docker-compose.yml 配置文件编写详解

作者：Leif160519
原文链接：https://ld246.com/article/1585017483391
来源网站：链滴

本文摘自：https://blog.csdn.net/qq_36148847/article/details/79427878

docker compose 在 Docker 容器运用中具有很大的学习意义，docker compose 是一个整合发布用的利器。而使用docker compose时，懂得如何编排 docker compose 配置文件是很重要的。

一. 前言

关于 docker compose 技术可以查看官方文档 Docker Compose

以下的内容是确立在已经下载好 Docker 以及 Docker Compose，可参看 Docker Compose 的官方装教程 Install Docker Compose

二. Docker Compose 配置文件的构建参数说明

首先，官方提供了一个 yaml Docker Compose 配置文件的标准例子

```yaml
version: "3"
services:

  redis:
    image: redis:alpine
    ports:
      - "6379"
    networks:
      - frontend
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]

  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 2
```

```
    update_config:
      parallelism: 2
    restart_policy:
      condition: on-failure

  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - 5001:80
    networks:
      - backend
    depends_on:
      - db
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]

  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]

networks:
  frontend:
  backend:

volumes:
```

```
  db-data:
```

此文件配置了多个服务，关于此配置文件的各个语句含义就需要弄懂配置选项的含义了

文件配置

compose 文件是一个定义服务、 网络和卷的 YAML 文件 。Compose 文件的默认路径是 ./docker-compose.yml

提示：可以是用 .yml 或 .yaml 作为文件扩展名

服务定义包含应用于为该服务启动的每个容器的配置，就像传递命令行参数一样 docker container create。同样，网络和卷的定义类似于 docker network create 和 docker volume create。

正如 docker container create 在 Dockerfile 指定选项，如 CMD、 EXPOSE、VOLUME、ENV，默认情况下，你不需要再次指定它们docker-compose.yml。

可以使用 Bash 类 ${VARIABLE} 语法在配置值中使用环境变量。

配置选项

1 .bulid

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镜像，然后使用这个镜像启动服务容器

```
build: /path/to/build/dir
```

也可以是相对路径

```
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
```

例子

```
version: '3'
services:
  webapp:
    build: ./dir
```

如果 context 中有指定的路径，并且可以选定 Dockerfile 和 args。那么 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：

```
version: '3'
services:
  webapp:
```

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

与 ENV 不同的是，ARG 可以为空值

```
args:
  - buildno
  - password
```

如果要指定image以及 build ，选项格式为

```
build: ./dir
image: webapp:tag
```

这会在 ./dir 目录生成一个名为 webaapp 和标记为 tag 的镜像

2. 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，它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 ontext

```
build:
  context: ./dir
```

3 . dockerfile

使用此dockerfile文件来构建，必须指定构建路径

```
build:
  context: .
  dockerfile: Dockerfile-alternate
```

4 . args

添加构建参数，这些参数是仅在构建过程中可访问的环境变量

首先， 在Dockerfile中指定参数：

```
ARG buildno
ARG password
RUN echo "Build number: $buildno"
RUN script-requiring-password.sh "$password"
```

然后指定build下的参数,可以传递映射或列表

```
build:
  context: .
  args:
```

```
    buildno: 1
    password: secret
```

或

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

指定构建参数时可以省略该值，在这种情况下，构建时的值默认构成运行环境中的值

```
args:
  - buildno
  - password
```

Note： YAML 布尔值（true，false，yes，no，on，off）必须使用引号括起来，以为了能够正常被析为字符串

5 . cache_from

编写缓存解析镜像列表

```
build:
  context: .
  cache_from:
    - alpine:latest
    - corp/web_app:3.14
```

6. labels

使用 Docker标签 将元数据添加到生成的镜像中，可以使用数组或字典。

建议使用反向 DNS 标记来防止签名与其他软件所使用的签名冲突

```
build:
  context: .
  labels:
    com.example.description: "Accounting webapp"
    com.example.department: "Finance"
    com.example.label-with-empty-value: ""
```

或

```
build:
  context: .
  labels:
    - "com.example.description=Accounting webapp"
    - "com.example.department=Finance"
    - "com.example.label-with-empty-value"
```

