示例：

- Your first docker-compose.yml File
- Add a new service and redeploy

build

在构建时应用的配置选项。

`build` 可以指定为包含构建上下文路径的字符串：

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

也可以是，在上下文和可选的的Dockerfile and args中被指定的包含路径的对象:

```
version: '3'
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
```

如果您同时指定`image`和`build`，则使用以下指定的`webapp`和可选项tag对构建的镜像进行命名：

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

这将从./dir目录创建一个名字为`webapp` ，标签为tag的镜像

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 。该`docker stack`命令仅接受预先构建好的镜像。

CONTEXT

包含Dockerfile的目录的路径，或者是git存储库的url。

当提供的值是相对路径时，它被解释为相对于Compose文件的位置。此目录也是发送给Docker守护程序的构建上下文。

compose使用生成的名字构建并标记镜像，然后使用该镜像。

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

DOCKERFILE

备用Dockerfile。

Compose使用备用文件来构建。还必须指定构建路径。

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

ARGS

添加构建参数，这些参数只能在构建过程中访问。

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。您可以传递字典或列表：

```
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， 则ARG在FROM下的build指令中不可用，如果您需要在两个位置都可以使用ARG，请在`FROM`指令下指定它。See Understand how ARGS and FROM interact for usage details.

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：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。

image

指定容器启动的镜像，可以是repository/tag 或者十部分镜像ID。

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

如果图像不存在，则Compose尝试拉取它，除非您还指定了build，在这种情况下，它使用指定的选项构建它并使用指定的标记对其进行标记。

depends_on

表述服务之间的依赖关系，服务之间的依赖导致以下行为：

- `docker-compose up`以依赖顺序启动服务。在以下示例中，`db`和`redis`在`web`之前启动。
- `docker-compose up SERVICE`自动包含`SERVICE`的依赖服务。在以下示例中，`docker-compose up web`同时也创建并启动了`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`redis`之前停止。

简单的例子：

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

使用depends_on时需要注意以下几点：

- depends_on并不意味着启动web前只是等待db和redis “ready”,而是要他们已经“start”。如果您需要等待服务准备就绪，请参阅控制启动顺序以 获取有关此问题的更多信息以及解决此问题的策略。
- 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 。

expose

暴露端口而不将它们发布到主机 - 它们只能被链接服务访问。只能指定内部端口。

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

links

警告：该--link标志是Docker的遗留功能。它最终可能被删除。除非您绝对需要继续使用它，否则我们建议您使用user-defined networks来促进两个容器之间的通信，而不是使用--link。user-defined networks不支持但是 --link支持的一个功能是在容器之间共享环境变量。但是，您可以使用其他机制（如卷）以更可控的方式在容器之间共享环境变量。

链接到另一个服务中的容器。指定服务名称和链接别名（SERVICE:ALIAS），或仅指定服务名称。

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

被链接的容器可以通过与别名相同的主机名访问，如果未指定别名，则可以访问服务名称。

启用服务进行通信不需要链接 - 默认情况下，任何服务都可以通过该服务的名称访问任何其他服务。（另请参阅 Links topic in Networking in Compose。）

链接还以与depends_on相同的方式表示服务之间的依赖关系 ，因此它们确定服务启动的顺序。

注意

- 如果同时定义链接和网络，则它们之间具有链接的服务必须共享至少一个共同的网络才能进行通信。
- This option is ignored when deploying a stack in swarm mode with a (version 3) Compose file.

ports

暴露端口。

注意：端口映射与network_mode: host不兼容

SHORT SYNTAX

指定ports（`HOST:CONTAINER`）或仅指定容器端口（选择临时的主机端口）

注意：以`HOST:CONTAINER`格式映射端口时，使用低于60的容器端口时可能会遇到错误的结果，因为YAML会将格式`xx:yy`中的数字解析为base-60值。因此，我们建议始终将端口映射明确指定为字符串。

