# Docker-compose简介

## 1，Docker-compose简介

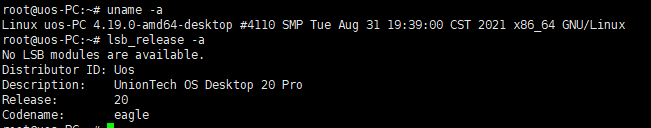
Docker-Compose项目是Docker官方的开源项目，负责实现对Docker容器集群的快速编排。Docker-Compose将所管理的容器分为三层，分别是：

* 工程（project）
* 服务（service）
* 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，若无特殊指定工程名即为当前目录名。一个工程当中可包含多个服务，每个服务中定义了容器运行的镜像，参数，依赖。一个服务当中可包括多个容器实例，Docker-Compose并没有解决负载均衡的问题，因此需要借助其它工具实现服务发现及负载均衡。 Docker-Compose的工程配置文件默认为docker-compose.yml，可通过环境变量COMPOSE\_FILE或-f参数自定义配置文件，其定义了多个有依赖关系的服务及每个服务运行的容器。 使用一个Dockerfile模板文件，可以让用户很方便的定义一个单独的应用容器。在工作中，经常会碰到需要多个容器相互配合来完成某项任务的情况。例如要实现一个Web项目，除了Web服务容器本身，往往还需要再加上后端的数据库服务容器，甚至还包括负载均衡容器等。 Compose允许用户通过一个单独的docker-compose.yml模板文件（YAML 格式）来定义一组相关联的应用容器为一个项目（project）。

Docker-Compose项目由Python编写，调用Docker服务提供的API来对容器进行管理。因此，只要所操作的平台支持Docker API，就可以在其上利用Compose来进行编排管理。

## 2，Docker-compose的安装

安装环境查看  
也可以使用其他发行版本Linux系统  
  
安装  
apt-get install python-pip  
apt-get -y install docker-compose

红帽系Linux使用以下命令安装  
yum -y install python-pip  
yum -y install docker-compose

查看安装的版本  
docker-compose -v  
docker-compose version 1.21.0, build unknown

## 3，Docker-compose卸载

apt-get remove docker-compose

## 二，Docker-compose常用命令

**1，Docker-compose命令格式**

docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]  
命令选项如下

-f --file FILE指定Compose模板文件，默认为docker-compose.yml

-p --project-name NAME 指定项目名称，默认使用当前所在目录为项目名

--verbose 输出更多调试信息

-v，-version 打印版本并退出

--log-level LEVEL 定义日志等级(DEBUG, INFO, WARNING, ERROR, CRITICAL)

**2，docker-compose up**

docker-compose up [options] [--scale SERVICE=NUM...] [SERVICE...]

选项包括：

-d 在后台运行服务容器

-no-color 不是有颜色来区分不同的服务的控制输出

-no-deps 不启动服务所链接的容器

--force-recreate 强制重新创建容器，不能与-no-recreate同时使用

–no-recreate 如果容器已经存在，则不重新创建，不能与–force-recreate同时使用

–no-build 不自动构建缺失的服务镜像

–build 在启动容器前构建服务镜像

–abort-on-container-exit 停止所有容器，如果任何一个容器被停止，不能与-d同时使用

-t, –timeout TIMEOUT 停止容器时候的超时（默认为10秒）

–remove-orphans 删除服务中没有在compose文件中定义的容器

**3，docker-compose ps**

docker-compose ps [options] [SERVICE...]

列出项目中所有在运行的容器

**4，docker-compose stop**

docker-compose stop [options] [SERVICE...]

选项包括

-t, –timeout TIMEOUT 停止容器时候的超时（默认为10秒）

docker-compose stop

停止正在运行的容器，可以通过docker-compose start 再次启动

**5，docker-compose -h**

docker-compose -h

查看帮助

**6，docker-compose down**

docker-compose down [options]

停止和删除容器、网络、卷、镜像。

选项包括：

–rmi type，删除镜像，类型必须是：all，删除compose文件中定义的所有镜像；local，删除镜像名为空的镜像

-v, –volumes，删除已经在compose文件中定义的和匿名的附在容器上的数据卷

–remove-orphans，删除服务中没有在compose中定义的容器

docker-compose down

停用移除所有容器以及网络相关

**7，docker-compose logs**

docker-compose logs [options] [SERVICE...]

查看服务容器的输出。默认情况下，docker-compose将对不同的服务输出使用不同的颜色来区分。可以通过–no-color来关闭颜色。

docker-compose logs

查看服务容器的输出

-f 跟踪日志输出

**8，docker-compose bulid**

docker-compose build [options] [--build-arg key=val...] [SERVICE...]

构建（重新构建）项目中的服务容器。

选项包括：

–compress 通过gzip压缩构建上下环境

–force-rm 删除构建过程中的临时容器

–no-cache 构建镜像过程中不使用缓存

–pull 始终尝试通过拉取操作来获取更新版本的镜像

-m, –memory MEM为构建的容器设置内存大小

–build-arg key=val为服务设置build-time变量

服务容器一旦构建后，将会带上一个标记名。可以随时在项目目录下运行docker-compose build来重新构建服务

**9，docker-compose pull**

docker-compose pull [options] [SERVICE...]