7.shm_size

设置容器/dev/shm分区的大小，值为表示字节的整数值或表示字符的字符串

```
build:
  context: .
  shm_size: '2gb'
```

或

```
build:
  context: .
  shm_size: 10000000
```

8. target

根据对应的 Dockerfile 构建指定 Stage

```
build:
    context: .
    target: prod
```

9. cap_add、cap_drop

添加或删除容器功能，可查看 man 7 capabilities

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

Note:当用(Version 3) Compose 文件在群集模式下部署堆栈时，该选项被忽略。因为 docker stack 命令只接受预先构建的镜像

10. command

覆盖容器启动后默认执行的命令

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

该命令也可以是一个列表，方法类似于 dockerfile:

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

11. configs

使用服务 configs 配置为每个服务赋予相应的访问权限，支持两种不同的语法。

Note: 配置必须存在或在 configs 此堆栈文件的顶层中定义，否则堆栈部署失效

1.SHORT 语法

SHORT 语法只能指定配置名称，这允许容器访问配置并将其安装在 /<config_name> 容器内，源名

和目标装入点都设为配置名称。

```
version: "3.3"

services:
  redis:
    image: redis:latest
    deploy:
      replicas: 1
    configs:
      - my_config
      - my_other_config
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

以上实例使用 SHORT 语法将 redis 服务访问授予 my_config 和 my_other_config ,并被 my_other_onfig 定义为外部资源，这意味着它已经在 Docker 中定义。可以通过docker config create命令或过另一个堆栈部署。如果外部部署配置都不存在，则堆栈部署会失败并出现 config not found 错误。

Note: config 定义仅在 3.3 版本或在更高版本的撰写文件格式中受支持，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。

2. LONG 语法

LONG 语法提供了创建服务配置的更加详细的信息

source:Docker 中存在的配置的名称

target:要在服务的任务中装载的文件的路径或名称。如果未指定则默认为 /<source>

uid 和 gid:在服务的任务容器中拥有安装的配置文件的数字 UID 或 GID。如果未指定，则默认为在Linux上。Windows不支持。

mode:在服务的任务容器中安装的文件的权限，以八进制表示法。例如，0444 代表文件可读的。默是 0444。如果配置文件无法写入，是因为它们安装在临时文件系统中，所以如果设置了可写位，它被忽略。可执行位可以设置。如果您不熟悉 UNIX 文件权限模式，Unix Permissions Calculator

下面示例在容器中将 my_config 名称设置为 redis_config，将模式设置为 0440（group-readable 并将用户和组设置为 103。该 redis 服务无法访问 my_other_config 配置。

```
version: "3.3"
services:
  redis:
    image: redis:latest
    deploy:
      replicas: 1
    configs:
      - source: my_config
        target: /redis_config
        uid: '103'
        gid: '103'
```

```
    mode: 0440
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

可以同时授予多个配置的服务相应的访问权限，也可以混合使用 LONG 和 SHORT 语法。定义配置不意味着授予服务访问权限。

12. cgroup_parent

可以为容器选择一个可选的父 cgroup

```
cgroup_parent: m-executor-abcd
```

注意：当 使用（Version 3）Compose 文件在群集模式下部署堆栈时，忽略此选项

13. container_name

为自定义的容器指定一个名称，而不是使用默认的名称

```
container_name: my-web-container
```

因为 docker 容器名称必须是唯一的，所以如果指定了一个自定义的名称，不能扩展一个服务超过 1 容器

14. credential_spec

为托管服务账户配置凭据规范，此选项仅适用于 Windows 容器服务

在 credential_spec 上的配置列表格式为file://<filename>或 registry://<value-name>

使用file:应该注意引用的文件必须存在于 CredentialSpecs,docker 数据目录的子目录中。在 Windows 上，该目录默认为 C:\ProgramData\Docker\。以下示例从名为C:\ProgramData\Docker\CredentialSpecs\my-credential-spec.json 的文件加载凭证规范：

```
credential_spec:
  file: my-credential-spec.json
```

使用 registry: 将从守护进程主机上的 Windows 注册表中读取凭据规范。其注册表值必须位于：

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Virtualization\Containers\CredentialSpecs
```

下面的示例通过 my-credential-spec 注册表中指定的值加载凭证规范：