```
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"
```

LONG SYNTAX

长格式语法允许配置无法以简短形式表示的其他字段。

- `target`：容器内的端口
- `published`：公开暴露的港口
- `protocol`：端口协议（`tcp`或`udp`）
- `mode`：`host`-用于在每个节点上发布主机端口，或者`ingress`-用于群集模式端口实现负载平衡。

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

注意：长语法是v3.2中的新增功能

command

覆盖默认命令。

```
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"]
```

entrypoint

覆盖默认入口点。

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

入口点也可以是一个列表，方式类似于 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
```

注意：设置`entrypoint`都会覆盖默认的 由包含`ENTRYPOINT` 指令的Dockerfile 生成的镜像的 `entrypoint设置`，并 清除镜像上的任何默认命令 - 这意味着如果Dockerfile中有`CMD` 指令，则会被忽略。

也会忽略command命令。

restart

no是默认的重新启动策略，它不会在任何情况下重新启动容器。当always指定时，容器总是重新启动。 on-failure是在容器的退出码表明了一个故障错误时，重启容器。

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

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 。请改用restart_policy。

name

在3.4版文件格式中添加

为此卷设置自定义名称。name字段的使用参照volumes ，包含特殊的字符。该名称按原样使用，不受堆栈名称的限制。

```
version: '3.4'
volumes:
  data:
    name: my-app-data
```

它也可以与external一起使用：

```
version: '3.4'
volumes:
  data:
    external: true
    name: my-app-data
```

volumes

挂载主机路径或命名卷，指定为服务的子选项。

您可以将主机路径作为单个服务的定义的一部分进行安装，而无需在顶级volumes键中定义它。

但是，如果要跨多个服务重用卷，请在顶级volumes键中定义命名卷。Use named volumes with services, swarms, and stack files.

Note: top-level volumes 的key定义了一个named volume，并且从每个服务的volumes列表中引用它；在早期的Compose 文件格式中，这将替换volumes_from 。有关卷的一般信息，请参阅Use volumes和Volume Plugins。

这个示例展示了一个named volume (mydata) 正在被web 服务使用，和为单个服务定义的绑定挂载(first path under db service volumes)。db 服务也使用了一个名叫dbdata （second path under db service volumes）的volume,但使用旧的字符串格式定义它以安装命名卷. Named volumes 必须在 top-level volumes下列出, as shown.

```
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:
```

SHORT SYNTAX

（可选）指定主机（`HOST:CONTAINER`）上的路径或访问模式（`HOST:CONTAINER:ro`）。

您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。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
```

LONG SYNTAX

长格式语法允许配置无法以简短形式表示的其他字段。

- `type`：安装类型`volume`，`bind`或`tmpfs`
- `source`：mount的源，主机上用于绑定挂载的路径，或顶级`volumes`键中定义的卷的名称。不适用于tmpfs挂载。
- `target`：容器中安装卷的路径
- `read_only`：将卷设置为只读的标识
- `bind`：配置其他绑定选项
  - `propagation`：用于绑定的传播模式
- `volume`：配置其他卷选项
  - `nocopy`：用于在创建卷时禁用从容器复制数据的标识
- `tmpfs`：配置其他tmpfs选项
  - `size`：tmpfs mount的大小（以bytes字节为单位）
- `consistency`：mount的一致性要求，包含`consistent`（主机和容器具有相同的视图），`cached`（读缓存，主机视图是权威的）或`delegated`（读写缓存，容器的视图是权威的）

```
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:
```

注意：长语法是v3.2中的新增功能

Volume configuration reference

虽然可以在文件上作为服务声明的一部分声明卷，但本部分允许您创建可以跨多个服务重用的**命名卷 named volumes**（不依赖`volumes_from`），并且可以使用docker命令行和API轻松恢复和检查。有关更多信息，请参阅 docker volume子命令文档。

这是一个双服务设置的示例，其中数据库的数据目录与另一个服务作为卷共享，以便可以定期备份它：

