Compose file reference

Estimated reading time: 32 minutes

The Compose file is a YAML file defining services, networks and volumes. The default path for a Compose file is `./docker-compose.yml`.

> **Tip:** You can use either a `.yml` or `.yaml` extension for this file. They both work.

A service definition contains configuration which will be applied to each container started for that service, much like passing command-line parameters to `docker run`. Likewise, network and volume definitions are analogous to `docker network create` and `docker volume create`.

As with `docker run`, options specified in the Dockerfile (e.g., `CMD`, `EXPOSE`, `VOLUME`, `ENV`) are respected by default - you don't need to specify them again in `docker-compose.yml`.

You can use environment variables in configuration values with a Bash-like `${VARIABLE}` syntax - see variable substitution for full details.

Service configuration reference

> **Note:** There are several versions of the Compose file format – version 1 (the legacy format, which does not support volumes or networks) and version 2, as well as 2.1 (the most up-to-date). For more information, see the Versioning section.

This section contains a list of all configuration options supported by a service definition.

build

Configuration options that are applied at build time.

`build` can be specified either as a string containing a path to the build context, or an object with the path specified under context and optionally dockerfile and args.

```
build: ./dir

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

If you specify `image` as well as `build`, then Compose names the built image with the `webapp` and optional `tag` specified in `image`:

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

This will result in an image named `webapp` and tagged `tag`, built from `./dir`.

> **Note**: In the version 1 file format, `build` is different in two ways:
>
> 1. Only the string form (`build: .`) is allowed - not the object form.
> 2. Using `build` together with `image` is not allowed. Attempting to do so results in an error.

context

> Version 2 file format and up. In version 1, just use build.

Either a path to a directory containing a Dockerfile, or a url to a git repository.

When the value supplied is a relative path, it is interpreted as relative to the location of the Compose file. This directory is also the build context that is sent to the Docker daemon.

Compose will build and tag it with a generated name, and use that image thereafter.

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

dockerfile

Alternate Dockerfile.

Compose will use an alternate file to build with. A build path must also be specified.

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

> **Note**: In the version 1 file format, `dockerfile` is different in two ways:
>
> 1. It appears alongside `build`, not as a sub-option:
>
>    ```
>    build: .
>    dockerfile: Dockerfile-alternate
>    ```
>
> 2. Using `dockerfile` together with `image` is not allowed. Attempting to do so results in an error.

args

> Version 2 file format and up.

Add build arguments, which are environment variables accessible only during the build process.

First, specify the arguments in your Dockerfile:

```
ARG buildno
ARG password

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

Then specify the arguments under the `build` key. You can pass either a mapping or a list:

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

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

You can omit the value when specifying a build argument, in which case its value at build time is the value in the environment where Compose is running.

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

> **Note**: YAML boolean values (`true`, `false`, `yes`, `no`, `on`, `off`) must be enclosed in quotes, so that the parser interprets them as strings.

cap_add, cap_drop

Add or drop container capabilities. See `man 7 capabilities` for a full list.

```
cap_add:
  - ALL

cap_drop:
  - NET_ADMIN
  - SYS_ADMIN
```

command

Override the default command.

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

The command can also be a list, in a manner similar to dockerfile:

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

cgroup_parent

Specify an optional parent cgroup for the container.

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

container_name

Specify a custom container name, rather than a generated default name.

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

Because Docker container names must be unique, you cannot scale a

service beyond 1 container if you have specified a custom name. Attempting to do so results in an error.

devices

List of device mappings. Uses the same format as the `--device` docker client create option.

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

depends_on

Express dependency between services, which has two effects:

- `docker-compose up` will start services in dependency order. In the following example, `db` and `redis` will be started before `web`.
- `docker-compose up SERVICE` will automatically include `SERVICE`'s dependencies. In the following example, `docker-compose up web` will also create and start `db` and `redis`.

Simple example:

```
version: '2'
services:
  web:
    build: .
    depends_on:
      - db
      - redis
  redis:
    image: redis
  db:
    image: postgres
```

> **Note:** `depends_on` will not wait for `db` and `redis` to be "ready" before

> starting `web` - only until they have been started. If you need to wait for a service to be ready, see Controlling startup order for more on this problem and strategies for solving it.

dns

Custom DNS servers. Can be a single value or a list.

