

Systems Advanced Docker Containers

Docker Compose



Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be



Docker Compose

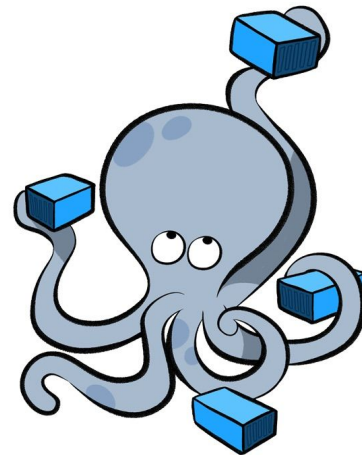
Compose is een tool voor het definiëren en uitvoeren van multi-container Docker applicaties. Met Compose gebruik je een YAML-bestand om de services van je applicatie te configureren. Vervolgens creëer en start je met één commando alle services uit je configuratie.

Compose werkt in alle omgevingen: productie, staging, ontwikkeling, testen, maar ook CI-workflows. Het heeft ook commando's voor het beheer van de hele levenscyclus van je applicatie:

- services starten, stoppen en herbouwen
- de status van running services bekijken
- de log output van running services streamen
- Een eenmalig commando uitvoeren op een service

De belangrijkste kenmerken van Compose die het effectief maken zijn:

- meerdere geïsoleerde omgevingen op een enkele host
- behoudt van volume data wanneer containers worden aangemaakt
- enkel containers die veranderd zijn worden aangemaakt
- ondersteunt variabelen en het verplaatsen van een compositie tussen omgevingen



Docker Compose installatie

Vanaf Docker Compose V2 is "compose" een Docker CLI sub-commando ("**docker compose**"), maar moet voorlopig op linux nog als aparte Docker CLI plugin geïnstalleerd worden, tenzij je Docker Desktop for Linux gebruikt.

Windows

De Docker Compose plugin wordt automatisch mee geïnstalleerd met Docker Desktop for Windows.

Linux

De Docker Compose plugin wordt automatisch mee geïnstalleerd met Docker Desktop for Linux.

Voor installatie van de Docker Compose plugin in de Ubuntu VM: [Overview | Docker Documentation](#).

Testen

docker compose version

```
# thraa @ DESKTOP-TOMC in ~ [16:03:51]
$ docker compose version
Docker Compose version v2.10.2

# thraa @ DESKTOP-TOMC in ~ [16:03:56]
$ _
```

Versieverwarring en The Compose Specification

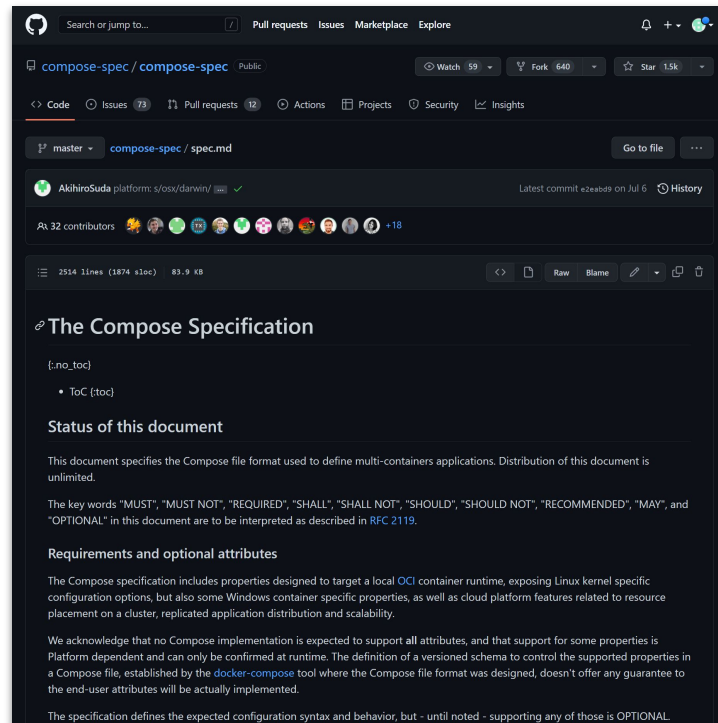
Online zal je nog veel (de meeste) voorbeelden en verwijzingen tegenkomen naar **Docker Compose V1**:

