docker-compose编排参数详解

一、前言

Compose是一个用于定义和运行多容器Docker应用程序的工具。使用Compose，您可以使用YAML文件来配置应用程序的服务。然后，使用单个命令，您可以从配置中创建并启动所有服务。

Compose可以.yml 或 .yaml 作为文件扩展名。

Compose适用于所有环境：生产，登台，开发，测试以及CI工作流程。

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 的官方安装教程 Install Docker Compose

二、Docker Compose 配置文件的构建参数说明

2.1 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。

build 可以指定为包含构建上下文路径的字符串：

```
version:"3.7"
services:
    webapp:
        build:./dir
```

或者，作为具有在上下文中指定的路径的对象以及可选的Dockerfile和args：

```
version:"3.7"
services:
    webapp:
        build:
            context:./dir
            dockerfile:Dockerfile-alternate
            args:
                buildno:1
```

如果您指定image以及build，则使用以下指定的webapp和可选项对构建的图像进行命名：

这会在./dir目录下生成一个名为webapp和标记的图像tag。

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

> 注意: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。

2.2 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 context。

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

2.3 dockerfile

使用此 dockerfile 文件来构建，必须指定构建路径。

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

2.4 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 gitcommithash

RUN echo "Build number: $buildno"
RUN echo "Based on commit: $gitcommithash"
```

然后在build键下指定参数，可以传递映射或列表：

```
build:
    context:.
    args:
        buildno:1
        gitcommithash:cdc3b19
build:
    context:.
    args:
        -buildno=1
        -gitcommithash=cdc3b19
```

> 注意：在Dockerfile中，如果ARG在FROM指令之前指定， ARG则在构建说明中不可用FROM。如果您需要在两个位置都可以使用参数，请在FROM指令下指定它。

您可以在指定构建参数时省略该值，在这种情况下，它在构建时的值是运行Compose的环境中的值。

```
args:
    -buildno
    -gitcommithash
```

> 注意：YAML布尔值（true，false，yes，no，on，off）必须用引号括起来，这样分析器会将它们解释为字符串。

2.5 cache_from

编写缓存解析镜像列表，此选项是v3.2中的新选项。

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

2.6 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，此选项是v3.3中的新选项。

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

2.7 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'
build:
    context:.
    shm_size:10000000
```

2.8 target

根据对应的 Dockerfile 构建指定 Stage

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

2.9 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
```

> 注意: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

2.10 cgroup_parent

可以为容器选择一个可选的父 cgroup_parent

```
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

2.11. 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，方式类似于dockerfile：

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

2.12. 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

2.12.1 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，源名称和目标装入点都设为配置名称。

```
version:"3.7"
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_config 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 命令或通过另一个堆栈部署。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。

> 注意: 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。

2.12.2 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。默认是 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，它将被忽略。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:"3.7"
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。定义配置并不意味着授予服务访问权限。

2.13 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
```

由于Docker容器名称必须是唯一的，因此如果指定了自定义名称，则无法将服务扩展到1个容器之外。试图这样做会导致错误。

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

2.14 credential_spec

> 注意：此选项已在v3.3中添加。在Compose版本3.8中支持使用具有撰写文件的组托管服务帐户（GMSA）配置。

配置托管服务帐户的凭据规范。此选项仅用于使用Windows容器的服务。

使用 file时：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
```

示例GMSA配置

为服务配置GMSA凭据规范时，只需指定凭据规范config，如以下示例所示：

```
version: "3.8"
services:
   myservice:
      image: myimage:latest
      credential_spec:
          config: my_credential_spec

configs:
   my_credentials_spec:
      file: ./my-credential-spec.json
```

2.15 depends_on

容器中服务之间的依赖关系，依赖关系会导致以下行为：

- docker-compose up以依赖顺序启动服务。在以下示例中，db并redis在之前启动web。
- docker-compose up SERVICE自动包含SERVICE依赖项。在以下示例中，docker-compose up web还创建并启动db和redis。
- docker-compose stop按依赖顺序停止服务。在以下示例中，web在db和之前停止redis

```
version:"3.7"
services:
    web:
        build:.
        depends_on:
            -db
            -redis
    redis:
        image:redis
    db:
        image:postgres
```

使用时需要注意以下几点depends_on：

depends_on不会在启动web之前等待db和redis“就绪”

版本3不再支持condition形式depends_on。

使用版本3 Compose文件在swarm模式下部署堆栈depends_on时，将忽略该选项 。

2.16 deploy

指定与部署和运行服务相关的配置。

