7 Podman Compose

7.1 Inleiding

In dit hoofdstuk maak je kennis met Podman Compose. Dit is een hulpmiddel waarmee je meerdere containers tegelijk kunt starten en beheren met één configuratiebestand.

Een container kun je zien als een klein, geïsoleerd pakketje met software, bijvoorbeeld een database of een webserver. Vaak heb je niet één container nodig, maar meerdere die met elkaar samenwerken. Denk aan vorig hoofdstuk:

- Een database (MariaDB) waarin je gegevens opslaat.
- Een webapplicatie (WordPress) die de gegevens gebruikt.

Zonder Podman Compose moet je al deze containers handmatig starten met lange commando's zoals je gezien hebt in vorig hoofdstuk. Met Compose beschrijf je alles in een YAML-bestand. Dat bestand bevat:

- Welke containers je nodig hebt.
- Welke instellingen en wachtwoorden ze gebruiken.
- Welke mappen of volumes gegevens moeten opslaan.
- Hoe de containers met elkaar communiceren via netwerken.

> YML is de afkorting van YAML Ain't Markup Language, wat aangeeft dat YAML geen opmaaktaal is. Het is een formaat dat vooral wordt gebruikt voor het opslaan van configuratiegegevens op een eenvoudige en leesbare manier. YAML-bestanden hebben de extensie .yml of .yaml.

Daarna start je de hele set met één enkel commando: podman-compose up. Dit maakt het leven van een systeembeheerder veel makkelijker en het is ook makkelijk om er later op terug te vallen.

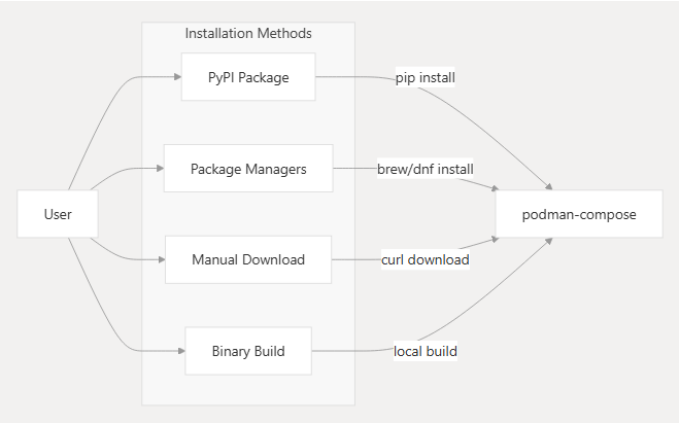
Podman-compose gebruikt dezelfde Compose-spec (het YAML-formaat) als Docker Compose. Het verschil zit alleen in de engine erachter:

- Docker Compose start containers via de Docker-daemon.
- Podman Compose start containers via Podman (daemonless, eventueel rootless).

7.2 Installatie Podman Compose

We installeren podman-compose op ServerXX.

Podman compose is geschreven in Python. Je kan podman-compose op meerdere manieren, zoals vele pakketten trouwens, installeren.



Wij gaan gebruik maken van pip3. Dat is ook de aangewezen methode. Met pip3 (pakketbeheerder voor Python 3) kan je op een eenvoudige manier podman-compose installeren.

Installeer hiervoor eerst pip3.

```
student@serverXX:~$ sudo dnf install -y python3-pip
```

Daarna installeer je via pip3 podman-compose.

```
student@serverXX:~$ pip3 install podman-compose

Defaulting to user installation because normal site-packages is not writeable

Collecting podman-compose

…
```

Na installatie is het commando uiteraard beschikbaar.

```
student@serverXX:~$ podman-compose

usage: podman-compose [-h] [-v] [--in-pod in_pod] [--pod-args pod_args] [--env-
file env_file] [-f file]

…
```

We checken de versie van podman-compose.

```
student@serverXX:~$ podman-compose --version

podman-compose version 1.5.0

podman version 5.4.0
```

7.3 Componenten van een .yml-bestand

7.3.1 Services

In Podman Compose worden containers gedefinieerd als services. Hieronder een eenvoudig voorbeeld van een service die een bericht schrijft en actief blijft.

```
services:
  rhel_work:
    image: registry.access.redhat.com/ubi10/ubi
    container_name: rhel_worker
    command: ["/bin/bash", "-lc", "echo 'Hello from RHEL 10'; sleep 3600"]
    volumes:
      - shared_volume:/config:Z
    networks:
      - rhel_network
```

Hieronder de verschillende onderdelen:

