Docker 编排利器 - Compose简介

1.安装

```
#更换pip源
mkdir ~/.pip
echo "[global]
index-url = https://pypi.douban.com/simple/
" > ~/.pip/pip.conf
sudo pip3 install docker-compose
```

2.基本命令

YAML模板文件语法

默认的模板文件是`docker-compose.yml`,其中定义的每个服务都必须通过`image`指令指定镜像或`build`指令（需要Dockerfile）来自动构建。
其他大部分都跟`docker run`中类似。
如果使用build指令，在Dockerfile中设置的选项（例如：`CMD,EXPOSE,VOLUME,ENV`等）将自动被获取，无需在docker-compose.yml中再次被设置。
1、image

指定为镜像名称或镜像ID。如果镜像不存在，Compose将尝试从互联网拉取这个镜像，例如：

```
image: ubuntu
image: orchardup/postgresql
image: a4bc65fd
```

2、build

指定Dockerfile所在文件夹的路径。Compose将会利用他自动构建这个镜像，然后使用这个镜像。
`build: ./dir`
3、command

覆盖容器启动后默认执行的命令。
`command: bundle exec thin -p 3000`
4、links

链接到其他服务容器，使用服务名称(同时作为别名)或服务别名（SERVICE:ALIAS）都可以

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

注意：使用别名会自动在服务器中的`/etc/hosts`里创建，如：`172.17.2.186 db`，相应的环境变量也会被创建。

5、external_links

链接到docker-compose.yml外部的容器，甚至并非是Compose管理的容器。参数格式和links类似。

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

```
  - project_db_2:sqlserver
```

6、ports

暴露端口信息。
宿主机器端口：容器端口（HOST:CONTAINER）格式或者仅仅指定容器的端口（宿主机器将会随机分配端口）都可以。

```
ports:
 - "3306"
 - "8080:80"
 - "127.0.0.1:8090:8001"
```

注意：当使用 `HOST:CONTAINER` 格式来映射端口时，如果你使用的容器端口小于 60 你可能会得到错误得结果，因为 YAML 将会解析 `xx:yy` 这种数字格式为 60 进制。所以建议采用字符串格式。

7、expose

暴露端口，与posts不同的是expose只可以暴露端口而不能映射到主机，只供外部服务连接使用；仅可以指定内部端口为参数。

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

8、volumes

设置卷挂载的路径。可以设置宿主机路径:容器路径（ `host:container` ）或加上访问模式（ `host:container:ro` ） `ro` 就是 `readonly` 的意思，只读模式。

```
volumes:
 - /var/lib/mysql:/var/lib/mysql
 - /configs/mysql:/etc/configs/:ro
```

9、volunes_from

挂载另一个服务或容器的所有数据卷。

```
volumes_from:
 - service_name
 - container_name
```

10、environment

设置环境变量。可以属于数组或字典两种格式。
如果只给定变量的名称则会自动加载它在Compose主机上的值，可以用来防止泄露不必要的数据。

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

11、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，则以后者为准。

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

环境变量文件中每一行都必须有注释，支持#开头的注释行。

```
# common.env: Set Rails/Rack environment
RACK_ENV=development
```

12、extends

基于已有的服务进行服务扩展。例如我们已经有了一个webapp服务，模板文件为common.yml.

```
# common.yml
webapp:
build: ./webapp
environment:
\ - DEBUG=false
\ - SEND_EMAILS=false
```

编写一个新的 development.yml 文件，使用 common.yml 中的 webapp 服务进行扩展。
development.yml

```
web:
extends:
file: common.yml
service:
  webapp:
    ports:
      \ - "8080:80"
    links:
      \ - db
    envelopment:
      - DEBUG=true
   db:
    image: mysql:5.7
```

后者会自动继承common.yml中的webapp服务及相关的环境变量。
13、net

设置网络模式。使用和docker client 的 –net 参数一样的值。

```
# 容器默认连接的网络，是所有Docker安装时都默认安装的docker0网络.
net: "bridge"
# 容器定制的网络栈.
net: "none"
# 使用另一个容器的网络配置
net: "container:[name or id]"
# 在宿主网络栈上添加一个容器，容器中的网络配置会与宿主的一样
net: "host"
```

Docker会为每个节点自动创建三个网络：

网络名称 作用
bridge 容器默认连接的网络，是所有Docker安装时都默认安装的docker0网络
none 容器定制的网络栈
host 在宿主网络栈上添加一个容器，容器中的网络配置会与宿主的一样
附录：

操作名称 命令
创建网络 `docker network create -d bridge mynet`
查看网络列表 `docker network ls`
14、pid

和宿主机系统共享进程命名空间，打开该选项的容器可以相互通过进程id来访问和操作。
`pid: "host"`
15、dns

配置DNS服务器。可以是一个值，也可以是一个列表。

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

16、cap_add，cap_drop

添加或放弃容器的Linux能力（Capability）。

```
cap_add:
 - ALL
cap_drop:
 - NET_ADMIN
 - SYS_ADMIN
```

17、dns_search

配置DNS搜索域。可以是一个值也可以是一个列表。

```
dns_search: example.com
dns_search:
 - domain1.example.com
 \ - domain2.example.com
working_dir, entrypoint, user, hostname, domainname, mem_limit, privileged, restart,
stdin_open, tty, cpu_shares
```

这些都是和 docker run 支持的选项类似。

```
cpu_shares: 73
working_dir: /code
entrypoint: /code/entrypoint.sh
user: postgresql
hostname: foo
domainname: foo.com
mem_limit: 1000000000
privileged: true
restart: always
stdin_open: true
```

```
tty: true
```

docker-compose.yml实例

```
version: "2"

services:
### console
    console:
        build:
            context: ./images/console
            args:
                # console 容器 www-data用户密码
                - USERPASS=root
                - GIT_NAME=yangnan
                - GIT_EMAIL=20706149@qq.com
                - INSTALL_YARN=false
        volumes_from:
            - php-fpm
            - nginx
            - mysql
            - redis
        volumes:
            - ./ssh:/home/www-data/.ssh
        links:
            - redis
            - mysql
        tty: true

### php-fpm
    php-fpm:
        build: ./images/php-fpm
        volumes:
            - ./app/:/var/www/

### nginx
    nginx:
        image: nginx
        ports:
            - "8081:80"
        volumes_from:
            - php-fpm
        volumes:
            - ./logs/nginx/:/var/log/nginx/
            - ./images/nginx/sites:/etc/nginx/conf.d/
        links:
            - php-fpm

### mysql
    mysql:
        image: mysql
        ports:
            - "7706:3306"
        environment:
            MYSQL_ROOT_PASSWORD: "123"
            MYSQL_DATABASE: "test"
            MYSQL_USER: "root"
            MYSQL_PASSWORD: "123"
        volumes:
            - ./data/mysql:/var/lib/mysql
```

```
### redis
    redis:
        image: redis
        ports:
            - "6379:6379"
        volumes:
            - ./data/redis:/data
```

注意事项：

使用compose对Docker容器进行编排管理时，需要编写docker-compose.yml文件，初次编写时，容易遇到一些比较低级的问题，导致执行docker-compose up时先解析yml文件的错误。比较常见的是yml对缩进的严格要求。
yml文件还行后的缩进，不允许使用tab键字符，只能使用空格，而空格的数量也有要求，经过实际测试，发现每一行增加一个空格用于缩进是正常的。
比如：

```
web:
<Tab>build:
<Tab><Tab>command:
...
```

否则，很容易引起各种 `yaml.scanner.ScannerError:` 的错误提示。