拉取服务依赖的镜像。

选项包括：

–ignore-pull-failures，忽略拉取镜像过程中的错误

–parallel，多个镜像同时拉取

–quiet，拉取镜像过程中不打印进度信息

docker-compose pull

拉取服务依赖的镜像

**10，docker-compose restart**

docker-compose restart [options] [SERVICE...]

重启项目中的服务。

选项包括：

-t, –timeout TIMEOUT，指定重启前停止容器的超时（默认为10秒）

docker-compose restart

重启项目中的服务

**11，docker-compose rm**

docker-compose rm [options] [SERVICE...]

删除所有（停止状态的）服务容器。

选项包括：

–f, –force，强制直接删除，包括非停止状态的容器

-v，删除容器所挂载的数据卷

docker-compose rm

删除所有（停止状态的）服务容器。推荐先执行docker-compose stop命令来停止容器。

**12，docker-compose start**

docker-compose start [SERVICE...]

docker-compose start

启动已经存在的服务容器。

**13，docker-compose run**

docker-compose scale web=3 db=2

设置指定服务运行的容器个数。通过service=num的参数来设置数量

**14，docker-compose scale**

docker-compose scale web=3 db=2

设置指定服务运行的容器个数。通过service=num的参数来设置数量

**15，docker-compose pause**

docker-compose pause [SERVICE...]

暂停一个服务容器

**16，docker-compose kill**

docker-compose kill [options] [SERVICE...]

通过发送SIGKILL信号来强制停止服务容器。

支持通过-s参数来指定发送的信号，例如通过如下指令发送SIGINT信号：

docker-compose kill -s SIGINT

**17，docker-compose config**

docker-compose config [options]

验证并查看compose文件配置。

选项包括：

–resolve-image-digests 将镜像标签标记为摘要

-q, –quiet 只验证配置，不输出。 当配置正确时，不输出任何内容，当文件配置错误，输出错误信息

–services 打印服务名，一行一个

–volumes 打印数据卷名，一行一个

**18，docker-compose create**

docker-compose create [options] [SERVICE...]

为服务创建容器。

选项包括：

–force-recreate：重新创建容器，即使配置和镜像没有改变，不兼容–no-recreate参数

–no-recreate：如果容器已经存在，不需要重新创建，不兼容–force-recreate参数

–no-build：不创建镜像，即使缺失

–build：创建容器前　　，生成镜像

**19，docker-compose exec**

docker-compose exec [options] SERVICE COMMAND [ARGS...]

选项包括：

-d 分离模式，后台运行命令。

–privileged 获取特权。

–user USER 指定运行的用户。

-T 禁用分配TTY，默认docker-compose exec分配TTY。

–index=index，当一个服务拥有多个容器时，可通过该参数登陆到该服务下的任何服务，例如：docker-compose exec –index=1 web /bin/bash ，web服务中包含多个容器

**20，docker-compose port**

docker-compose port [options] SERVICE PRIVATE\_PORT

显示某个容器端口所映射的公共端口。

选项包括：

–protocol=proto，指定端口协议，TCP（默认值）或者UDP

–index=index，如果同意服务存在多个容器，指定命令对象容器的序号（默认为1）

**21，docker-compose push**

docker-compose push [options] [SERVICE...]

推送服务依的镜像。

选项包括：

–ignore-push-failures 忽略推送镜像过程中的错误

**22，docker-compose stop**

docker-compose stop [options] [SERVICE...]

停止运行的容器

**23，docker-compose uppause**

docker-compose unpause [SERVICE...]

恢复处于暂停状态中的服务。

## 三，Docker-compose模板文件

1，Docker-compose模板文件简介

Compose允许用户通过一个docker-compose.yml模板文件（YAML格式）来定义一组相关联的应用容器为一个项目（project）。Compose模板文件是一个定义服务、网络和卷的YAML文件。Compose模板文件默认路径是当前目录下的docker-compose.yml，可以使用.yml或.yaml作为文件扩展名。

Docker-Compose标准模板文件应该包含version、services、networks 三大部分，最关键的是services和networks两个部分。  
举例

version: '3'

services:

web:

image: dockercloud/hello-world

ports:

- 8080

networks:

- front-tier

- back-tier

redis:

image: redis

links:

- web

networks:

- back-tier

lb:

image: dockercloud/haproxy

ports:

- 80:80

links:

- web

networks:

- front-tier

- back-tier

volumes:

- /var/run/docker.sock:/var/run/docker.sock

networks:

front-tier:

driver: bridge

back-tier:

driver: bridge

Compose目前有三个版本分别为Version 1，Version 2，Version 3，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将来会被弃用。