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

dns_search

Custom DNS search domains. Can be a single value or a list.

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

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
```

entrypoint

Override the default entrypoint.

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

The entrypoint can also be a list, in a manner similar to dockerfile:

```
entrypoint:
    - php
    - -d
    - zend_extension=/usr/local/lib/php/extensions/no-debug-non-zts-2010052
    - -d
    - memory_limit=-1
    - vendor/bin/phpunit
```

> **Note:** Setting `entrypoint` will both override any default entrypoint set on the service's image with the `ENTRYPOINT` Dockerfile instruction, *and* clear out any default command on the image - meaning that if there's a `CMD` instruction in the Dockerfile, it will be ignored.

env_file

Add environment variables from a file. Can be a single value or a list.

If you have specified a Compose file with `docker-compose -f FILE`, paths in `env_file` are relative to the directory that file is in.

Environment variables specified in `environment` override these values.

```
env_file: .env
```

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

Compose expects each line in an env file to be in `VAR=VAL` format. Lines

beginning with # (i.e. comments) are ignored, as are blank lines.

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

> **Note:** If your service specifies a build option, variables defined in environment files will *not* be automatically visible during the build. Use the args sub-option of `build` to define build-time environment variables.

The value of `VAL` is used as is and not modified at all. For example if the value is surrounded by quotes (as is often the case of shell variables), the quotes will be included in the value passed to Compose.

environment

Add environment variables. You can use either an array or a dictionary. Any boolean values; true, false, yes no, need to be enclosed in quotes to ensure they are not converted to True or False by the YML parser.

Environment variables with only a key are resolved to their values on the machine Compose is running on, which can be helpful for secret or host-specific values.

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

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

> **Note:** If your service specifies a build option, variables defined in `environment` will *not* be automatically visible during the build. Use the

> args sub-option of `build` to define build-time environment variables.

expose

Expose ports without publishing them to the host machine - they'll only be accessible to linked services. Only the internal port can be specified.

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

extends

Extend another service, in the current file or another, optionally overriding configuration.

You can use `extends` on any service together with other configuration keys. The `extends` value must be a dictionary defined with a required `service` and an optional `file` key.

```
extends:
  file: common.yml
  service: webapp
```

The `service` the name of the service being extended, for example `web` or `database`. The `file` is the location of a Compose configuration file defining that service.

If you omit the `file` Compose looks for the service configuration in the current file. The `file` value can be an absolute or relative path. If you specify a relative path, Compose treats it as relative to the location of the current file.

You can extend a service that itself extends another. You can extend indefinitely. Compose does not support circular references and `docker-`

`compose` returns an error if it encounters one.

For more on `extends`, see the the extends documentation.

external_links

Link to containers started outside this `docker-compose.yml` or even outside of Compose, especially for containers that provide shared or common services. `external_links` follow semantics similar to `links` when specifying both the container name and the link alias (`CONTAINER:ALIAS`).

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

> **Note:** If you're using the version 2 file format, the externally-created containers must be connected to at least one of the same networks as the service which is linking to them.

extra_hosts

Add hostname mappings. Use the same values as the docker client `--add-host` parameter.

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

An entry with the ip address and hostname will be created in `/etc/hosts` inside containers for this service, e.g:

```
162.242.195.82  somehost
50.31.209.229   otherhost
```

group_add

> Version 2 file format and up.

Specify additional groups (by name or number) which the user inside the container will be a member of. Groups must exist in both the container and the host system to be added. An example of where this is useful is when multiple containers (running as different users) need to all read or write the same file on the host system. That file can be owned by a group shared by all the containers, and specified in `group_add`. See the Docker documentation for more details.

A full example:

```
version: '2'
services:
    image: alpine
    group_add:
      - mail
```

Running `id` inside the created container will show that the user belongs to the `mail` group, which would not have been the case if `group_add` were not used.

image

Specify the image to start the container from. Can either be a repository/tag or a partial image ID.

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

If the image does not exist, Compose attempts to pull it, unless you have

also specified build, in which case it builds it using the specified options and tags it with the specified tag.

> **Note**: In the version 1 file format, using `build` together with `image` is not allowed. Attempting to do so results in an error.

isolation

> Added in version 2.1 file format.

Specify a container's isolation technology. On Linux, the only supported value is `default`. On Windows, acceptable values are `default`, `process` and `hyperv`. Refer to the Docker Engine docs for details.

labels

Add metadata to containers using Docker labels. You can use either an array or a dictionary.

It's recommended that you use reverse-DNS notation to prevent your labels from conflicting with those used by other software.