```
version:"3.7"
services:
    redis:
        image:redis:alpine
```

```
        deploy:
            replicas:6
            update_config:
                parallelism:2
                delay:10s
            restart_policy:
                condition:on-failure
```

有几个子选项可供选择：

2.16.1 endpoint_mode

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“ 部位用于访问网络上的服务。

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。如果想使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）功能就非常实用。

```
version:"3.7"

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:
```

2.16.2 labels

指定服务的标签，这些标签仅在服务上设置

```
version:"3.7"
services:
    web:
        image:web
        deploy:
            labels:
                com.example.description:"Thislabelwillappearonthewebservice"
```

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

```
version:"3.7"
services:
    web:
        image:web
        labels:
            com.example.description:"Thislabelwillappearonallcontainersforthewebservice"
```

2.16.3 mode

global:每个集节点只有一个容器

replicated:指定容器数量（默认）

```
version:"3.7"
services:
    worker:
        image:dockersamples/examplevotingapp_worker
        deploy:
            mode:global
```

2.16.4 placement

指定 constraints 和 preferences

```
version:"3.7"
services:
    db:
        image:postgres
        deploy:
            placement:
                constraints:
                    -node.role == manager
                    -engine.labels.operatingsystem == ubuntu 14.04
                preferences:
                    -spread:node.labels.zone
```

2.16.5 replicas

如果服务是 replicated（默认），需要指定运行的容器数量

```
version:"3.7"
services:
```

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

2.16.6 resources

配置资源限制

```
version:"3.7"
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），并且保留 20M 了内存和 0.25 CPU时间。

2.16.7 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成功 window，则此尝试不计入配置max_attempts 值。例如，如果 max_attempts 值为 2，并且第一次尝试重新启动失败，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）。

```
version:"3.7"
services:
    redis:
        image:redis:alpine
        deploy:
            restart_policy:
                condition:on-failure
                delay:5s
                max_attempts:3
                window:120s
```

2.16.8 rollback_config

配置在更新失败的情况下应如何回滚服务。

parallelism：一次回滚的容器数。如果设置为0，则所有容器同时回滚。
delay：每个容器组的回滚之间等待的时间（默认为0）。
failure_action：如果回滚失败该怎么办。一个continue或pause（默认pause）
monitor：每次更新任务后的持续时间以监视失败(ns|us|ms|s|m|h)（默认为0）。
max_failure_ratio：回滚期间容忍的失败率（默认为0）。
order：回滚期间的操作顺序。其中一个stop-first（旧任务在启动新任务之前停止），或者start-first（首先启动新任务，并且正在运行的任务暂时重叠）（默认stop-first）。

2.16.9 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（新的任务首先启动，并且正在运行的任务短暂重叠）（默认 stop-first）

```
version:"3.7"
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

2.17 devices

设置映射列表，与 Docker 客户端的 --device 参数类似

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

2.18 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
```

2.19 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
```

2.20 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 中可以定义接入点，覆盖 Dockerfile 中的定义

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

入口点也可以是一个列表，方式类似于 dckerfile：

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

2.21 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所在的目录

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
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

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

a.env 文件

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

b.env文件

```
# b.env
VAR=hello
```

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 之后，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 arg 标签。

2.22 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。

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

2.23 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

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

2.24 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 links 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 能够连接这些不在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_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）。

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

2.25 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 类似。

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

在/etc/hosts此服务的内部容器中创建具有ip地址和主机名的条目，例如：

```
162.242.195.82  somehost
50.31.209.229   otherhost
```

2.26 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 ；如果它是一个字符串，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"]
```

如上所述，但包裹在内/bin/sh。以下两种形式都是等同的。

```
test:["CMD-SHELL","curl-fhttp://localhost||exit1"]
test:curl -f https://localhost || exit 1
```

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
```

2.27 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。

如果镜像不存在，Compose 会自动拉去镜像。

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

2.28 init

在容器内运行init，转发信号并重新获得进程。将此选项设置true是为服务启用此功能。

```
version:"3.7"
services:
    web:
        image:alpine:latest
        init:true
```

2.29 isolation

指定容器的隔离技术。在Linux上，唯一支持的值是default。在Windows中，可接受的值是default，process和 hyperv。

2.30 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
```

2.31 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小存储数量

```
driver: "json-file"
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

