Docker Compose

Table of Contents

Prerequisites 3
Prerequisites to be installed in Linux 3
Docker Compose File Specifics 4
Creating Docker File 4
Creating Docker Compose File 4
Build and run Docker Image 5
Docker Compose Configuration Options 6
1. Build 6
2. Context 6
3. dockerfile 6
dockerfile: Dockerfile 4. args 6
5. cache_from 6
6. labels 7
7. network 7
8. target 7
9. command 8
10. configs 8
11. container_name 8
12. deploy 9
13. labels 9
14. resources 10
15. devices 10
16. entrypoint 10
17. env_file 10
18. environment 11
19. expose 11
19. healthcheck 11
20. image 11
21 init 12
22. logging 12
23. network_mode 12
24. networks 13
25. ports 14

26. profiles 14
27. secrets 15
28. volumes 16
28.1. external 17
29. Variable substitution 17

Prerequisites

Install Docker :
https://www.youtube.com/watch?v=_9AWYlt86B8

Download and Install Docker-compose here:
https://github.com/docker/compose/releases/

Install Cygwin
https://www.youtube.com/watch?v=QonIPpKodCw
https://github.com/lakelse/videos/tree/master/01-install-cygwin-on-windows-youtube
In Windows 10 you need to install Cygwin to run docker commands from the terminal

Prerequisites to be installed in Linux

Install Docker
https://linuxconfig.org/how-to-install-docker-on-ubuntu-20-04-lts-focal-fossa

Download and Install Docker-compose here:
https://github.com/docker/compose/releases/

In Linux you use the terminal to run docker commands using sudo.

Docker Compose File Specifics

It is important to know that the docker compose file is a YAML file and YAML files have a strict syntax
Just like the Python language the indent levels and depths are strict.
If you change that and you break the file
You can use JSON files but for the documentation ans the sake of this tutorial YML will be used for compatibility

Here is a Lab where JSON Docker Compose files ae used
https://dockerlabs.collabnix.com/intermediate/workshop/DockerCompose/Lab_%2324:_Use_JSON_instead_of_YAML_compose_file_in_Docker%3F.html

For an example of a docker compose file written for the Rabbit QM POC have a look at this page
https://github.com/nic0michael/RabbitMQProducerMicroservice/tree/master/DockerCompose

This will be considered outside the scope of this document

Creating Docker File

1. Create a folder: helloweb/

2. Create a folder: prod/
3. Inside this folder create a file create a HTML file : index.html with this content:

```
<html>
<body>
<h1>Hellow World</h1>
</body>
</html>
```

4. create a file called: Dockerfile with this content:

```
FROM centos:latest
MAINTAINER NewstarCorporation
RUN yum -y install httpd
COPY index.html /var/www/html/
CMD [“/usr/sbin/httpd”, “-D”, “FOREGROUND”]
EXPOSE 80
```

5. Navigate to the upper folder helloweb/

Creating Docker Compose File

6. Create a file called docker-compose.yml with this content:

```
version: '2'
services:
  prod:
```

```
image: helloweb
build:
  context: prod
  dockerfile: Dockerfile
ports:
  - 80: 80
restart: on-failure
```

Build and run Docker Image

7. open this folder in the Command Prompt run this command to build Docker Image:
docker-compose build

8. In the Command Prompt run this command to show docker images:
docker images

9. In the Command Prompt run this command to run docker image:
docker run -p 80:80 -it helloweb

I will now refer to the Docker compose documentation and provide a short summary of Docker Compose Configuration options to add to your docker Compos YAML file :
docker-compose.yml

Docker Compose Configuration Options

1. Build

can be specified either as a string containing a path to the build context:

```
build:
  context: prod
```

Here we have a Prod folder for the Production build

2. Context

3. dockerfile

Alternate Dockerfile.
Compose uses an alternate file to build with. A build path must also be specified.

dockerfile: Dockerfile

4. args

Add build arguments, which are environment variables accessible only during the build process.
First, specify the arguments in your Dockerfile:

```
ARG buildno
ARG gitcommithash

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

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

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

5. cache_from

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

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

6. labels

Added in version 3.3 file format

Add metadata to the resulting image 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.

```
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"
```

7. network

Added in version 3.4 file format

Set the network containers connect to for the RUN instructions during build.