```
version: "3"

services:
  db:
    image: db
    volumes:
      - data-volume:/var/lib/db
  backup:
    image: backup-service
    volumes:
      - data-volume:/var/lib/backup/data

volumes:
  data-volume:
```

顶级`volumes`键下的条目可以为空，在这种情况下，它使用引擎配置的默认驱动程序（在大多数情况下，这是`local`驱动程序）。（可选）您可以使用以下键进行配置：

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
```

driver_opts

将选项列表指定为键值对，以传递给此卷的驱动程序。这些选项取决于驱动程序 - 有关详细信息，请参阅驱动程序的文档。可选的。

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

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。

为3.3和下面的格式的版本，`external`可以不与其它卷配置键（`driver`，`driver_opts`，`labels`）结合使用。版本3.4及更高版本不再存在此限制。

在下面的示例中，Compose 不是尝试创建一个叫`[projectname]_data`的卷，而是 查找一个名为data的现有卷，并将其挂载到db服务的容器中。

```
version: '3'

services:
  db:
    image: postgres
    volumes:
      - data:/var/lib/postgresql/data

volumes:
  data:
```

```
  external: true
```

不支持在版本3.4文件格式中 使用external.name`name`。

您还可以在Compose文件中， 使用 用来指代它的名称分别指定卷的名称：

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

始终使用docker stack deploy创建外部卷

如果使用docker stack deploy以群集模式启动应用程序（而不是docker compose up），则会创建不存在的外部卷。在群集模式下，当服务定义卷时，会自动创建卷。由于服务任务是在新节点上调度的，因此 swarmkit会在本地节点上创建卷。要了解更多信息，请参阅moby / moby＃29976。

VOLUMES FOR SERVICES, SWARMS, AND STACK FILES

使用服务，群组和`docker-stack.yml`文件时，请记住，支持服务的任务（容器）可以部署在群中的任何节点上，每次更新服务时，这可能是不同的节点。

如果没有指定源的named volumes，Docker会为支持服务的每个任务创建一个匿名卷。删除关联的容器后，匿名卷不会保留。

如果你希望数据持久化，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，对服务设置约束，以便将其任务部署在具有卷的节点上。

例如， votingapp sample in Docker Labs 的docker-stack.yml定义了一个服务叫db，运行postgres数据库，它被配置为命名卷（named volume）以在swarm上保留数据， 并且仅限于在`manager`节点上运行。以下是该文件中的相关剪辑：

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

CACHING OPTIONS FOR VOLUME MOUNTS (DOCKER DESKTOP FOR MAC)

On Docker 17.04 CE Edge and up, including 17.06 CE Edge and Stable, you can configure container-and-host consistency requirements for bind-mounted directories in Compose files to allow for better performance on read/write of volume mounts. These options address issues specific to `osxfs` file sharing, and therefore are only applicable on Docker Desktop for Mac.

The flags are:

- `consistent`: Full consistency. The container runtime and the host maintain an identical view of the mount at all times. This is the default.
- `cached`: The host's view of the mount is authoritative. There may be delays before updates made on the host are visible within a container.
- `delegated`: The container runtime's view of the mount is authoritative. There may be delays before updates made in a container are visible on the host.

Here is an example of configuring a volume as `cached`:

```
version: '3'
services:
  php:
    image: php:7.1-fpm
    ports:
      - "9000"
    volumes:
```

```
    - .:/var/www/project:cached
```

Full detail on these flags, the problems they solve, and their `docker run` counterparts is in the Docker Desktop for Mac topic Performance tuning for volume mounts (shared filesystems).

network_mode

网络模式。使用与docker client `--network`参数相同的值，加上特殊表单`service:[service name]`。

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

Notes

- This option is ignored when deploying a stack in swarm mode with a (version 3) Compose file.
- `network_mode: "host"` cannot be mixed with links.

networks

Networks to join, referencing entries under the top-level `networks` key.

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

ALIASES

网络上此服务的别名（备用主机名）。同一网络上的其他容器可以使用服务名称或此别名连接到其中一个服务的容器。

由于`aliases`是网络范围的，因此相同的服务可以在不同的网络上具有不同的别名。

注意：网络范围的别名可以由多个容器共享，甚至可以由多个服务共享。如果这样使用，则无法保证名称解析为哪个容器。

一般格式如下所示。

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

在下面的例子中，提供了三种服务（`web`，`worker`，和`db`），和两个网络（`new` and `legacy`）。在`new`网络上，该`db`服务可以通过主机名`db`或`database被访问`，在`legacy`网络上，可以通过`db`或`mysql被访问`。

```
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:
```

IPV4_ADDRESS，IPV6_ADDRESS

在加入网络时为此服务指定容器的静态IP地址。

在top-level networks section对应的网络设置中必须存在一个`ipam`块，包含覆盖每个静态地址的子网设置。如果需要IPv6寻址，`enable_ipv6`选项必须存在，并且必须使用2.X版本的Compose文件，如下所示。

注意：这些选项目前不适用于群集模式。

An example:

```
version: '2.1'

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
```

logging

service的日志配置

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

该`driver`名称指定服务容器的日志记录驱动程序，与docker run的`--log-driver`选项一样(documented here)。

默认值为json-file。

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

注意：只有 json-file和journald 可以直接从docker-compose up和docker-compose logs中获取日志。 使用任何其他驱动程序不会打印任何日志。

通过options 为logging driver指定日志选项，与docker run的--log-opt选项一样。

Logging选项是键值对。 syslog选项的一个例子：

```
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。为此，请使用键值对来获得最大存储大小和最大文件数：

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

上面显示的示例将存储日志文件，直到它们达到max-size200kB，然后转存它们。存储的各个日志文件的数量由max-file值指定。随着日志超出最大限制，将删除较旧的日志文件以允许存储新日志。

以下是docker-compose.yml限制日志记录存储的示例文件：

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

可用的日志选项取决于您使用的日志记录驱动程序

上面用于控制日志文件和大小的示例使用特定于json-file driver选项。其他日志记录驱动程序不提供这些特定选项。有关支持的日志记录驱动及其选项的完整列表，请参阅logging drivers.

env_file

从文件添加环境变量。可以是单个值或列表。

如果已通过docker-compose -f FILE指定Compose文件，则env_file中的路径是相对于该文件所在的目录。

在environment部分中 声明的环境变量会覆盖这些值 - 即使这些值为空或未定义，这也适用。

```
env_file: .env
```

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

Compose期望env文件中的每一行都是VAR=VAL格式化的。以#开头的行被视为注释并被忽略。空行也被忽略。

```
# Set Rails/Rack environment
RACK_ENV=development
```

注意：如果您的服务指定了 build 选项，则在构建期间，环境文件中定义的变量不会自动显示。使用build的子选项args定义构建时环境变量。

VAL的值按原样使用，根本不进行修改。 例如，如果值由引号括起（通常是shell变量的情况），则引号包含在传递给Compose的值中。

请记住，列表中文件的顺序对于确定分配给多次显示的变量的值非常重要。列表中的文件从上到下进行处理。对于文件a.env中指定的相同变量，并在文件b.env中指定了不同的值，如果b.env列在下面（后面），则取b.env中的值。例如，在docker-compose.yml中给出以下声明：

```
services:
```

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

And the following files:

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

and

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

`$VAR` is `hello`.

DNS

自定义DNS服务器。可以是单个值或列表。

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

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
```

external_links

链接到外部容器，这些容器不是由docker-compose.yml启动，甚至不属于Compose，尤其是提供共享或公共服务的容器。在指定容器名称和链接别名（CONTAINER：ALIAS）时，external_links遵循与`links` 类似的语义。

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

注意：

如果您使用的是版本2或更高版本的文件格式，则外部创建的容器必须至少连接到与链接它们的服务所在网络相同的网络之一。Links 是一种传统选择。我们建议使用 networks。

在群集模式下 使用（版本3）Compose文件部署堆栈时，将忽略此选项。

extra_hosts

添加主机名映射。使用与docker client `--add-host`参数相同的值。

```
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
```

healthcheck

版本2.1文件格式及以上。

配置运行的检查以确定此服务的容器是否“健康”。有关 healthchecks如何工作的详细信息，请参阅HEALTHCHECK Dockerfile指令的文档 。

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

`interval`，`timeout`和`start_period`指定为durations.

