<https://blog.csdn.net/zhuchunyan_aijia/article/details/80111629>

先来看一份 docker-compose.yml 文件

1. version: '2'
2. services:
3. web:
4. image: dockercloud/hello-world
5. ports:
6. - 8080
7. networks:
8. - front-tier
9. - back-tier
11. redis:
12. image: redis
13. links:
14. - web
15. networks:
16. - back-tier
18. lb:
19. image: dockercloud/haproxy
20. ports:
21. - 80:80
22. links:
23. - web
24. networks:
25. - front-tier
26. - back-tier
27. volumes:
28. - /var/run/docker.sock:/var/run/docker.sock
30. networks:
31. front-tier:
32. driver: bridge
33. back-tier:
34. driver: bridge

可以看到一份标准配置文件应该包含 version、services、networks 三大部分，其中最关键的就是 services 和 networks 两个部分，下面先来看 services 的书写规则。

**1. image**

1. services:
2. web:
3. image: hello-world

在 services 标签下的第二级标签是 web，这个名字是用户自己自定义，它就是服务名称。  
image 则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 将会尝试拉取这个镜像。  
例如下面这些格式都是可以的：

1. image: redis
2. image: ubuntu:14.04
3. image: tutum/influxdb
4. image: example-registry.com:4000/postgresql
5. image: a4bc65fd

**2. 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 启动之时执行构建任务，这个构建标签就是 build，它可以指定 Dockerfile 所在文件夹的路径。Compose 将会利用它自动构建这个镜像，然后使用这个镜像启动服务容器。

1. 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。

1. build:
2. context: ../
3. dockerfile: path/of/Dockerfile

注意 build 都是一个目录，如果你要指定 Dockerfile 文件需要在 build 标签的子级标签中使用 dockerfile 标签指定，如上面的例子。  
如果你同时指定了 image 和 build 两个标签，那么 Compose 会构建镜像并且把镜像命名为 image 后面的那个名字。

1. build: ./dir
2. image: webapp:tag

既然可以在 docker-compose.yml 中定义构建任务，那么一定少不了 arg 这个标签，就像 Dockerfile 中的 ARG 指令，它可以在构建过程中指定环境变量，但是在构建成功后取消，在 docker-compose.yml 文件中也支持这样的写法：

1. build:
2. context: .
3. args:
4. buildno: 1
5. password: secret

下面这种写法也是支持的，一般来说下面的写法更适合阅读。

1. build:
2. context: .
3. args:
4. - buildno=1
5. - password=secret

与 ENV 不同的是，ARG 是允许空值的。例如：

1. args:
2. - buildno
3. - password

这样构建过程可以向它们赋值。

注意：YAML 的布尔值（true, false, yes, no, on, off）必须要使用引号引起来（单引号、双引号均可），否则会当成字符串解析。

**3. command**

使用 command 可以覆盖容器启动后默认执行的命令。

command: bundle exec thin -p 3000

也可以写成类似 Dockerfile 中的格式：

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

**4.container\_name**

前面说过 Compose 的容器名称格式是：<项目名称><服务名称><序号>  
虽然可以自定义项目名称、服务名称，但是如果你想完全控制容器的命名，可以使用这个标签指定：

container\_name: app

这样容器的名字就指定为 app 了。

**5.depends\_on**

在使用 Compose 时，最大的好处就是少打启动命令，但是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了应用容器，这时候应用容器会因为找不到数据库而退出，为了避免这种情况我们需要加入一个标签，就是 depends\_on，这个标签解决了容器的依赖、启动先后的问题。  
例如下面容器会先启动 redis 和 db 两个服务，最后才启动 web 服务：

1. version: '2'
2. services:
3. web:
4. build: .
5. depends\_on:
6. - db
7. - redis
8. redis:
9. image: redis
10. db:
11. image: postgres

注意的是，默认情况下使用 docker-compose up web 这样的方式启动 web 服务时，也会启动 redis 和 db 两个服务，因为在配置文件中定义了依赖关系。

**6.dns**

和 --dns 参数一样用途，格式如下：

1. dns: 8.8.8.8

也可以是一个列表：

1. dns:
2. - 8.8.8.8
3. - 9.9.9.9

此外 dns\_search 的配置也类似：

1. dns\_search: example.com
2. dns\_search:
3. - dc1.example.com
4. - dc2.example.com

**7. tmpfs**

挂载临时目录到容器内部，与 run 的参数一样效果：

1. tmpfs: /run
2. tmpfs:
3. - /run
4. - /tmp

**8. entrypoint**

在 Dockerfile 中有一个指令叫做 ENTRYPOINT 指令，用于指定接入点，第四章有对比过与 CMD 的区别。  
在 docker-compose.yml 中可以定义接入点，覆盖 Dockerfile 中的定义：

1. entrypoint: /code/entrypoint.sh

格式和 Docker 类似，不过还可以写成这样：

1. entrypoint:
2. - php
3. - -d
4. - zend\_extension=/usr/local/lib/php/extensions/no-debug-non-zts-20100525/xdebug.so
5. - -d
6. - memory\_limit=-1
7. - vendor/bin/phpunit

**9.env\_file**

还记得前面提到的 .env 文件吧，这个文件可以设置 Compose 的变量。而在 docker-compose.yml 中可以定义一个专门存放变量的文件。  
如果通过 docker-compose -f FILE 指定了配置文件，则 env\_file 中路径会使用配置文件路径。

如果有变量名称与 environment 指令冲突，则以后者为准。格式如下：

1. env\_file: .env

或者根据 docker-compose.yml 设置多个：

1. env\_file:
2. - ./common.env
3. - ./apps/web.env
4. - /opt/secrets.env

注意的是这里所说的环境变量是对宿主机的 Compose 而言的，如果在配置文件中有 build 操作，这些变量并不会进入构建过程中，如果要在构建中使用变量还是首选前面刚讲的 arg 标签。

**10. environment**

与上面的 env\_file 标签完全不同，反而和 arg 有几分类似，这个标签的作用是设置镜像变量，它可以保存变量到镜像里面，也就是说启动的容器也会包含这些变量设置，这是与 arg 最大的不同。  
一般 arg 标签的变量仅用在构建过程中。而 environment 和 Dockerfile 中的 ENV 指令一样会把变量一直保存在镜像、容器中，类似 docker run -e 的效果。

1. environment:
2. RACK\_ENV: development
3. SHOW: 'true'
4. SESSION\_SECRET:
6. environment:
7. - RACK\_ENV=development
8. - SHOW=true
9. - SESSION\_SECRET

**11. expose**

这个标签与Dockerfile中的EXPOSE指令一样，用于指定暴露的端口，但是只是作为一种参考，实际上docker-compose.yml的端口映射还得ports这样的标签。

1. expose:
2. - "3000"
3. - "8000"

**12. external\_links**

在使用Docker过程中，我们会有许多单独使用docker run启动的容器，为了使Compose能够连接这些不在docker-compose.yml中定义的容器，我们需要一个特殊的标签，就是external\_links，它可以让Compose项目里面的容器连接到那些项目配置外部的容器（前提是外部容器中必须至少有一个容器是连接到与项目内的服务的同一个网络里面）。  
格式如下：

1. external\_links:
2. - redis\_1
3. - project\_db\_1:mysql
4. - project\_db\_1:postgresql

**13. extra\_hosts**

添加主机名的标签，就是往/etc/hosts文件中添加一些记录，与Docker client的--add-host类似：

1. extra\_hosts:
2. - "somehost:162.242.195.82"
3. - "otherhost:50.31.209.229"

启动之后查看容器内部hosts：

1. 162.242.195.82 somehost
2. 50.31.209.229 otherhost

**14. labels**

向容器添加元数据，和Dockerfile的LABEL指令一个意思，格式如下：

1. labels:
2. com.example.description: "Accounting webapp"
3. com.example.department: "Finance"
4. com.example.label-with-empty-value: ""
5. labels:
6. - "com.example.description=Accounting webapp"
7. - "com.example.department=Finance"
8. - "com.example.label-with-empty-value"

**15. links**

还记得上面的depends\_on吧，那个标签解决的是启动顺序问题，这个标签解决的是容器连接问题，与Docker client的--link一样效果，会连接到其它服务中的容器。  
格式如下：

1. links:
2. - db
3. - db:database
4. - redis

使用的别名将会自动在服务容器中的/etc/hosts里创建。例如：

1. 172.12.2.186 db
2. 172.12.2.186 database
3. 172.12.2.187 redis

相应的环境变量也将被创建。

**16. logging**

这个标签用于配置日志服务。格式如下：

1. logging:
2. driver: syslog
3. options:
4. syslog-address: "tcp://192.168.0.42:123"

默认的driver是json-file。只有json-file和journald可以通过docker-compose logs显示日志，其他方式有其他日志查看方式，但目前Compose不支持。对于可选值可以使用options指定。  
有关更多这方面的信息可以阅读官方文档：  
[https://docs.docker.com/engine/admin/logging/overview/](https://link.jianshu.com/?t=https://docs.docker.com/engine/admin/logging/overview/)

**17. 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。容器使用这个标签将能够访问和操纵其他容器和宿主机的名称空间。

**18. ports**

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。

1. ports:
2. - "3000"
3. - "8000:8000"
4. - "49100:22"
5. - "127.0.0.1:8001:8001"

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。

**19. security\_opt**

为每个容器覆盖默认的标签。简单说来就是管理全部服务的标签。比如设置全部服务的user标签值为USER。

1. security\_opt:
2. - label:user:USER
3. - label:role:ROLE

**20. stop\_signal**

设置另一个信号来停止容器。在默认情况下使用的是SIGTERM停止容器。设置另一个信号可以使用stop\_signal标签。

stop\_signal: SIGUSR1

**21. 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] 这样的格式，或者使用 [HOST:CONTAINER:ro] 这样的格式，后者对于容器来说，数据卷是只读的，这样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：

1. volumes:
2. *// 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）。*
3. - /var/lib/mysql
5. *// 使用绝对路径挂载数据卷*
6. - /opt/data:/var/lib/mysql
8. *// 以 Compose 配置文件为中心的相对路径作为数据卷挂载到容器。*
9. - ./cache:/tmp/cache
11. *// 使用用户的相对路径（~/ 表示的目录是 /home/<用户目录>/ 或者 /root/）。*
12. - ~/configs:/etc/configs/:ro
14. *// 已经存在的命名的数据卷。*
15. - datavolume:/var/lib/mysql

如果你不使用宿主机的路径，你可以指定一个volume\_driver。

volume\_driver: mydriver

**22. volumes\_from**

从其它容器或者服务挂载数据卷，可选的参数是 :ro或者 :rw，前者表示容器只读，后者表示容器对数据卷是可读可写的。默认情况下是可读可写的。

1. volumes\_from:
2. - service\_name
3. - service\_name:ro
4. - container:container\_name
5. - container:container\_name:rw

**23. cap\_add, cap\_drop**

添加或删除容器的内核功能。详细信息在前面容器章节有讲解，此处不再赘述。

1. cap\_add:
2. - ALL
4. cap\_drop:
5. - NET\_ADMIN
6. - SYS\_ADMIN

**24. cgroup\_parent**

指定一个容器的父级cgroup。

cgroup\_parent: m-executor-abcd

**25. devices**

设备映射列表。与Docker client的--device参数类似。

1. devices:
2. - "/dev/ttyUSB0:/dev/ttyUSB0"

**26. extends**

这个标签可以扩展另一个服务，扩展内容可以是来自在当前文件，也可以是来自其他文件，相同服务的情况下，后来者会有选择地覆盖原有配置。

1. extends:
2. file: common.yml
3. service: webapp

用户可以在任何地方使用这个标签，只要标签内容包含file和service两个值就可以了。file的值可以是相对或者绝对路径，如果不指定file的值，那么Compose会读取当前YML文件的信息。  
更多的操作细节在后面的12.3.4小节有介绍。

**27. network\_mode**

网络模式，与Docker client的--net参数类似，只是相对多了一个service:[service name] 的格式。  
例如：

1. network\_mode: "bridge"
2. network\_mode: "host"
3. network\_mode: "none"
4. network\_mode: "service:[service name]"
5. network\_mode: "container:[container name/id]"

可以指定使用服务或者容器的网络。

**28. networks**

加入指定网络，格式如下：

1. services:
2. some-service:
3. networks:
4. - some-network
5. - other-network

关于这个标签还有一个特别的子标签aliases，这是一个用来设置服务别名的标签，例如：

1. services:
2. some-service:
3. networks:
4. some-network:
5. aliases:
6. - alias1
7. - alias3
8. other-network:
9. aliases:
10. - alias2

相同的服务可以在不同的网络有不同的别名。

**29. 其它**

还有这些标签：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  
上面这些都是一个单值的标签，类似于使用docker run的效果。

1. cpu\_shares: 73
2. cpu\_quota: 50000
3. cpuset: 0,1
5. user: postgresql
6. working\_dir: /code
8. domainname: foo.com
9. hostname: foo
10. ipc: host
11. mac\_address: 02:42:ac:11:65:43
13. mem\_limit: 1000000000
14. memswap\_limit: 2000000000
15. privileged: true
17. restart: always
19. read\_only: true
20. shm\_size: 64M
21. stdin\_open: true
22. tty: true