- met het **docker-compose** commando ipv **docker compose**
- **docker-compose.yml** ipv **compose.yml** als compose file
- compose files die beginnen met een **version: key**
 - **version: 2** # voor lokale deployment
 - **version: 3** # voor deployment naar 'Docker Swarm' - een early cloud container platform van Docker

Dit is deprecated en wordt allemaal uitgefaseerd vanaf Docker Compose V2.

De enige juiste Docker Compose documentatie heet vanaf nu de "The Compose Specification" en vind je hier op github:

<https://github.com/compose-spec/compose-spec/blob/master/spec.md>



Quick start

Docker Compose gebruiken is eigenlijk een proces in drie stappen:

1. Definieer de omgeving van je app met een Dockerfile zodat deze overal gereproduceerd kan worden.
2. Definieer de services waaruit je app bestaat in **compose.yml** zodat ze samen in een geïsoleerde omgeving kunnen draaien.
3. Voer tenslotte **docker compose up** uit en Compose zal je hele app starten en uitvoeren.

Compose kan deployen naar je lokale machine (default), maar is ook rechtstreeks **geïntegreerd met cloud platforms** zoals Amazon [Elastic Container Service](#) en Microsoft [Azure Container Instances](#). De opstap van je lokale machine naar de cloud is bijna onbestaand. (behalve voor je budget :^)

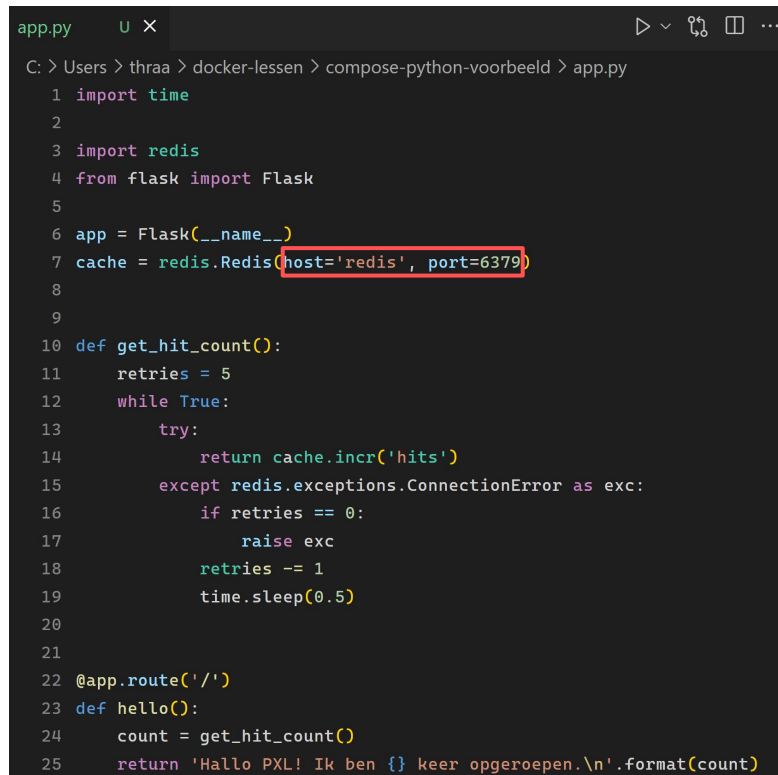
Een Compose voorbeeld - stap 1: de applicatie

We bouwen een simpele Python web app, die het Flask micro web framework gebruikt en die een hit counter bijhoudt in Redis.