注意：`start_period`仅支持v3.4及更高版本的撰写文件格式。

`test`必须是字符串或列表。如果是列表，则第一项必须是`NONE`，`CMD`或者`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 -f http://localhost || exit 1"]
```

```
test: curl -f https://localhost || exit 1
```

要禁用镜像设置的任何默认healthcheck ，您可以使用`disable: true`。这相当于指定`test: ["NONE"]`。

```
healthcheck:
  disable: true
```

pid

```
pid: "host"
```

将PID模式设置为主机PID模式。这打开了容器和主机操作系统之间的PID地址空间共享。使用此标志启动的容器可以访问和操作裸机计算机命名空间中的其他容器，反之亦然。

deploy

仅限第3版。

指定与部署和运行服务相关的配置。这只在部署 swarm with docker stack deploy时生效，并且被`docker-compose up` and `docker-compose run`忽略。

```
version: '3'
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：

ENDPOINT_MODE

为连接到群集的外部客户端指定服务发现方法。

仅限3.3版。

- `endpoint_mode: vip` - Docker为服务分配虚拟IP（VIP），作为客户端到达网络服务的前端。Docker在客户端和服务的可用工作节点之间路由请求，而无需客户端知道有多少节点参与服务或其IP地址或端口。（这是默认设置。）
- `endpoint_mode: dnsrr` - DNS循环（DNSRR）服务发现不使用单个虚拟IP。Docker为服务设置DNS条目，以便服务名称的DNS查询返回IP地址列表，并且客户端直接连接到其中一个。在您要使用自己的负载均衡器或混合Windows和Linux应用程序的情况下，DNS循环法非常有用。

```
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:
```

其余参数见：https://docs.docker.com/compose/compose-file/#deploy

environment

您可以使用数组或字典添加环境变量。任何布尔值; true，false，yes no，需要用引号括起来，以确保YML解析器不会将它们转换为True或False。

仅具有key的环境变量将解析为计算机正在运行的计算机上的值，这对于保密或特定于主机的值很有用。

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

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

注意：如果您的服务指定了 build 选项，则在构建期间，环境文件中定义的变量不会自动显示。使用build的子选项args定义构建时环境变量。

CACHE_FROM

注意：此选项是v3.2中的新选项

A list of images that the engine uses for cache resolution.

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

LABELS

注意：此选项是v3.2中的新选项

使用Docker标签将元数据添加到生成的图像中。您可以使用数组或字典。

我们建议您使用反向DNS表示法来防止您的标签与其他软件使用的标签冲突。

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

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

SHM_SIZE

在3.5版文件格式中添加

设置此构建容器的/dev/shm分区大小。指定为表示字节数的整数值或表示字节值的字符串。

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

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

TARGET

在3.4版文件格式中添加

按照Dockerfile内部定义构建指定的阶段. See the multi-stage build docs for details.

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

cap_add, cap_drop

添加或删除容器功能。 See `man 7 capabilities` for a full list.

```
cap_add:
  - ALL
```

```
cap_drop:
  - NET_ADMIN
  - SYS_ADMIN
```

这些选项在 deploying a stack in swarm mode 的compose文件（版本3）中被忽略

cgroup_parent

为容器指定可选的父cgroup。

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

这些选项在 deploying a stack in swarm mode 的compose文件（版本3）中被忽略

configs

Grant access to configs on a per-service basis using the per-service `configs` configuration. Two different syntax variants are supported.

Note: The config must already exist or be defined in the top-level `configs` configuration of this stack file, or stack deployment fails.

For more information on configs, see configs.

SHORT SYNTAX

short syntax variant 仅指定配置名称，这将授予容器对配置的访问权限并将其挂载到容器内部目录`/<config_name>`。源名称和目标挂载点都设置为配置名称。

以下示例使用短语法授予`redis`对`my_config`和`my_other_config`configs 的服务访问权限。`my_config` 的值被设置为./my_config.txt的文件内容，`my_other_config` 被定义为外部资源，意味着它已经在docker中通过docker config create命令，或者被另一个 stack deployment定义过，如果外部配置不存在，则stack deployment失败并显示`config not found`错误。

注意：`config`定义仅在compose文件格式的3.3及更高版本中受支持。

```
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
```

