Docker之Compose服务编排

Compose是Docker的服务编排工具，主要用来构建基于Docker的复杂应用，Compose 通过一个配置文件来管理多个Docker容器，非常适合组合使用多个容器进行开发的场景。

说明：**Compose是Fig的升级版，Fig已经不再维护。Compose向下兼容Fig，所有`fig.yml`只需要更名为`docker-compose.yml`即可被Compose使用。**

服务编排工具使得Docker应用管理更为方便快捷。 Compose网站：https://docs.docker.com/compose/

安装Compose:

```
# 方法一：
curl -L https://github.com/docker/compose/releases/download/1.8.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose

# Linux下等效于
$ curl -L https://github.com/docker/compose/releases/download/1.8.1/docker-compose-Linux-x86_64 > /usr/local/bin/docker-compose; chmod +x /usr/local/bin/docker-compose

# 方法二：使用pip安装，版本可能比较旧
$ yum install python-pip python-dev
$ pip install docker-compose
```

```
11
12 # 方法三：作为容器安装
13 sudo curl -L https://github.com/docker/compose/releases/downloa
d/1.20.1/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/
docker-compose
14 chmod +x /usr/local/bin/docker-compose
15
16 # 方法四：离线安装
17 # 下载[docker-compose-Linux-x86_64](https://github.com/docker/co
mpose/releases/download/1.8.1/docker-compose-Linux-x86_64)，然后重
新命名添加可执行权限即可：
18 $ mv docker-compose-Linux-x86_64 /usr/local/bin/docker-compose;
19 $ chmod +x /usr/local/bin/docker-compose
```

安装完成后可以查看版本：

```
1 # docker-compose --version
2 docker-compose 1.8.1
```

升级

如果你使用的是 Compose 1.2或者早期版本，当你升级完成后，你需要删除或者迁移你现有的容器。这是因为，1.3版本， Composer 使用 Docker 标签来对容器进行检测，所以它们需要重新创建索引标记。

卸载

```
1 $ rm /usr/local/bin/docker-compose
2
3 # 卸载使用pip安装的compose
4 $ pip uninstall docker-compose
```

Compose区分Version 1和Version 2（Compose 1.6.0+，Docker Engine 1.10.0+）。Version 2支持更多的指令。Version 1没有声明版本默认是"version 1"。Version 1将来会被弃用。

版本1指的是忽略`version`关键字的版本；版本2必须在行首添加`version: '2'`。

入门示例

一般步骤

1、定义Dockerfile，方便迁移到任何地方；

2、编写docker-compose.yml文件；

3、运行`docker-compose up`启动服务

示例

准备工作：提前下载好镜像：

```
docker pull mysql
docker pull wordpress
```

需要新建一个空白目录，例如wptest。新建一个docker-compose.yml

```
version: '2'
services:
 web:
 image: wordpress:latest
 links:
 - db
 ports:
 - "8002:80"
 environment:
  WORDPRESS_DB_HOST: db:3306
  WORDPRESS_DB_PASSWORD: 123456
  db:
  image: mysql
  environment:
  - MYSQL_ROOT_PASSWORD=123456
 以上命令的意思是新建db和wordpress容器。等同于：
```

```
$ docker run --name db -e MYSQL_ROOT_PASSWORD=123456 -d mysql
$ docker run --name some-wordpress --link db:mysql -p 8002:80 -d
wordpress
```

注意，如果你是直接从fig迁移过来的，且`web`里`links`是`- db:mysql`，这里会提示没有给wordpress设置环境变量，这里需要添加环境变量`WORDPRESS_DB_HOST`和`WORDPRESS_DB_PASSWORD`。

好，我们启动应用：

```
# docker-compose up
Creating wptest_db_1...
Creating wptest_wordpress_1...
```

```
Attaching to wptest_db_1, wptest_wordpress_1
wordpress_1 | Complete! WordPress has been successfully copied to /var/www/html
```

就成功了。浏览器访问 http://localhost:8002（或 http://host-ip:8002）即可。

默认是前台运行并打印日志到控制台。如果想后台运行，可以：

```
docker-compose up -d
```

服务后台后，可以使用下列命令查看状态：

```
# docker-compose ps
 Name Command State Ports
------------------------------------------------------------------------
figtest_db_1 docker-entrypoint.sh mysqld Up 3306/tcp
figtest_wordpress_1 docker-entrypoint.sh apach ... Up 0.0.0.0:8002->80/tcp

# docker-compose logs
Attaching to wptest_wordpress_1, wptest_db_1
db_1 | 2016-10-4T14:38:46.98030Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).
db_1 | 2016-10-4T14:38:46.99974Z 0 [Note] mysqld (mysqld 5.7.15) starting as process 1 ...
db_1 | 2016-10-4T14:38:46.27191Z 0 [Note] InnoDB: PUNCH HOLE support available

```

停止服务：

```
# docker-compose stop
Stopping wptest_wordpress_1...
Stopping wptest_db_1...
```

重新启动服务：

```
docker-compose restart
```

docker-compose.yml参考

每个docker-compose.yml必须定义`image`或者`build`中的一个，其它的是可选的。

image

指定镜像tag或者ID。示例：

```
image: redis
image: ubuntu:14.04
image: tutum/influxdb
image: example-registry.com:4000/postgresql
image: a4bc65fd
```

注意，在`version 1`里同时使用`image`和`build`是不允许的，`version 2`则可以，如果同时指定了两者，会将`build`出来的镜像打上名为`image`标签。

build

用来指定一个包含`Dockerfile`文件的路径。一般是当前目录`.`。Fig将build并生成一个随机命名的镜像。

注意，在`version 1`里`bulid`仅支持值为字符串。`version 2`里支持对象格式。

```
build: ./dir

build:
 context: ./dir
 dockerfile: Dockerfile-alternate
 args:
 buildno: 1
```