**2，image**

image是指定服务的镜像名称或镜像ID。如果镜像在本地不存在，Compose将会尝试拉取镜像。

services:

web:

image: hello-world

**3，build**

服务除了可以基于指定的镜像，还可以基于一份Dockerfile，在使用up启动时执行构建任务，构建标签是build，可以指定Dockerfile所在文件夹的路径。Compose将会利用Dockerfile自动构建镜像，然后使用镜像启动服务容器。

build: /path/to/build/dir

也可以是相对路径，只要上下文确定就可以读取到Dockerfile。

build: ./dir

设定上下文根目录，然后以该目录为准指定Dockerfile。

build:

context: ../

dockerfile: path/of/Dockerfile

build都是一个目录，如果要指定Dockerfile文件需要在build标签的子级标签中使用dockerfile标签指定。 如果同时指定image和build两个标签，那么Compose会构建镜像并且把镜像命名为image值指定的名字。

**4,context**

context选项可以是Dockerfile的文件路径，也可以是到链接到git仓库的url，当提供的值是相对路径时，被解析为相对于撰写文件的路径，此目录也是发送到Docker守护进程的context

build:

context: ./dir

**5,dockerfile**

使用dockerfile文件来构建，必须指定构建路径

build:

context: .

dockerfile: Dockerfile-alternate

**6,commond**

command: bundle exec thin -p 3000

**7,container\_name**

Compose的容器名称格式是：<项目名称><服务名称><序号>  
可以自定义项目名称、服务名称，但如果想完全控制容器的命名，可以使用标签指定：

container\_name: app

**8,depends\_on**

在使用Compose时，最大的好处就是少打启动命令，但一般项目容器启动的顺序是有要求的，如果直接从上到下启动容器，必然会因为容器依赖问题而启动失败。例如在没启动数据库容器的时候启动应用容器，应用容器会因为找不到数据库而退出。depends\_on标签用于解决容器的依赖、启动先后的问题

version: '2'

services:

web:

build: .

depends\_on:

- db

- redis

redis:

image: redis

db:

image: postgres

上述YAML文件定义的容器会先启动redis和db两个服务，最后才启动web 服务。

**9,PID**

pid: "host"

将PID模式设置为主机PID模式，跟主机系统共享进程命名空间。容器使用pid标签将能够访问和操纵其他容器和宿主机的名称空间。

**10,ports**

ports用于映射端口的标签。  
使用HOST:CONTAINER格式或者只是指定容器的端口，宿主机会随机映射端口。

ports:

- "3000"

- "8000:8000"

- "49100:22"

- "127.0.0.1:8001:8001"

当使用HOST:CONTAINER格式来映射端口时，如果使用的容器端口小于60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。

**11,extra\_hosts**

添加主机名的标签，会在/etc/hosts文件中添加一些记录。

extra\_hosts:

- "somehost:162.242.195.82"

- "otherhost:50.31.209.229"

启动后查看容器内部hosts：

162.242.195.82 somehost

50.31.209.229 otherhost

**12,volumes**

挂载一个目录或者一个已存在的数据卷容器，可以直接使用 [HOST:CONTAINER]格式，或者使用[HOST:CONTAINER:ro]格式，后者对于容器来说，数据卷是只读的，可以有效保护宿主机的文件系统。 Compose的数据卷指定路径可以是相对路径，使用 . 或者 .. 来指定相对目录。 数据卷的格式可以是下面多种形式

volumes:

// 只是指定一个路径，Docker 会自动在创建一个数据卷（这个路径是容器内部的）。

- /var/lib/mysql

// 使用绝对路径挂载数据卷

- /opt/data:/var/lib/mysql

// 以 Compose 配置文件为中心的相对路径作为数据卷挂载到容器。

- ./cache:/tmp/cache

// 使用用户的相对路径（~/ 表示的目录是 /home/<用户目录>/ 或者 /root/）。

- ~/configs:/etc/configs/:ro

// 已经存在的命名的数据卷。

- datavolume:/var/lib/mysql

如果不使用宿主机的路径，可以指定一个volume\_driver。

**13,volumes\_from**

从另一个服务或容器挂载其数据卷：

volumes\_from:

- service\_name

- container\_name

**14,dns**

自定义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

**15,expose**

暴露端口，但不映射到宿主机，只允许能被连接的服务访问。仅可以指定内部端口为参数，如下所示：

expose:

- "3000"

- "8000"

**16,links**

链接到其它服务中的容器。使用服务名称（同时作为别名），或者服务名称:服务别名（如 SERVICE:ALIAS），例如

links:

- db

- db:database

- redis

**17,net**

设置网络模式。

net: "bridge"

net: "none"

net: "host"

## 四，Docker-compose模板文件示例

1，Docker-compose模板文件编写

docker-compose.yml

version: '2'

services:

web1:

image: nginx

ports:

- "6061:80"

container\_name: "web1"

networks:

- dev

web2:

image: nginx

ports:

- "6062:80"

container\_name: "web2"

networks:

- dev

- pro

web3:

image: nginx

ports:

- "6063:80"

container\_name: "web3"

networks:

- pro

networks:

dev:

driver: bridge

pro:

driver: bridge

docker-compose.yml文件指定了3个web服务

2，启动应用

创建一个webapp目录，将docker-compose.yaml文件拷贝到webapp目录下，使用docker-compose启动应用。

docker-compose up -d

3，服务访问

通过浏览器访问web1，web2，web3

http://127.0.0.1:6061

http://127.0.0.1:6062

http://127.0.0.1:6063