LONG SYNTAX

long syntax在 service's task containers中，关于配置是如何被创建的 提供了更细粒度。

- `source`：Docker中存在的配置名称。
- `target`： service's task containers中文件挂载的路径和名称。如果未指定，则默认为`/<source>`。
- `uid`和`gid`：在service's task containers中，拥有已挂载配置文件的数字UID或GID。在Linux如果未指定，则默认为`0`。Windows不支持。
- `mode`：以八进制表示的在 service's task containers中挂载的文件的权限。例如，`0444` 代表世界可读。默认是`0444`。配置无法写入，因为它们安装在临时文件系统中，因此如果设置了可写位，则会将其忽略。可以设置可执行位。如果您不熟悉UNIX文件权限模式，则可能会发现此 permissions calculator 很有用。

以下示例在容器中设置`my_config`到`redis_config`，将模式设置为`0440`（group-readable）并将用户和组设置为`103`。该`redis`服务无权访问该`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
```

您可以授予对多个配置的服务访问权限，您可以混合使用长短语法。定义配置并不意味着授予服务访问权限。

CONTAINER_NAME

指定自定义容器名称，而不是生成的默认名称。

```
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 。

credential_spec

注意：此选项已在v3.3中添加。

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。credential_spec的格式必须是`file://<filename>` or `registry://<value-name>`.

当使用file时，引用文件必须存在于Docker的数据目录CredentialSpecs的子目录，在windows中默认为C:\ProgramData\Docker\；以下示例，从名为C:\ProgramData\Docker\CredentialSpecs\my-credential-spec.json的文件中加载credential spec：

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

使用时`registry:`，将从守护程序主机上的Windows注册表中读取凭据规范。具有给定名称的注册表值必须位于：

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

以下示例从注册表中名为`my-credential-spec` 中加载凭据规范：

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

devices

设备映射列表。使用与docker client create选项`--device`相同的格式。

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

init

在3.7版文件格式中添加。

在容器内运行init，转发信号并重新获得进程。设置布尔值以使用默认值`init`，或指定自定义的路径。

```yaml
version: '3.7'
services:
  web:
    image: alpine:latest
    init: true
```

```yaml
version: '2.2'
services:
  web:
    image: alpine:latest
    init: /usr/libexec/docker-init
```

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`。 有关详细信息，请参阅 Docker Engine文档。

domainname，hostname，ipc，mac_address，privileged，read_only，shm_size，stdin_open，tty，user，working_dir

每个都是单个值，功能类似于 他们在docker run命令中对应的功能。请注意，这`mac_address`是一个遗留选项。

```yaml
user: postgresql
working_dir: /code

domainname: 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
```

Specifying durations

某些配置选项（例如`interval`和`check`的子选项`timeout`）接受字符串作为持续时间，格式如下所示：

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

支持的单位是`us`，`ms`，`s`，`m`和`h`。

secrets

Grant access to secrets on a per-service basis using the per-service `secrets` configuration. Two different syntax variants are supported.

Note: The secret must already exist or be defined in the top-level `secrets` configuration of this stack file, or stack deployment fails.

For more information on secrets, see secrets.

SHORT SYNTAX

The short syntax variant only specifies the secret name. This grants the container access to the secret and mounts it at `/run/secrets/<secret_name>` within the container. The source name and destination mountpoint are both set to the secret name.

The following example uses the short syntax to grant the `redis` service access to the `my_secret` and `my_other_secret` secrets. The value of `my_secret` is set to the contents of the file `./my_secret.txt`, and `my_other_secret` is defined as an external resource, which means that it has already been defined in Docker, either by running the `docker secret create` command or by another stack deployment. If the external secret does not exist, the stack deployment fails with a `secret not found` error.

```
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
```

LONG SYNTAX

The long syntax provides more granularity in how the secret is created within the service's task containers.