```
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"
```

links

Link to containers in another service. Either specify both the service name and a link alias (`SERVICE:ALIAS`), or just the service name.

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

Containers for the linked service will be reachable at a hostname identical to the alias, or the service name if no alias was specified.

Links also express dependency between services in the same way as depends_on, so they determine the order of service startup.

> **Note:** If you define both links and networks, services with links between them must share at least one network in common in order to communicate.

logging

> Version 2 file format and up. In version 1, use log_driver and log_opt.

Logging configuration for the service.

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

The `driver` name specifies a logging driver for the service's containers, as with the `--log-driver` option for docker run (documented here).

The default value is json-file.

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

> **Note:** Only the `json-file` and `journald` drivers make the logs available directly from `docker-compose up` and `docker-compose logs`. Using any other driver will not print any logs.

Specify logging options for the logging driver with the `options` key, as with the `--log-opt` option for `docker run`.

Logging options are key-value pairs. An example of `syslog` options:

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

log_driver

> Version 1 file format only. In version 2, use logging.

Specify a log driver. The default is `json-file`.

log_opt

> Version 1 file format only. In version 2, use logging.

Specify logging options as key-value pairs. An example of `syslog` options:

```
log_opt:
  syslog-address: "tcp://192.168.0.42:123"
```

net

> Version 1 file format only. In version 2, use network_mode.

Network mode. Use the same values as the docker client `--net` parameter. The `container:...` form can take a service name instead of a container name or id.

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

network_mode

> Version 2 file format and up. In version 1, use net.

Network mode. Use the same values as the docker client `--net` parameter, plus the special form `service:[service name]`.

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

networks

> Version 2 file format and up. In version 1, use net.

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

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

aliases

Aliases (alternative hostnames) for this service on the network. Other containers on the same network can use either the service name or this alias to connect to one of the service's containers.

Since `aliases` is network-scoped, the same service can have different

aliases on different networks.

> **Note**: A network-wide alias can be shared by multiple containers, and even by multiple services. If it is, then exactly which container the name will resolve to is not guaranteed.

The general format is shown here.

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

In the example below, three services are provided (`web`, `worker`, and `db`), along with two networks (`new` and `legacy`). The `db` service is reachable at the hostname `db` or `database` on the `new` network, and at `db` or `mysql` on the `legacy` network.

```
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

Specify a static IP address for containers for this service when joining the network.

The corresponding network configuration in the top-level networks section must have an `ipam` block with subnet and gateway configurations covering each static address. If IPv6 addressing is desired, the `enable_ipv6` option must be set.

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
        gateway: 172.16.238.1
      - subnet: 2001:3984:3989::/64
        gateway: 2001:3984:3989::1
```

link_local_ips

> Added in version 2.1 file format.

Specify a list of link-local IPs. Link-local IPs are special IPs which belong to a well known subnet and are purely managed by the operator, usually dependent on the architecture where they are deployed. Therefore they are not managed by docker (IPAM driver).

Example usage:

```
version: '2.1'
services:
  app:
    image: busybox
    command: top
    networks:
      app_net:
        link_local_ips:
          - 57.123.22.11
          - 57.123.22.13
networks:
  app_net:
    driver: bridge
```

pid

Sets the PID mode to the host PID mode. This turns on sharing between container and the host operating system the PID address space. Containers launched with this flag will be able to access and manipulate other containers in the bare-metal machine's namespace and vise-versa.

ports

Expose ports. Either specify both ports (`HOST:CONTAINER`), or just the container port (a random host port will be chosen).

> **Note:** When mapping ports in the `HOST:CONTAINER` format, you may experience erroneous results when using a container port lower than 60, because YAML will parse numbers in the format `xx:yy` as sexagesimal (base 60). For this reason, we recommend always explicitly specifying your port mappings as strings.