Hiervoor gebruiken we de directory `compose-python-voorbeeld/` uit onze git repo <https://github.com/PXL-Systems-Advanced/docker-lessen.git>.

Stap 1: We bouwen (of ontvangen) een simpele web app in Python

- we bekijken de source code in `app.py`
- er wordt connectie gemaakt met een Redis instance met hostname "redis", op poort "6379"
- in `requirements.txt` staan de Python-specifieke dependencies die Python gebruikt om de app te bouwen



```
app.py  U X  ▶ ⌵ 🔍 □ ...
C: > Users > thraa > docker-lessen > compose-python-voorbeeld > app.py
1  import time
2
3  import redis
4  from flask import Flask
5
6  app = Flask(__name__)
7  cache = redis.Redis(host='redis', port=6379)
8
9
10 def get_hit_count():
11     retries = 5
12     while True:
13         try:
14             return cache.incr('hits')
15         except redis.exceptions.ConnectionError as exc:
16             if retries == 0:
17                 raise exc
18             retries -= 1
19             time.sleep(0.5)
20
21
22 @app.route('/')
23 def hello():
24     count = get_hit_count()
25     return 'Hallo PXL! Ik ben {} keer opgeroepen.\n'.format(count)
```

Een Compose voorbeeld - stap 2: maak een Dockerfile

De Dockerfile wordt gebruikt om een docker image te bouwen. De image bevat alle dependencies die de Python applicatie nodig heeft, inclusief Python zelf.

Bekijk de bestaande Dockerfile die het volgende doet:

- Bouw een image, beginnend met de Python 3.7 image.
- Stel de working directory in op /code.
- Stel environment variabelen in die gebruikt worden door het flask commando.
- Installeer gcc en andere dependencies
- Kopieer requirements.txt en installeer de Python dependencies.
- Voeg metadata toe aan de image om te beschrijven dat de container luistert op poort 5000
- Kopieer de huidige directory . in het project naar de workdir . in de image.
- Stel het standaard commando voor de container in op **flask run**.

Dockerfile U X

C: > Users > thraa > docker-lessen > compose-python-voorbeeld > D

```
1 FROM python:3.7-alpine
2 WORKDIR /code
3 ENV FLASK_APP=app.py
4 ENV FLASK_RUN_HOST=0.0.0.0
5 RUN apk add --no-cache gcc musl-dev linux-headers
6 COPY requirements.txt requirements.txt
7 RUN pip install -r requirements.txt
8 EXPOSE 5000
9 COPY . .
10 CMD ["flask", "run"]
```

Een Compose voorbeeld - stap 3: maak een Compose file

Bekijk de **compose.yml** file.

- Dit Compose-bestand definieert twee services: **web** en **redis**.
- De **web** service gebruikt een image die is opgebouwd uit de Dockerfile in de huidige directory. Het bindt dan de container en de host machine aan de exposed port, **8000**. Deze voorbeeld service gebruikt de standaardpoort voor de Flask webserver, **5000**.
- De **redis** service gebruikt een public [Redis image](#) uit de dockerhub registry.

compose.yml U X

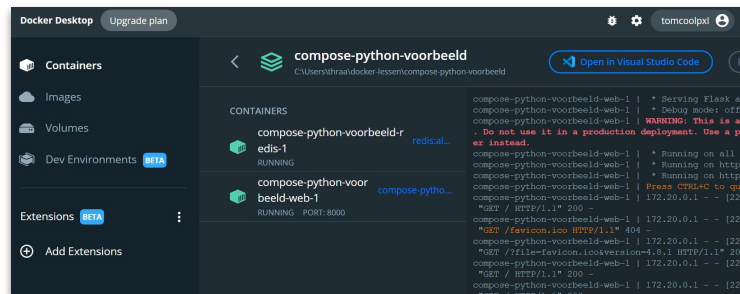
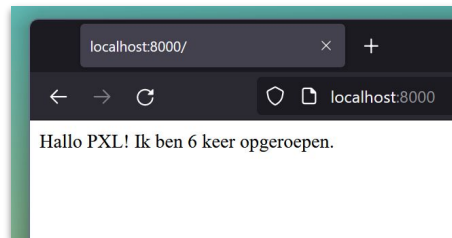
C: > Users > thraa > docker-lessen > cor

```
1 services:
2   web:
3     build: .
4     ports:
5       - "8000:5000"
6   redis:
7     image: "redis:alpine"
```