- `source`: The name of the secret as it exists in Docker.
- `target`: The name of the file to be mounted in `/run/secrets/` in the service's task containers. Defaults to `source` if not specified.
- `uid` and `gid`: The numeric UID or GID that owns the file within `/run/secrets/` in the service's task containers. Both default to `0` if not specified.
- `mode`: The permissions for the file to be mounted in `/run/secrets/` in the service's task containers, in octal notation. For instance, `0444` represents world-readable. The default in Docker 1.13.1 is `0000`, but is be `0444` in newer versions. Secrets cannot be writable because they are mounted in a temporary filesystem, so if you set the writable bit, it is ignored. The executable bit can be set. If you aren't familiar with UNIX file permission modes, you may find this permissions calculator useful.

The following example sets name of the `my_secret` to `redis_secret` within the container, sets the mode to `0440` (group-readable) and sets the user and group to `103`. The `redis` service does not have access to the `my_other_secret` secret.

```
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
```

You can grant a service access to multiple secrets and you can mix long and short syntax. Defining a secret does not imply granting a service access to it.

security_opt

Override the default labeling scheme for each container.

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

Note: This option is ignored when deploying a stack in swarm mode with a (version 3) Compose file.

stop_grace_period

Specify how long to wait when attempting to stop a container if it doesn't handle SIGTERM (or whatever stop signal has been specified with `stop_signal`), before sending SIGKILL. Specified as a duration.

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

By default, `stop` waits 10 seconds for the container to exit before sending SIGKILL.

stop_signal

Sets an alternative signal to stop the container. By default `stop` uses SIGTERM. Setting an alternative signal using `stop_signal` causes`stop` to send that signal instead.

```
stop_signal: SIGUSR1
```

sysctls

Kernel parameters to set in the container. You can use either an array or a dictionary.

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

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

Note: This option is ignored when deploying a stack in swarm mode with a (version 3) Compose file.

tmpfs

Version 2 file format and up.

Mount a temporary file system inside the container. Can be a single value or a list.

```
tmpfs: /run
```

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

Note: This option is ignored when deploying a stack in swarm mode with a (version 3-3.5) Compose file.

Version 3.6 file format and up.

Mount a temporary file system inside the container. Size parameter specifies the size of the tmpfs mount in bytes. Unlimited by default.

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

ulimits

Override the default ulimits for a container. You can either specify a single limit as an integer or soft/hard limits as a mapping.

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

userns_mode

```
userns_mode: "host"
```

Disables the user namespace for this service, if Docker daemon is configured with user namespaces. See dockerd for more information.

Note: This option is ignored when deploying a stack in swarm mode with a (version 3) Compose file.

Network configuration reference

The top-level `networks` key lets you specify networks to be created.

- For a full explanation of Compose's use of Docker networking features and all network driver options, see the Networking guide.
- For Docker Labs tutorials on networking, start with Designing Scalable, Portable Docker Container Networks

driver

Specify which driver should be used for this network.

The default driver depends on how the Docker Engine you're using is configured, but in most instances it is `bridge` on a single host and `overlay` on a Swarm.

The Docker Engine returns an error if the driver is not available.

```
driver: overlay
```

BRIDGE

Docker defaults to using a `bridge` network on a single host. For examples of how to work with bridge networks, see the Docker Labs tutorial on Bridge networking.

OVERLAY

The `overlay` driver creates a named network across multiple nodes in a swarm.

- For a working example of how to build and use an `overlay` network with a service in swarm mode, see the Docker Labs tutorial on Overlay networking and service discovery.
- For an in-depth look at how it works under the hood, see the networking concepts lab on the Overlay Driver Network Architecture.

HOST OR NONE

Use the host's networking stack, or no networking. Equivalent to `docker run --net=host` or `docker run --net=none`. Only used if you use `docker stack` commands. If you use the `docker-compose` command, use network_mode instead.

The syntax for using built-in networks like `host` and `none` is a little different. Define an external network with the name `host` or `none` (that Docker has already created automatically) and an alias that Compose can use (`hostnet` or `nonet` in these examples), then grant the service access to that network, using the alias.

```
version: '3.7'
services:
  web:
    # ...
    networks:
```

```
      hostnet: {}

networks:
  hostnet:
    external: true
    name: host
```

```
services:
  web:
    # ...
    networks:
      nonet: {}

networks:
  nonet:
    external: true
    name: none
```