```
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"
```

security_opt

Override the default labeling scheme for each container.

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

stop_signal

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

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
```

volumes, volume_driver

Mount paths or named volumes, optionally specifying a path on the host machine (`HOST:CONTAINER`), or an access mode (`HOST:CONTAINER:ro`). For version 2 files, named volumes need to be specified with the top-level `volumes` key. When using version 1, the Docker Engine will create the named volume automatically if it doesn't exist.

You can mount a relative path on the host, which will expand relative to the directory of the Compose configuration file being used. Relative paths should always begin with `.` or `..`.

```
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
```

If you do not use a host path, you may specify a `volume_driver`.

```
volume_driver: mydriver
```

Note that for version 2 files, this driver will not apply to named volumes (you should use the `driver` option when declaring the volume instead). For version 1, both named volumes and container volumes will use the specified driver.

> Note: No path expansion will be done if you have also specified a `volume_driver`.

See Docker Volumes and Volume Plugins for more

volumes_from

Mount all of the volumes from another service or container, optionally specifying read-only access (`ro`) or read-write (`rw`). If no access level is specified, then read-write will be used.

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

> **Note:** The `container:...` formats are only supported in the version 2 file format. In version 1, you can use container names without marking them as such:

```
- service_name
- service_name:ro
- container_name
- container_name:rw
```

cpu_shares, cpu_quota, cpuset, domainname, hostname, ipc, mac_address, mem_limit, memswap_limit, oom_score_adj, privileged, read_only, restart, shm_size, stdin_open, tty, user, working_dir

Each of these is a single value, analogous to its docker run counterpart.

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

user: postgresql
working_dir: /code

domainname: foo.com
hostname: foo
ipc: host
mac_address: 02:42:ac:11:65:43

mem_limit: 1000000000
memswap_limit: 2000000000
privileged: true

oom_score_adj: 500

restart: always

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

> **Note:** The following options are only available for version 2 and up:
>
> - `oom_score_adj`

Volume configuration reference

While it is possible to declare volumes on the fly as part of the service declaration, this section allows you to create named volumes that can be reused across multiple services (without relying on `volumes_from`), and are easily retrieved and inspected using the docker command line or API. See the docker volume subcommand documentation for more information.

driver

Specify which volume driver should be used for this volume. Defaults to `local`. The Docker Engine will return an error if the driver is not available.

driver_opts

Specify a list of options as key-value pairs to pass to the driver for this volume. Those options are driver-dependent - consult the driver's documentation for more information. Optional.

```
 driver_opts:
   foo: "bar"
   baz: 1
```

external

If set to `true`, specifies that this volume has been created outside of Compose. `docker-compose up` will not attempt to create it, and will raise an error if it doesn't exist.

`external` cannot be used in conjunction with other volume configuration keys (`driver`, `driver_opts`).

In the example below, instead of attempting to create a volume called `[projectname]_data`, Compose will look for an existing volume simply called `data` and mount it into the `db` service's containers.

```
version: '2'

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

volumes:
  data:
    external: true
```

You can also specify the name of the volume separately from the name used to refer to it within the Compose file:

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

labels

> Added in version 2.1 file format.

Add metadata to containers using Docker labels. You can use either an array or a dictionary.

It's recommended that you use reverse-DNS notation to prevent your labels from conflicting with those used by other software.

```
labels:
  com.example.description: "Database volume"
  com.example.department: "IT/Ops"
  com.example.label-with-empty-value: ""

labels:
  - "com.example.description=Database volume"
  - "com.example.department=IT/Ops"
  - "com.example.label-with-empty-value"
```

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, see the Networking guide.

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 will be `bridge` on a single host and `overlay` on a Swarm.

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

driver_opts

Specify a list of options as key-value pairs to pass to the driver for this network. Those options are driver-dependent - consult the driver's documentation for more information. Optional.

```
driver_opts:
  foo: "bar"
  baz: 1
```

enable_ipv6

> Added in version 2.1 file format.

Enable IPv6 networking on this network.

ipam

Specify custom IPAM config. This is an object with several properties, each of which is optional:

- `driver`: Custom IPAM driver, instead of the default.
- `config`: A list with zero or more config blocks, each containing any of the following keys:
  - `subnet`: Subnet in CIDR format that represents a network segment
  - `ip_range`: Range of IPs from which to allocate container IPs

- `gateway`: IPv4 or IPv6 gateway for the master subnet
- `aux_addresses`: Auxiliary IPv4 or IPv6 addresses used by Network driver, as a mapping from hostname to IP

A full example:

```
ipam:
  driver: default
  config:
    - subnet: 172.28.0.0/16
      ip_range: 172.28.5.0/24
      gateway: 172.28.5.254
      aux_addresses:
        host1: 172.28.1.5
        host2: 172.28.1.6
        host3: 172.28.1.7
```

internal

> Version 2 file format and up.