```
build:
  context: .
  network: host
build:
  context: .
  network: custom_network_1
```

Use none to disable networking during build:

```
build:
  context: .
  network: none
```

8. target

Build the specified stage as defined inside the Dockerfile. See the multi-stage build docs for details.

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

9. 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"]

10. configs

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

Short Syntax

```
    configs:
      - my_config
      - my_other_config
configs:
  my_config:
    file: ./my_config.txt
  my_other_config:
    external: true
```

Long Syntax

```
    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
```

11. container_name

Specify a custom container name, rather than a generated default name.
container_name: my-web-container

12. deploy

Specify configuration related to the deployment and running of services. This only takes effect when deploying to a swarm with docker stack deploy, and is ignored by docker-compose up and docker-compose run.

```
version: "3.9"
services:
  redis:
    image: redis:alpine
    deploy:
      replicas: 6
      placement:
        max_replicas_per_node: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
```

13. labels

Specify labels for the service. These labels are only set on the service, and not on any containers for the service.

```
version: "3.9"
services:
  web:
    image: web
    deploy:
      labels:
        com.example.description: "This label will appear on the web service"
```

To set labels on containers instead, use the labels key outside of deploy:

```
version: "3.9"
services:
  web:
    image: web
    labels:
      com.example.description: "This label will appear on all containers for the web service"
```

14. resources

Configures resource constraints.

```
version: "3.9"
services:
  redis:
    image: redis:alpine
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
```

15. devices

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

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

16. 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", "memory_limit=-1", "vendor/bin/phpunit"]

17. 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 declared in the environment section override these values – this holds true even if those values are empty or undefined.

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

18. 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
```

19. 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"
```

19. healthcheck

Configure a check that's run to determine whether or not containers for this service are "healthy". See the docs for the HEALTHCHECK Dockerfile instruction for details on how healthchecks work.

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

20. image

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

```
image: redis

image: ubuntu:18.04

image: tutum/influxdb

image: example-registry.com:4000/postgresql

image: a4bc65fd
```

21 init

Run an init inside the container that forwards signals and reaps processes. Set this option to true to enable this feature for the service.

```
version: "3.9"
services:
  web:
    image: alpine:latest
    init: true
```

22. logging

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"
```

23. network_mode

Network mode. Use the same values as the docker client --network 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]"
```

24. networks

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

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

version: "3.9"
```

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.

```
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:
```

25. ports

To specify the mappings of the ports

Short syntax
There are three options:

1 Specify both ports (HOST:CONTAINER)
2 Specify just the container port (an ephemeral host port is chosen for the host port).
3 Specify the host IP address to bind to AND both ports (the default is 0.0.0.0, meaning all interfaces): (IPADDR:HOSTPORT:CONTAINERPORT). If HOSTPORT is empty (for example 127.0.0.1::80), an ephemeral port is chosen to bind to on the host.

```
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"
  - "127.0.0.1::5000"
  - "6060:6060/udp"
  - "12400-12500:1240"
```

Long Syntax
The long form syntax allows the configuration of additional fields that can't be expressed in the short form.

target: the port inside the container
published: the publicly exposed port
protocol: the port protocol (tcp or udp)
mode: host for publishing a host port on each node, or ingress for a swarm mode port to be load balanced.

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

26. profiles

profiles defines a list of named profiles for the service to be enabled under. When not set, the service is always enabled. For the services that make up your core application you should omit profiles so they will always be started.

Valid profile names follow the regex format [a-zA-Z0-9][a-zA-Z0-9_.-]+.

```
profiles: ["frontend", "debug"]
profiles:
  - frontend
  - debug
```

27. secrets

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

Short syntax

```
version: "3.9"
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

```
version: "3.9"
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
```

28. volumes

Mount host paths or named volumes, specified as sub-options to a service.

You can mount a host path as part of a definition for a single service, and there is no need to define it in the top level volumes key.

But, if you want to reuse a volume across multiple services, then define a named volume in the top-level volumes key. Use named volumes with

```
version: "3.9"
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

```
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

```
version: "3.9"
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:
```

28.1. external

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

```
version: "3.9"

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

volumes:
  data:
    external: true
```

29. Variable substitution

Your configuration options can contain **environment variables set on the server**. Compose uses the variable values from the shell environment in which docker-compose is run. For example, suppose the shell contains POSTGRES_VERSION=9.3 and you supply this configuration:

```
db:
  image: "postgres:${POSTGRES_VERSION}"
```