```
version:"3.7"
services:
    some-service:
        image:some-service
        logging:
            driver:"json-file"
            options:
                max-size:"200k"
                max-file:"10"
```

2.32 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]，可以指定使用服务或者容器的网络。

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

2.33 networks

加入指定网络

```
services:
    some-service:
        networks:
          -some-network
          -other-network
```

2.34 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，相同的服务可以在不同的网络有不同的别名。

```
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:"3.7"

services:
    web:
        image:"nginx:alpine"
        networks:
            -new

    worker:
        image:"my-worker-image:latest"
        networks:
            -legacy

    db:
        image:mysql
        networks:
            new:
                aliases:
                    -database
            legacy:
                aliases:
                    -mysql
```

```
networks:
    new:
    legacy:
```

2.35 ipv4_address、ipv6_address

为服务的容器指定一个静态 IP 地址

```
version:"3.7"

services:
    app:
        image:nginx:alpine
        networks:
            app_net:
                ipv4_address:172.16.238.10
                ipv6_address:2001:3984:3989::10

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

2.36 pid

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。使用此标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

```
pid: "host"
```

2.37 ports

映射端口

2.37.1 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），宿主机会随机映射端口

```
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。

2.37.2 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负载

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

2.38 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 on-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
```

2.40 secrets

通过 secrets为每个服务授予相应的访问权限

2.40.1 SHORT 语法

```
version:"3.7"
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
```

2.40.2 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，那么默认为此值

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 代表可读。

```
version:"3.7"
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
```

2.41 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

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

2.42 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（或指定的任何停止信号）），则需要等待的时间

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

2.43 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
```

2.44 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
```

注意：使用（版本3）Compose文件在群集模式下部署堆栈时，将忽略此选项

2.45 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
```

在容器内安装临时文件系统。Size参数指定tmpfs mount的大小（以字节为单位）。默认无限制。

```
- type: tmpfs
    target: /app
    tmpfs:
      size: 1000
```

注意：使用（版本3-3.5）Compose文件在群集模式下部署堆栈时，将忽略此选项 。

2.46 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
```

2.47 userns_mode

如果Docker守护程序配置了用户名称空间，则禁用此服务的用户名称空间。

```
userns_mode: "host"
```

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

2.48 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

```
version: "3.7"
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:
```

此示例显示服务使用的命名卷（mydata）web以及为单个服务（db服务 下的第一个路径volumes）定义的绑定安装。该db服务还使用名为dbdata（db服务中的第二个路径volumes）的命名卷，但使用旧字符串格式定义它以安装命名卷。必须在顶级volumes键下列出命名卷

2.48.1 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。相对路径应始终以 . 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
```

2.48.2 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：配置其他卷选项

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

consistent：完全一致。容器运行时和主机始终保持相同的安装视图。这是默认值。
cached：主机的mount视图是权威的。在主机上进行的更新在容器中可见之前可能会有延迟。
delegated：容器运行时的mount视图是权威的。在容器中进行的更新在主机上可见之前可能会有延迟。

```
version: "3.7"
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:
```

2.49 driver

指定应为此卷使用哪个卷驱动程序。默认为Docker Engine配置使用的任何驱动程序，在大多数情况下是local。如果驱动程序不可用，则在docker-compose up尝试创建卷时Engine会返回错误 。

```
driver: foobar
```

2.50 driver_opts

将选项列表指定为键值对，以传递给此卷的驱动程序。

```
volumes:
  example:
    driver_opts:
      type: "nfs"
      o: "addr=10.40.0.199,nolock,soft,rw"
      device: ":/docker/example"
```

2.51 external

如果设置为true，则指定已在Compose之外创建此卷。docker-compose up不会尝试创建它，如果它不存在则引发错误。

在下面的示例中，[projectname]_dataCompose 不是尝试创建一个被调用的卷，而是 查找简单调用的现有卷data并将其挂载到db服务的容器中。

```
version: "3.7"
services:
```

```
  db:
    image: postgres
    volumes:
      - data:/var/lib/postgresql/data
volumes:
  data:
    external: true
```

还可以在Compose文件中与用于引用它的名称分别指定卷的名称：

```
volumes:
  data:
    external:
      name: actual-name-of-volume
```

2.52 name

为此卷设置自定义名称。name字段可用于引用包含特殊字符的卷。该名称按原样使用，不会使用堆栈名称作为范围。

```
version: "3.7"
volumes:
  data:
    name: my-app-data
```

它也可以与external参数一起使用：

```
version: "3.7"
volumes:
  data:
    external: true
    name: my-app-data
```