By default, Docker also connects a bridge network to it to provide external connectivity. If you want to create an externally isolated overlay network, you can set this option to `true`.

labels

> Added in version 2.1 file format.

Add metadata to containers using Docker labels. You can use either an array or a dictionary.

It's recommended that you use reverse-DNS notation to prevent your labels from conflicting with those used by other software.

```
labels:
  com.example.description: "Financial transaction network"
```

```
  com.example.department: "Finance"
  com.example.label-with-empty-value: ""

labels:
  - "com.example.description=Financial transaction network"
  - "com.example.department=Finance"
  - "com.example.label-with-empty-value"
```

external

If set to `true`, specifies that this network has been created outside of Compose. `docker-compose up` will not attempt to create it, and will raise an error if it doesn't exist.

`external` cannot be used in conjunction with other network configuration keys (`driver`, `driver_opts`, `group_add`, `ipam`, `internal`).

In the example below, `proxy` is the gateway to the outside world. Instead of attempting to create a network called `[projectname]_outside`, Compose will look for an existing network simply called `outside` and connect the `proxy` service's containers to it.

```
version: '2'

services:
  proxy:
    build: ./proxy
    networks:
      - outside
      - default
  app:
    build: ./app
    networks:
      - default

networks:
  outside:
    external: true
```

You can also specify the name of the network separately from the name used to refer to it within the Compose file:

```
networks:
  outside:
    external:
      name: actual-name-of-network
```

Versioning

There are currently three versions of the Compose file format:

- Version 1, the legacy format. This is specified by omitting a `version` key at the root of the YAML.
- Version 2, the recommended format. This is specified with a `version: '2'` entry at the root of the YAML.
- Version 2.1, an upgrade over version 2 that takes advantage of the Docker Engine's newest features. Specify with a `version: '2.1'` entry at the root of the YAML.

To move your project from version 1 to 2, see the Upgrading section.

> **Note:** If you're using multiple Compose files or extending services, each file must be of the same version - you cannot mix version 1 and 2 in a single project.

Several things differ depending on which version you use:

- The structure and permitted configuration keys
- The minimum Docker Engine version you must be running
- Compose's behaviour with regards to networking

These differences are explained below.

Version 1

Compose files that do not declare a version are considered “version 1”. In those files, all the services are declared at the root of the document.

Version 1 is supported by **Compose up to 1.6.x**. It will be deprecated in a future Compose release.

Version 1 files cannot declare named volumes, networks or build arguments.

Example:

```
web:
  build: .
  ports:
   - "5000:5000"
  volumes:
   - .:/code
  links:
   - redis
redis:
  image: redis
```

Version 2

Compose files using the version 2 syntax must indicate the version number at the root of the document. All services must be declared under the `services` key.

Version 2 files are supported by **Compose 1.6.0+** and require a Docker Engine of version **1.10.0+**.

Named volumes can be declared under the `volumes` key, and networks can be declared under the `networks` key.

Simple example:

```
version: '2'
```

```
services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
  redis:
    image: redis
```

A more extended example, defining volumes and networks:

```
version: '2'
services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
    networks:
      - front-tier
      - back-tier
  redis:
    image: redis
    volumes:
      - redis-data:/var/lib/redis
    networks:
      - back-tier
volumes:
  redis-data:
    driver: local
networks:
  front-tier:
    driver: bridge
  back-tier:
    driver: bridge
```

Version 2.1

An upgrade of version 2 that introduces new parameters only available with Docker Engine version **1.12.0+**

Introduces the following additional parameters:

- `link_local_ips`
- `isolation`
- `labels` for volumes and networks

Upgrading

In the majority of cases, moving from version 1 to 2 is a very simple process:

1. Indent the whole file by one level and put a `services:` key at the top.
2. Add a `version: '2'` line at the top of the file.