Een Compose voorbeeld - stap 4: build & run met Compose

docker compose up

- Compose haalt een Redis image op, bouwt een image voor je code en start de services die je gedefinieerd hebt. In dit geval wordt de code statisch gekopieerd naar de image tijdens het bouwen.
- Surf naar <http://localhost:8000/>
- Refresh een paar keer.
- Doe dat nog eens met een andere webbrowser.
- Typ **docker image ls** in een andere terminal.
- Check status met Docker Desktop.
- Stop de app met **docker compose down** in de 2de terminal of druk [CTRL]+[C] in de originele terminal.



Een Compose voorbeeld - stap 5: voeg een bind mount toe

Edit de `compose.yml` file

- Voeg de volgende regels toe in de `web:` service, onder de `ports:` sectie.

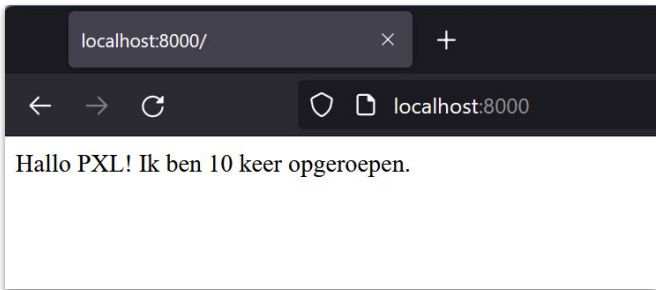
```
volumes:
  - ./code
environment:
  FLASK_DEBUG: True
```

- De nieuwe `volumes:` key koppelt de project directory (current dir) op de host aan `/code` in de container, zodat je de code on-the-fly kan wijzigen, zonder de image opnieuw te hoeven bouwen.
- De `environment:` key stelt de `FLASK_ENV` environment variable in, die `flask run` vertelt om in developer mode te runnen en de code opnieuw te laden bij elke verandering. Deze modus mag alleen worden gebruikt in development.
- Als je niet zeker bent over de verandering, spiek dan in `compose-2.yml`.

```
compose-2.yml U X
C: > Users > thraa > docker-lessen > cd
1  services:
2    web:
3      build: .
4      ports:
5        - "8000:5000"
6      volumes:
7        - ./code
8      environment:
9        FLASK_DEBUG: True
10   redis:
11     image: "redis:alpine"
```

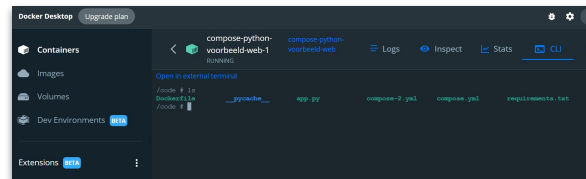
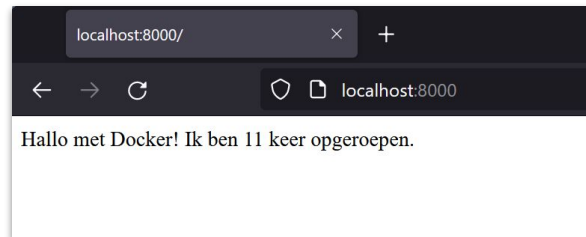
Een Compose voorbeeld - stap 6: rebuild & run

- Typ vanuit je project directory **docker compose up** om de app te bouwen met de bijgewerkte Compose file, en uit te voeren.
- Check de Hallo PXL boodschap opnieuw in een webbrowser, en refresh om de count te zien toenemen.



Een Compose voorbeeld - stap 7: update de app

- Omdat de applicatiecode nu in de container is gemount met behulp van een volume, kun je wijzigingen aanbrengen in de code en de wijzigingen direct zien, zonder de image opnieuw te hoeven bouwen.
- Gebruik Docker Desktop om een CLI te openen in de web-1 container. Typ `ls /code`
- Wijzig de boodschap in `app.py` op de host en sla op. Verander bijvoorbeeld de "Hallo PXL!" boodschap in "Hallo met Docker!"
- Refresh de page in je browser. De boodschap zou moeten worden bijgewerkt, en de count zou nog steeds moeten oplopen.



Een Compose voorbeeld - stap 8: andere commando's

- Als je je services op de achtergrond wil runnen, kan je de **-d** flag (voor "detached" modus) doorgeven aan **docker compose up** en **docker compose ps** gebruiken om te zien wat er momenteel draait.
 - docker compose up -d**
 - docker compose ps**
- Met het **docker compose run** commando kan je eenmalige commando's uitvoeren voor je services.
 - docker compose run web cat /etc/os-release**
- Zie **docker compose --help** voor andere beschikbare commando's.
- Als je Compose hebt gestart met **docker compose up -d**, stop dan je services zodra je ermee klaar bent:
 - docker compose stop**
- Je kan alles neerhalen, waarbij de containers volledig worden verwijderd, met het commando **down**. Geef **--volumes** door om ook het data volume dat door de Redis container wordt gebruikt te verwijderen:
 - docker compose down --volumes**

```
# thraa @ DESKTOP-TOMC in ~\docker-lesseen\compose-python-voorbeeld on git:main [00:08:24]
$ docker compose ps
NAME                                COMMAND                                SERVICE    STATUS    PORTS
compose-python-voorbeeld-redis-1    "docker-entrypoint.s..."            redis      running   6379/tcp
compose-python-voorbeeld-web-1      "flask run"                            web        running   0.0.0.0:8000->5000/tcp

# thraa @ DESKTOP-TOMC in ~\docker-lesseen\compose-python-voorbeeld on git:main [00:08:40]
$ docker compose run web cat /etc/os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.16.2
PRETTY_NAME="Alpine Linux v3.16"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://gitlab.alpinelinux.org/alpine/aports/-/issues"

# thraa @ DESKTOP-TOMC in ~\docker-lesseen\compose-python-voorbeeld on git:main [00:09:04]
$ docker compose down --volumes
[.] Running 3/3
- Container compose-python-voorbeeld-redis-1    Removed    0.5s
- Container compose-python-voorbeeld-web-1      Removed    0.7s
- Network compose-python-voorbeeld_default      Removed    0.2s

# thraa @ DESKTOP-TOMC in ~\docker-lesseen\compose-python-voorbeeld on git:main [00:09:23]
$
```

Veel gebruikte Compose commando's

<code>docker compose <CMD> <SERVICE></code>	als <SERVICE> wordt ingevuld geldt het commando voor die service, maar als het leeg is, geldt het voor alle services.
<code>docker compose build</code>	builden op basis van de compose.yml file
<code>docker compose up (-d)</code>	maakt de containers en eventuele netwerken aan en runt de containers (detached)
<code>docker compose down (--volume)</code>	stopt en verwijdert de containers, netwerken. --volume verwijdert ook de volumes.
<code>docker compose config</code>	toont de interne staat van de compose yaml file
<code>docker compose logs</code>	toont de logs van alle service containers die horen bij deze Docker Compose
<code>docker compose ps</code>	toont de service containers die horen bij deze Docker Compose
<code>docker compose start</code>	start de containers
<code>docker compose stop</code>	stopt de containers
<code>docker compose rm (--volume)</code>	verwijdert gestopte containers en met --volumes ook de anonymous volumes
<code>docker compose rm --stop (--volume)</code>	--stop stopt eerst de containers alvorens ze te verwijderen

Docker Compose drupal voorbeeld

Ga naar de directory `drupal-voorbeeld/` van de repo en bekijk de `compose.yml` file.

- 2 services: `drupal` en `postgres`, beide gebaseerd op "latest" version van official image in dockerhub
- allebei worden ge-restart wanneer ze down gaan/crashen/...
- `drupal`
 - port-forwarding 8080 op de host naar 80 in de container.
 - 3 bind mounts van lokale directories naar container directories
 - 1 anonymous volume `/var/www/html/sites`
- `postgres`
 - stelt environment variable in op "example" in de container
 - 1 bind mount van lokale directory `./postgres-data` naar directory in de container

Check dockerhub naar beide official images en vooral extra uitleg onderaan over hoe je die moet gebruiken.

`docker compose up -d`

surfen naar <http://localhost:8080>

`docker compose down -v`

`compose.yml` U X

C: > Users > thraa > docker-lessen > drupal-voorbeeld > `compose.yml`

```
1 services:
2   drupal:
3     image: drupal
4     ports:
5       - 8080:80
6     volumes:
7       - ./modules:/var/www/html/modules
8       - ./profiles:/var/www/html/profiles
9       - ./themes:/var/www/html/themes
10      - /var/www/html/sites
11     restart: always
12   postgres:
13     image: postgres
14     environment:
15       POSTGRES_PASSWORD: example
16     volumes:
17       - ./postgres-data:/var/lib/postgresql/data
18     restart: always
```

Docker Compose drupal voorbeeld

Drupal is een beetje een speciaal geval: Als je toch een host-volume wilt gebruiken voor de sites-map, dan dien je deze eerst voor te bereiden door die op te vullen tijdens een *eerste aparte run* van de drupal container. Zie dockerhub voor meer info.

```
docker run --rm drupal tar -cC  
/var/www/html/sites . | sudo tar -xC ./sites
```

compose-fixed.yml bevat de aangepaste versie

```
docker compose -f compose-fixed.yml up -d
```

```
docker compose config
```

surfen naar <http://localhost:8080>

```
docker compose down -v
```

```
compose.yml U  compose-fixed.yml U X  
C: > Users > thraa > docker-lessen > drupal-voorbeeld > compose-fixed.yml  
1  services:  
2    drupal:  
3      image: drupal  
4      ports:  
5        - 8080:80  
6      volumes:  
7        - ./modules:/var/www/html/modules  
8        - ./profiles:/var/www/html/profiles  
9        - ./themes:/var/www/html/themes  
10       - ./sites:/var/www/html/sites  
11      restart: always  
12  postgres:  
13    image: postgres  
14    environment:  
15      POSTGRES_PASSWORD: example  
16    volumes:  
17      - ./postgres-data:/var/lib/postgresql/data  
18    restart: always
```


Oefening Compose

- Maak een nieuwe directory **WordpressMariaDB** aan.
- Maak een nieuwe compose file aan en zorg dat er een WordPress-website draait met een Mariadb-database.
- Zorg er voor dat
 - de mariadb database files op de docker host staan.
 - de wordpress website kan geraadpleegd worden via poort 7654.
 - de website files op de docker host staan.
- Bij problemen
 - Opnieuw proberen nadat je de directories van de volumes hebt verwijderd.
 - Check met **docker compose ps** en **docker compose logs**.

Oefening Compose

github repo

<https://github.com/PXL-Systems-Advanced/docker-lessen.git>

```
cd ./compose-oef-1/
```

Opgave README.md:

<https://github.com/PXL-Systems-Advanced/docker-lessen/tree/main/compose-oef-1>