```
credential_spec:
  registry: my-credential-spec
```

15. deploy

指定与部署和运行服务相关的配置

```
version: '3'
services:
  redis:
    image: redis:alpine
    deploy:
      replicas: 6
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
```

这里有几个子选项

1. endpoint_mode

指定连接到群组外部客户端服务发现方法

endpoint_mode:vip： Docker 为该服务分配了一个虚拟 IP(VIP),作为客户端的 “前端 “ 部位用于问网络上的服务。

endpoint_mode: dnsrr : DNS轮询（DNSRR）服务发现不使用单个虚拟 IP。Docker为服务设置 DNS 条目，使得服务名称的 DNS 查询返回一个 IP 地址列表，并且客户端直接连接到其中的一个。如果想用自己的负载平衡器，或者混合 Windows 和 Linux 应用程序，则 DNS 轮询调度（round-robin）能就非常实用。

```
version: "3.3"
services:

  wordpress:
    image: wordpress
    ports:
      - 8080:80
    networks:
      - overlay
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: vip

  mysql:
    image: mysql
    volumes:
       - db-data:/var/lib/mysql/data
    networks:
       - overlay
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: dnsrr

volumes:
```

```
  db-data:

networks:
  overlay:
```

相关信息：Swarm 模式 CLI 命令 、Configure 服务发现

2.labels

指定服务的标签，这些标签仅在服务上设置。

```
version: "3"
services:
  web:
    image: web
    deploy:
      labels:
        com.example.description: "This label will appear on the web service"
```

通过将 deploy 外面的 labels 标签来设置容器上的 labels

```
version: "3"
services:
  web:
    image: web
    labels:
      com.example.description: "This label will appear on all containers for the web service"
```

3.mode

global:每个集节点只有一个容器

replicated:指定容器数量（默认）

```
version: '3'
services:
  worker:
    image: dockersamples/examplevotingapp_worker
    deploy:
      mode: global
```

4. placement

指定 constraints 和 preferences

```
version: '3'
services:
  db:
    image: postgres
    deploy:
      placement:
        constraints:
          - node.role == manager
```

```
      - engine.labels.operatingsystem == ubuntu 14.04
    preferences:
      - spread: node.labels.zone
```

5 .replicas

如果服务是 replicated（默认），需要指定运行的容器数量

```
version: '3'
services:
  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 6
```

6. resources

配置资源限制

```
version: '3'
services:
  redis:
    image: redis:alpine
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
```

此例子中，redis 服务限制使用不超过 50M 的内存和 0.50（50%）可用处理时间（CPU），并且 保 20M 了内存和 0.25 CPU时间

7. restart_policy

配置容器的重新启动，代替 restart

condition:值可以为 none 、on-failure 以及 any(默认)

delay: 尝试重启的等待时间，默认为 0

max_attempts:在放弃之前尝试重新启动容器次数（默认：从不放弃）。如果重新启动在配置中没有功 window，则此尝试不计入配置max_attempts 值。例如，如果max_attempts值为 2，并且第一尝试重新启动失败，则可能会尝试重新启动两次以上。

windows:在决定重新启动是否成功之前的等时间，指定为持续时间（默认值：立即决定）。

```yaml
version: "3"
services:
  redis:
    image: redis:alpine
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
```

8. update_config

配置更新服务，用于无缝更新应用（rolling update)

parallelism：一次性更新的容器数量

delay：更新一组容器之间的等待时间。

failure_action：如果更新失败，可以执行的的是 continue、rollback 或 pause （默认）

monitor：每次任务更新后监视失败的时间(ns|us|ms|s|m|h)（默认为0）

max_failure_ratio：在更新期间能接受的失败率

order：更新次序设置，top-first（旧的任务在开始新任务之前停止）、start-first（新的任务首先启，并且正在运行的任务短暂重叠）（默认 stop-first)

```yaml
version: '3.4'
services:
  vote:
    image: dockersamples/examplevotingapp_vote:before
    depends_on:
      - redis
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
        delay: 10s
        order: stop-first
```

不支持 ****Docker stack desploy ****的几个子选项

build、cgroup_parent、container_name、devices、tmpfs、external_links、inks、network_mode、restart、security_opt、stop_signal、sysctls、userns_mode

16. devices

设置映射列表，与 Docker 客户端的 --device 参数类似：

```yaml
devices:
  - "/dev/ttyUSB0:/dev/ttyUSB0"
```

17. depends_on

此选项解决了启动顺序的问题

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，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，是 depends_on，这个标签解决了容器的依赖、启动先后的问题。

指定服务之间的依赖关系，有两种效果

docker-compose up 以依赖顺序启动服务，下面例子中 redis 和 db 服务在 web 启动前启动

docker-compose up SERVICE 自动包含 SERVICE 的依赖性，下面例子中，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：

```
version: '3'
services:
  web:
    build: .
    depends_on:
      - db
      - redis
  redis:
    image: redis
  db:
    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

18. dns

自定义 DNS 服务器，与 --dns 具有一样的用途，可以是单个值或列表

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

19. dns_search

自定义 DNS 搜索域，可以是单个值或列表

```
dns_search: example.com
dns_search:
  - dc1.example.com
  - dc2.example.com
```

20. tmpfs

挂载临时文件目录到容器内部，与 run 的参数一样效果，可以是单个值或列表

```
tmpfs: /run
tmpfs:
  - /run
  - /tmp
```

21. entrypoint

在 Dockerfile 中有一个指令叫做 ENTRYPOINT 指令，用于指定接入点。在 docker-compose.yml 可以定义接入点，覆盖 Dockerfile 中的定义：

```
entrypoint: /code/entrypoint.sh
```

entrypoint 也可以是一个列表，方法类似于 dockerfile

```
entrypoint:
    - php
    - -d
    - zend_extension=/usr/local/lib/php/extensions/no-debug-non-zts-20100525/xdebug.so
    - -d
    - memory_limit=-1
    - vendor/bin/phpunit
```

21. env_file

从文件中添加环境变量。可以是单个值或是列表

如果已经用 docker-compose -f FILE 指定了 Compose 文件，那么 env_file 路径值为相对于该文件在的目录

但 environment 环境中的设置的变量会会覆盖这些值，无论这些值未定义还是为 None

```
env_file: .env
```

或者根据 docker-compose.yml 设置多个：

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

环境配置文件 env_file 中的声明每行都是以 VAR=VAL 格式，其中以 # 开头的被解析为注释而被忽略

注意环境变量配置列表的顺序,例如下面例子

docker_compose.yml

```
services:
  some-service:
    env_file:
      - a.env
      - b.env
```

a.env 文件

```
# a.env
VAR=1
```

b.env文件

对于在文件a.env 中指定的相同变量但在文件 b.env 中分配了不同的值，如果 b.env 像下面列在 a.env 后，则刚在 a.env 设置的值被 b.env 相同变量的值覆盖，此时 $VAR 值为 hello。此外，这里所说的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程中，如果要在构建中使用变量还是首选arg标签

22. environment

添加环境变量，可以使用数组或字典。与上面的 env_file 选项完全不同，反而和 arg 有几分类似，这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，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 指令一样会把变量直保存在镜像、容器中，类似 docker run -e 的效果

```
environment:
  RACK_ENV: development
  SHOW: 'true'
  SESSION_SECRET:
```

或

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

23. expose

暴露端口，但不映射到宿主机，只被连接的服务访问。这个标签与 Dockerfile 中的 EXPOSE 指令一，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 por s 这样的标签

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

24. external_links

链接到 docker-compose.yml 外部的容器，甚至 并非 Compose 项目文件管理的容器。参数格式跟 l nks 类似

在使用Docker过程中，会有许多单独使用 docker run 启动的容器的情况，为了使 Compose 能够连这些不在docker-compose.yml 配置文件中定义的容器，那么就需要一个特殊的标签，就是 external_ inks，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

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

25. 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 客户端 中的 --add-host 似：

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

具有 IP 地址和主机名的条目在 /etc/hosts 内部容器中创建。启动之后查看容器内部 hosts ，例如：

```
162.242.195.82  somehost
50.31.209.229   otherhost
```

26.healthcheck

用于检查测试服务使用的容器是否正常

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost"]
  interval: 1m30s
  timeout: 10s
  retries: 3
  start_period: 40s
```

interval，timeout 以及start_period都定为持续时间

test 必须是字符串或列表，如果它是一个列表，第一项必须是 NONE，CMD 或 CMD-SHELL ；如它是一个字符串，则相当于指定CMD-SHELL 后跟该字符串。

```
# Hit the local web app
test: ["CMD", "curl", "-f", "http://localhost"]
# As above, but wrapped in /bin/sh. Both forms below are equivalent.
test: ["CMD-SHELL", "curl -f http://localhost || exit 1"]
test: curl -f https://localhost || exit 1
```

如果需要禁用镜像的所有检查项目，可以使用 disable:true,相当于 test:["NONE"]

```
healthcheck:
  disable: true
```

27. image

从指定的镜像中启动容器，可以是存储仓库、标签以及镜像 ID

```
image: redis
```

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

如果镜像不存在，Compose 会自动拉去镜像

28. isolation

Linux 上仅仅支持 default 值

29. labels

使用 Docker 标签将元数据添加到容器，可以使用数组或字典。与 Dockerfile 中的LABELS类似：

```
labels:
  com.example.description: "Accounting webapp"
  com.example.department: "Finance"
  com.example.label-with-empty-value: ""

labels:
  - "com.example.description=Accounting webapp"
  - "com.example.department=Finance"
  - "com.example.label-with-empty-value"
```

30.links

链接到其它服务的中的容器，可以指定服务名称也可以指定链接别名（SERVICE：ALIAS)，与 Docker 客户端的 --link有一样效果，会连接到其它服务中的容器

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

使用的别名将会自动在服务容器中的 /etc/hosts 里创建。例如：

```
172.12.2.186  db
172.12.2.186  database
172.12.2.187  redis
```

相应的环境变量也将被创建

31. logging

配置日志服务

```
logging:
  driver: syslog
  options:
    syslog-address: "tcp://192.168.0.42:123"
```

该 driver值是指定服务器的日志记录驱动程序，默认值为 json-file,与 --log-diver 选项一样

```
driver: "json-file"
driver: "syslog"
driver: "none"
```

注意：只有驱动程序json-file和journald驱动程序可以直接从 docker-compose up 和 docker-compose logs 获取日志。使用任何其他方式不会显示任何日志。

对于可选值，可以使用 options 指定日志记录中的日志记录选项

```
driver: "syslog"
options:
  syslog-address: "tcp://192.168.0.42:123"
```

默认驱动程序 json-file 具有限制存储日志量的选项，所以，使用键值对来获得最大存储大小以及最存储数量

```
options:
  max-size: "200k"
  max-file: "10"
```

上面实例将存储日志文件，直到它们达到max-size:200kB，存储的单个日志文件的数量由该 max-file 值指定。随着日志增长超出最大限制，旧日志文件将被删除以存储新日志

docker-compose.yml 限制日志存储的示例

```
services:
  some-service:
    image: some-service
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
```

32. network_mode

网络模式，用法类似于 Docke 客户端的 --net 选项，格式为：service:[service name]

```
network_mode: "bridge"
network_mode: "host"
network_mode: "none"
network_mode: "service:[service name]"
network_mode: "container:[container name/id]"
```

可以指定使用服务或者容器的网络

33. networks

加入指定网络

```
services:
  some-service:
```

```
    networks:
     - some-network
     - other-network
```

34. aliases

同一网络上的其他容器可以使用服务器名称或别名来连接到其他服务的容器

```
services:
  some-service:
    networks:
      some-network:
        aliases:
         - alias1
         - alias3
      other-network:
        aliases:
         - alias2
```

下面实例中，提供 web 、worker以及db 服务，伴随着两个网络new和legacy。

```
version: '2'

services:
  web:
    build: ./web
    networks:
      - new
  worker:
    build: ./worker
    networks:
      - legacy
  db:
    image: mysql
    networks:
      new:
        aliases:
          - database
      legacy:
        aliases:
          - mysql

networks:
  new:
  legacy:
```

相同的服务可以在不同的网络有不同的别名

35. ipv4_address、ipv6_address

为服务的容器指定一个静态 IP 地址

```
version: '2.1'
```

```yaml
services:
  app:
    image: busybox
    command: ifconfig
    networks:
      app_net:
        ipv4_address: 172.16.238.10
        ipv6_address: 2001:3984:3989::10


networks:
  app_net:
    driver: bridge
    enable_ipv6: true
    ipam:
      driver: default
      config:
      -
        subnet: 172.16.238.0/24
      -
        subnet: 2001:3984:3989::/64
```

36. PID

```yaml
pid: "host"
```

将 PID 模式设置为主机 PID 模式，可以打开容器与主机操作系统之间的共享 PID 地址空间。使用此志启动的容器可以访问和操作宿主机的其他容器，反之亦然。

37. ports

映射端口

1. SHORT 语法

可以使用HOST:CONTAINER的方式指定端口，也可以指定容器端口（选择临时主机端口），宿主机随机映射端口

```yaml
ports:
- "3000"
- "3000-3005"
- "8000:8000"
- "9090-9091:8080-8081"
- "49100:22"
- "127.0.0.1:8001:8001"
- "127.0.0.1:5000-5010:5000-5010"
- "6060:6060/udp"
```

注意：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。

2. LONG 语法

LONG 语法支持 SHORT 语法不支持的附加字段

target：容器内的端口

published：公开的端口

protocol： 端口协议（tcp 或 udp）

mode：通过host 用在每个节点还是哪个发布的主机端口或使用 ingress 用于集群模式端口进行平衡载，

```
ports:
  - target: 80
    published: 8080
    protocol: tcp
    mode: host
```

38. secrets

通过 secrets为每个服务授予相应的访问权限

1. SHORT 语法

```
version: "3.1"
services:
  redis:
    image: redis:latest
    deploy:
      replicas: 1
    secrets:
      - my_secret
      - my_other_secret
secrets:
  my_secret:
    file: ./my_secret.txt
  my_other_secret:
    external: true
```

2.. LONG 语法

LONG 语法可以添加其他选项

source：secret 名称

target：在服务任务容器中需要装载在 /run/secrets/ 中的文件名称，如果 source 未定义，那么默为此值

uid&gid：在服务的任务容器中拥有该文件的 UID 或 GID 。如果未指定，两者都默认为 0。

mode：以八进制表示法将文件装载到服务的任务容器中 /run/secrets/ 的权限。例如，0444 代表可

。

```
version: "3.1"
services:

  redis:

    image: redis:latest

    deploy:

      replicas: 1

    secrets:

      - source: my_secret

        target: redis_secret

        uid: '103'

        gid: '103'

        mode: 0440

secrets:

  my_secret:

    file: ./my_secret.txt

  my_other_secret:

    external: true
```

39. 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 SER

```
security_opt:
  - label:user:USER
  - label:role:ROLE
```

40. stop_grace_period

在发送 SIGKILL 之前指定 stop_signal ，如果试图停止容器（如果它没有处理 SIGTERM（或指定的何停止信号）），则需要等待的时间

```
stop_grace_period: 1s
stop_grace_period: 1m30s
```

默认情况下，stop 在发送SIGKILL之前等待10秒钟容器退出

41. stop_signal

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

```
stop_signal: SIGUSR1
```

42. sysctls

在容器中设置的内核参数，可以为数组或字典

```
sysctls:
  net.core.somaxconn: 1024
  net.ipv4.tcp_syncookies: 0

sysctls:
  - net.core.somaxconn=1024
  - net.ipv4.tcp_syncookies=0
```

43. ulimits

覆盖容器的默认限制，可以单一地将限制值设为一个整数，也可以将soft/hard 限制指定为映射

```
ulimits:
  nproc: 65535
  nofile:
    soft: 20000
    hard: 40000
```

44. userns_mode

```
userns_mode: "host"
```

45. 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OST: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文件系统

```
version: "3.2"
services:
  web:
    image: nginx:alpine
    volumes:
      - type: volume
        source: mydata
        target: /data
        volume:
          nocopy: true
      - type: bind
        source: ./static
        target: /opt/app/static
```