It's more complicated if you're using particular configuration features:

- `dockerfile`: This now lives under the `build` key:

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

- `log_driver`, `log_opt`: These now live under the `logging` key:

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

- `links` with environment variables: As documented in the environment variables reference, environment variables created by links have been deprecated for some time. In the new Docker network system, they have been removed. You should either connect directly to the appropriate hostname or set the relevant environment variable yourself, using the link hostname:

```
web:
  links:
    - db
  environment:
    - DB_PORT=tcp://db:5432
```

- `external_links`: Compose uses Docker networks when running version 2 projects, so links behave slightly differently. In particular, two containers must be connected to at least one network in common in order to communicate, even if explicitly linked together.

  Either connect the external container to your app's default network, or connect both the external container and your service's containers to an external network.

- `net`: This is now replaced by network_mode:

```
net: host    ->  network_mode: host
net: bridge  ->  network_mode: bridge
net: none    ->  network_mode: none
```

  If you're using `net: "container:[service name]"`, you must now use `network_mode: "service:[service name]"` instead.

```
net: "container:web"  ->  network_mode: "service:web"
```

  If you're using `net: "container:[container name/id]"`, the value does not need to change.

```
net: "container:cont-name"  ->  network_mode: "container:cont-name"
net: "container:abc12345"   ->  network_mode: "container:abc12345"
```

- `volumes` with named volumes: these must now be explicitly declared in a top-level `volumes` section of your Compose file. If a service

mounts a named volume called `data`, you must declare a `data` volume in your top-level `volumes` section. The whole file might look like this:

```
version: '2'
services:
  db:
    image: postgres
    volumes:
      - data:/var/lib/postgresql/data
volumes:
  data: {}
```

By default, Compose creates a volume whose name is prefixed with your project name. If you want it to just be called `data`, declare it as external:

```
volumes:
  data:
    external: true
```

Variable substitution

Your configuration options can contain environment variables. Compose uses the variable values from the shell environment in which `docker-compose` is run. For example, suppose the shell contains `EXTERNAL_PORT=8000` and you supply this configuration:

```
web:
  build: .
  ports:
    - "${EXTERNAL_PORT}:5000"
```

When you run `docker-compose up` with this configuration, Compose looks for the `EXTERNAL_PORT` environment variable in the shell and substitutes its value in. In this example, Compose resolves the port mapping to

`"8000:5000"` before creating the `web` container.

If an environment variable is not set, Compose substitutes with an empty string. In the example above, if `EXTERNAL_PORT` is not set, the value for the port mapping is `:5000` (which is of course an invalid port mapping, and will result in an error when attempting to create the container).

You can set default values for environment variables using a `.env` file, which Compose will automatically look for. Values set in the shell environment will override those set in the `.env` file.

```
$ unset EXTERNAL_PORT
$ echo "EXTERNAL_PORT=6000" > .env
$ docker-compose up          # EXTERNAL_PORT will be 6000
$ export EXTERNAL_PORT=7000
$ docker-compose up          # EXTERNAL_PORT will be 7000
```

Both `$VARIABLE` and `${VARIABLE}` syntax are supported. Additionally when using the 2.1 file format, it is possible to provide inline default values using typical shell syntax:

- `${VARIABLE:-default}` will evaluate to `default` if `VARIABLE` is unset or empty in the environment.
- `${VARIABLE-default}` will evaluate to `default` only if `VARIABLE` is unset in the environment.

Other extended shell-style features, such as `${VARIABLE/foo/bar}`, are not supported.

You can use a `$$` (double-dollar sign) when your configuration needs a literal dollar sign. This also prevents Compose from interpolating a value, so a `$$` allows you to refer to environment variables that you don't want processed by Compose.

```
web:
```

```
build: .
command: "$$VAR_NOT_INTERPOLATED_BY_COMPOSE"
```

If you forget and use a single dollar sign (`$`), Compose interprets the value as an environment variable and will warn you:

The VAR_NOT_INTERPOLATED_BY_COMPOSE is not set. Substituting an empty string.

Compose documentation

- User guide
- Installing Compose
- Get started with Django
- Get started with Rails
- Get started with WordPress
- Command line reference

Feedback? Suggestions? Can't find something in the docs?
Edit this page ● Request docs changes ● Get support
Rate this page: