml，内容如下：

*version: '3'*

*services:*

*web:*

*build: .*

*ports:*

*- "5000:5000"*

*redis:*

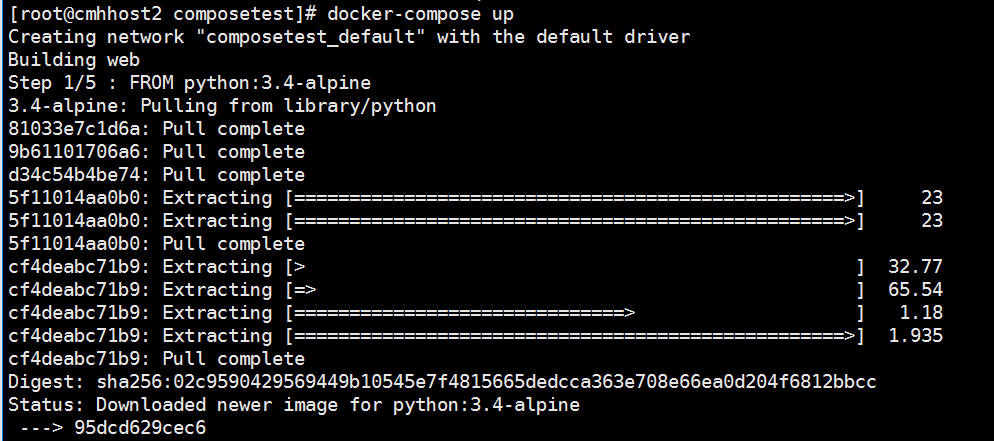
*image: "redis:alpine"*

定义了两个文件，web和redis，使用Dockerfile中的镜像启动两个服务，web端口为5000。Redis使用redis 镜像（从Docker Hub中获取）。

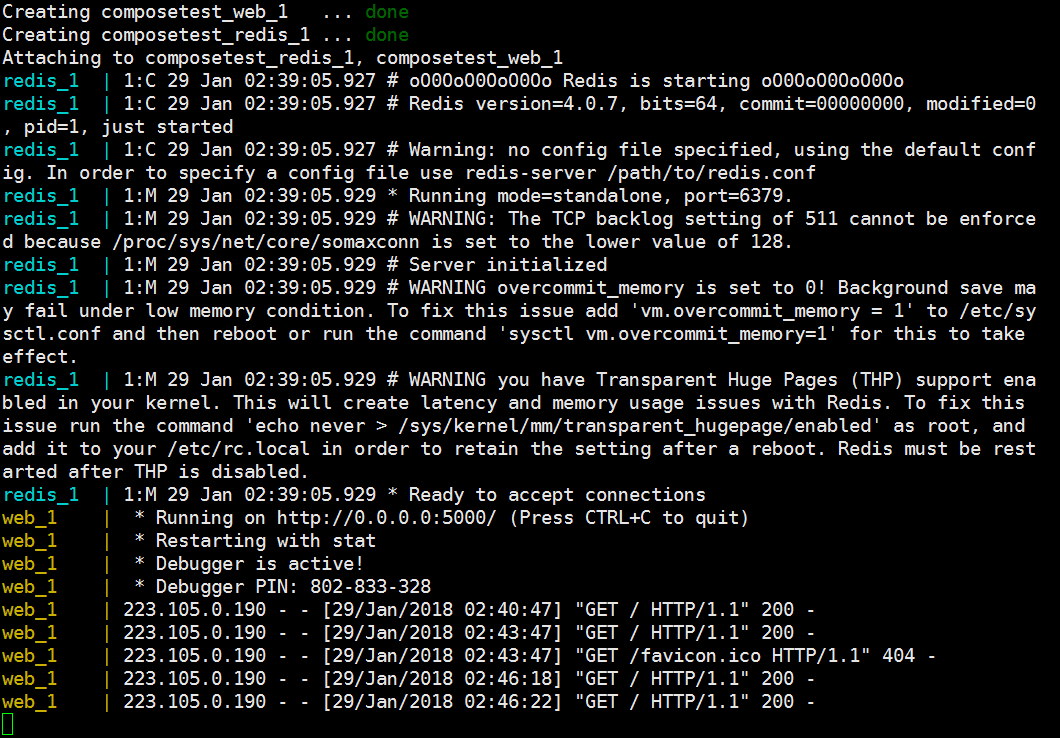
1. 使用Compose创建app，命令如下：

# docker-compose up

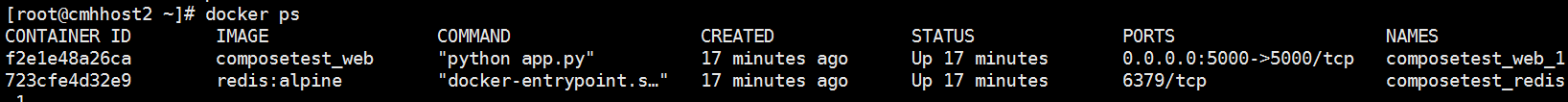
执行截图如下：



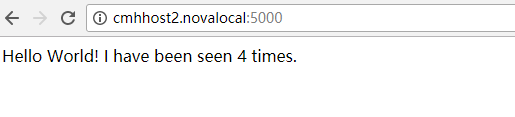
其中的web执行如下：



查看运行的container，如下所示：



打开页面：



至此使用Docker-compose构建了应用。

# Docker Compose配置文件详解

标准的配置文件包含：version、services和networks三部分

|—version，协议的版本号

|—services，定义服务

|— <name>，二级标签，用户自己定义，服务名称

|— image，服务镜像名称或镜像id，如果在本地不存在，则尝试拉取这个镜像

|— build，定义DockerFile，在使用up启动时自动构建镜像build，下面是定义参数

|— context: 路径

|— args，类似于与DockerFile的ARG指令

|— command，容器启动的执行命令，覆盖容器中默认的启动命令

|— container\_name， 定义容器名称

|— dependes\_on，在一个项目中容器的启动顺序是要有要求的，使用depends来定义容器的依赖、启动先后的问题

|— dns，和—dns参数一样的用途，配置参数可以是一个列表

|— tmpfs，挂载临时目录到容器内部，和run参数一样的效果

|— entrypoint，对应ENTRYPOINT指令，用户指定接入点

|— env\_file，使用的配置文件路径

|— environment，与arg类型，设置镜像变量，与ENV作用相同

|— expose，用户指定暴露的端口

|— external\_links，让compose项目中的容器连接到项目外的容器

|— extra\_hosts，添加主机名的标签，在/etc/hosts文件中添加一些记录与—add-host类似

|— labels，向容器添加元数据，与DockerFile中的LABLE指令一个意思

|— logging，配置日志服务

|— pid，设置主机的PID模式

|— ports，映射端口的标签

|— security\_opt，为每个容器覆盖默认的标签

|— stop\_signal，设置另一个信号来停止容器

|— volumes，挂载一个目录或者一个已存在的数据卷容器

|— volumens\_froms，从其他容器或者服务挂载数据卷

|— cap\_add, cap\_drop，添加或者删除容器的内核功能

|— cgroup\_parent，指定一个容器的父级别cgroup

|— devices，设备的映射列表

|— extends，扩展另一个服务，扩展内容来自当前文件，也可以来自其他文件

|—network\_mode，与—net参数类似

|— networks，加入指定网络

|—其他，cpu\_shares, cpu\_quota, cpuset, domainname, hostname, ipc, mac\_address, mem\_limit, memswap\_limit, privileged, read\_only, restart, shm\_size, stdin\_open, tty, user, working\_dir，单值的标签，类似docker run的效果

|—networks，配置自己的网络

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

https://github.com/docker/compose/releases