```
  db:
    image: postgres:latest
    volumes:
      - "/var/run/postgres/postgres.sock:/var/run/postgres/postgres.sock"
      - "dbdata:/var/lib/postgresql/data"

volumes:
  mydata:
  dbdata:
```

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
```

如果你不使用宿主机的路径，可以指定一个 volume_driver

```
volume_driver: mydriver
```

1. SHORT 语法

可以选择在主机（HOST:CONTAINER）或访问模式（HOST:CONTAINER:ro）上指定路径。

可以在主机上挂载相对路径，该路径相对于正在使用的 Compose 配置文件的目录进行扩展。相对路应始终以.或 ..开头

```
volumes:
  # Just specify a path and let the Engine create a volume
  - /var/lib/mysql
  # Specify an absolute path mapping
  - /opt/data:/var/lib/mysql
  # Path on the host, relative to the Compose file
  - ./cache:/tmp/cache
  # User-relative path
  - ~/configs:/etc/configs/:ro
  # Named volume
  - datavolume:/var/lib/mysql
```

2. LONG 语法

LONG 语法有些附加字段

type：安装类型，可以为 volume、bind 或 tmpfs

source：安装源，主机上用于绑定安装的路径或定义在顶级 volumes密钥中卷的名称 ,不适用于 tmpfs 类型安装。

target：卷安装在容器中的路径

read_only：标志将卷设置为只读

bind：配置额外的绑定选项

propagation：用于绑定的传播模式

volume：配置额外的音量选项

nocopy：创建卷时禁止从容器复制数据的标志

tmpfs：配置额外的 tmpfs 选项

size：tmpfs 的大小，以字节为单位

```
version: "3.2"
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - type: volume
        source: mydata
        target: /data
        volume:
          nocopy: true
      - type: bind
        source: ./static
        target: /opt/app/static

networks:
  webnet:

volumes:
  mydata:
```

46. volumes_from

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

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

47. 用于服务、群集以及堆栈文件的卷

在使用服务，群集和 docker-stack.yml 文件时，请记住支持服务的任务（容器）可以部署在群集中任何节点上，并且每次更新服务时都可能是不同的节点。

在缺少指定源的命名卷的情况下，Docker 为支持服务的每个任务创建一个匿名卷。关联的容器被移后，匿名卷不会保留。

如果希望数据持久存在，请使用可识别多主机的命名卷和卷驱动程序，以便可以从任何节点访问数据或者，对该服务设置约束，以便将其任务部署在具有该卷的节点上。

下面一个例子，Docker Labs 中 votingapp 示例的 docker-stack.yml文件中定义了一个称为 db 的务。它被配置为一个命名卷来保存群体上的数据， 并且仅限于在节点上运行。下面是来自该文件的部内容：db postgres manager

```
version: "3"
services:
  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]
```

48. restart

默认值为 no ，即在任何情况下都不会重新启动容器；当值为 always 时，容器总是重新启动；当值为 n-failure 时，当出现 on-failure 报错容器退出时，容器重新启动。

```
restart: "no"
restart: always
restart: on-failure
restart: unless-stopped
```

49. 其他选项

关于标签：cpu_shares、cpu_quota、 cpuse、domainname、hostname、 ipc、 mac_address、rivileged、 read_only、 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

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

user: postgresql
working_dir: /code

domainname: [foo.com](http://foo.com/)
hostname: foo
```

```
ipc: host
mac_address: 02:42:ac:11:65:43

privileged: true


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

50. 持续时间

某些配置选项如 check 的子选项interval以及timeout 的设置格式

```
2.5s
10s
1m30s
2h32m
5h34m56s
```

支持的单位有 us、ms、s、m 以及 h

51. 指定字节值

某些选项如bulid的子选项 shm_size

```
2b
1024kb
2048k
300m
1gb
```

支持的单位是 b，k，m 以及 g，或 kb， mb 和 gb。目前不支持十进制值

52. 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，相同服务情况下，后来者会有选择地覆盖原有配置

```
extends:
  file: common.yml
  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。

详细的文档可以查看官方文档：撰写 docker-compose.yml 文件参考

学习参考：Docker Compose 配置文件详解