`context`为路径，`dockerfile`为需要替换默认`docker-compose`的文件名，`args`为构建(build)过程中的环境变量，用于替换Dockerfile里定义的`ARG`参数，容器中不可用。示例：

```
Dockerfile:
ARG buildno
ARG password

RUN echo "Build number: $buildno"
RUN script-requiring-password.sh "$password"
docker-compose.yml:
```

```
build:
 context: .
 args:
 buildno: 1
 password: secret

build:
 context: .
 args:
 - buildno=1
 - password=secret
```

command

用来覆盖缺省命令。示例：

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

`command`也支持数组形式：

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

links

用于链接另一容器服务，如需要使用到另一容器的mysql服务。可以给出服务名和别名；也可以仅给出服务名，这样别名将和服务名相同。同`docker run --link`。示例：

```
links:
 - db
 - db:mysql
 - redis
```

使用了别名将自动会在容器的`/etc/hosts`文件里创建相应记录：

```
172.17.2.186 db
172.17.2.186 mysql
172.17.2.187 redis
```

所以我们在容器里就可以直接使用别名作为服务的主机名。

ports

用于暴露端口。同`docker run -p`。示例：

```
ports:
 - "3000"
 - "8000:8000"
```

```
  - "49100:22"
  - "127.0.0.1:8001:8001"
```

expose

expose提供container之间的端口访问，不会暴露给主机使用。同`docker run --expose`。

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

volumes

挂载数据卷。同`docker run -v`。示例：

```
volumes:
  - /var/lib/mysql
  - cache/:/tmp/cache
  - ~/configs:/etc/configs/:ro
```

volumes_from

挂载数据卷容器，挂载是容器。同`docker run --volumes-from`。示例：

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

`container:container_name`格式仅支持`version 2`。

environment

添加环境变量。同`docker run -e`。可以是数组或者字典格式：

```
environment:
 RACK_ENV: development
 SESSION_SECRET:

environment:
  - RACK_ENV=development
  - SESSION_SECRET
```

depends_on

用于指定服务依赖，一般是mysql、redis等。

指定了依赖，将会优先于服务创建并启动依赖。

links也可以指定依赖。

external_links

链接搭配docker-compose.yml文件或者Compose之外定义的服务，通常是提供共享或公共服务。格式与links相似：

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

注意，external_links链接的服务与当前服务必须是同一个网络环境。

extra_hosts

添加主机名映射。

```
extra_hosts:
 - "somehost:162.242.195.82"
 - "otherhost:50.31.209.229"
```

将会在/etc/hosts创建记录：

```
162.242.195.82 somehost
50.31.209.229 otherhost
```

extends

继承自当前yml文件或者其它文件中定义的服务，可以选择性的覆盖原有配置。

```
extends:
 file: common.yml
 service: webapp
```

service必须有，file可选。service是需要继承的服务，例如web、database。

net

设置网络模式。同docker的--net参数。

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

dns

自定义dns服务器。

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

```
4   - 9.9.9.9
```

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

这些命令都是单个值，含义请参考docker run。

```
cpu_shares: 73
cpu_quota: 50000
cpuset: 0,1

user: postgresql
working_dir: /code

domainname: foo.com
hostname: foo
ipc: host
mac_address: 02:42:ac:11:65:43

mem_limit: 1000000000
mem_limit: 128M
memswap_limit: 2000000000
privileged: true

restart: always

read_only: true
shm_size: 64M
stdin_open: true
tty: true
```

命令行参考

```
$ docker-compose
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
```

```
 docker-compose -h|--help

Options:
  -f, --file FILE Specify an alternate compose file (default: doc
ker-compose.yml)
  -p, --project-name NAME Specify an alternate project name (def
ault: directory name)
  --verbose Show more output
  -v, --version Print version and exit
  -H, --host HOST Daemon socket to connect to

  --tls Use TLS; implied by --tlsverify
  --tlscacert CA_PATH Trust certs signed only by this CA
  --tlscert CLIENT_CERT_PATH Path to TLS certificate file
  --tlskey TLS_KEY_PATH Path to TLS key file
  --tlsverify Use TLS and verify the remote
  --skip-hostname-check Don't check the daemon's hostname agains
t the name specified
  in the client certificate (for example if your docker host
  is an IP address)

Commands:
  build Build or rebuild services
  bundle Generate a Docker bundle from the Compose file
  config Validate and view the compose file
  create Create services
  down Stop and remove containers, networks, images, and volumes
  events Receive real time events from containers
  exec Execute a command in a running container
  help Get help on a command
  kill Kill containers
  logs View output from containers
  pause Pause services
  port Print the public port for a port binding
  ps List containers
  pull Pulls service images
```

```
push Push service images
restart Restart services
rm Remove stopped containers
run Run a one-off command
scale Set number of containers for a service
start Start services
stop Stop services
unpause Unpause services
up Create and start containers
version Show the Docker-Compose version information
```

批处理脚本

```
# 关闭所有正在运行容器
docker ps | awk '{print $1}' | xargs docker stop

# 删除所有容器应用
docker ps -a | awk '{print $1}' | xargs docker rm
# 或者
docker rm $(docker ps -a -q)
```

参考：

1、Overview of Docker Compose - Docker
https://docs.docker.com/compose/overview/
2、library/mysql - Docker Hub
https://hub.docker.com/_/mysql/
3、library/wordpress - Docker Hub
https://hub.docker.com/_/wordpress/