- services: hoofdsectie waarin je containers definieert.
    - Service name
      Het is een label waarmee Podman Compose deze container herkent en beheert als onderdeel van je hele stack.
      Hier: rhel_work
    - Image
      Het image dat gebruikt wordt (hier een RHEL/UBI 10 image).
      Hier: registry.access.redhat.com/ubi10/ubi
    - container_name
      Hiermee geef je de container een specifieke naam, wat handig kan zijn voor beheer.
      Hier: rhel_worker.
    - Command
      Hier geef je een specifiek commando op dat wordt uitgevoerd wanneer de container wordt gestart. Hier wordt bash gestart dat “Hello from UBI10” op het scherm zet en erna 3600 seconden slaapt.<
      Hier: ["/bin/bash", "-lc", "echo 'Hello from RHEL 10'; sleep 3600"]
    - Volumes
      koppelt een named volume aan /config. Met :Z krijgt het mountpoint correcte SELinux-labels op RHEL.
      Hier: shared_volume:/config:Z
    - Networks
      koppelt de service aan een user-defined (rootless) of bridge (rootful) netwerk.
      Hier: rhel_network

Let op met inspingen:

- Je moet spaties gebruiken, geen tabs. Tabs zijn verboden omdat ze op verschillende systemen anders geïnterpreteerd worden.
- Je mag zelf kiezen hoeveel spaties je gebruikt per niveau — 2, 4 of zelfs 6 — zolang je binnen hetzelfde bestand consequent blijft.
- De aanbevolen standaard is 2 spaties per niveau, omdat het overzichtelijk en compact is.

7.3.2 Volumes

Volumes worden gebruikt om persistente opslag te definiëren die tussen containerrestarts behouden blijft.

```
volumes:
  shared_volume:
```

Dit definieert een volume genaamd shared_volume dat vervolgens door services kan worden gebruikt.

Toewijzen van volumes aan een service…

```
services:
  rhel_worker:
    …
    volumes:
      - shared_volume:/config:Z
```

In dit voorbeeld wordt het volume shared_volume gemount naar de directory /config in de rhel_worker-container. Dit betekent dat bestanden die naar /config worden geschreven, opgeslagen blijven, zelfs als de container opnieuw wordt opgestart.

7.3.3 Netwerken

Netwerken zorgen ervoor dat containers met elkaar kunnen communiceren. Je kunt standaardnetwerken gebruiken of aangepaste netwerken aanmaken voor meer controle over de containercommunicatie.

```
networks:
      - rhel_network
```

Dit maakt een netwerk genaamd rhel_network aan. Containers die deel uitmaken van hetzelfde netwerk kunnen via hun servicenaam met elkaar communiceren.

Een netwerk gebruiken in services…

```
services:
```

```
  rhel_worker:
    …
    networks:
      - rhel_network
```

In dit voorbeeld maakt de app-container deel uit van het netwerk rhel_network. Hierdoor kan de container communiceren met andere containers die aan hetzelfde netwerk zijn gekoppeld.

7.3.4 Depends_on

Het -veld in een Compose-bestand zorgt ervoor dat de ene container pas wordt gestart nadat een andere container is opgestart.

```
services:
  app:
    depends_on:
      - config_writer
```

Dat betekent: start eerst config_writer, dan pas app.

Let op: depends_on garandeert niet dat de afhankelijkheid volledig klaar is (zoals een database die luistert op een poort). Het regelt alleen de startvolgorde van containers.

7.3.5 Environment-variabelen

Deze worden gebruikt om de containerinstellingen te beheren.

```
services:
  db:
    image: mariadb:latest
    container_name: mariadb
    environment:
      MARIADB_ROOT_PASSWORD: rootpass
      MARIADB_DATABASE: mydb
      MARIADB_USER: appuser
      MARIADB_PASSWORD: secret
     …
```

De environment-sectie is cruciaal voor het automatisch configureren van MariaDB bij het opstarten van de container. Ik denk dat dit duidelijk is aangezien we dit vorig hoofdstuk reeds besproken hebben.

7.3.6 Command

Het command-gedeelte wordt gebruikt om een specifiek commando uit te voeren wanneer de container start.

Hieronder enkele voorbeelden.

```
services:
  app:
    command: "echo Hallo wereld"
```

Dit is duidelijk: “Hallo wereld” wordt op het scherm gezet.

```
    command: ["/bin/bash", "-c", "echo Hallo wereld; sleep 3600"]
```

Dit is de lijstvorm.

- Wordt door Compose exact geïnterpreteerd als een reeks argumenten.
- Eénduidig: elk element in de lijst is één argument.
- Werkt betrouwbaar met complexe shell-opdrachten, variabelen, pipes, , , enz.
- Dit is de aanbevolen vorm bij gebruik van shellcommando’s in Compose.

```
    command: "/bin/bash -c echo Hallo wereld; sleep 3600"
```

Gebruik dit niet!

Compose kan dit als volgt opslitsen: ["/bin/bash", "-c", "echo", "Hallo", "wereld;", "sleep", "3600"]

Dit werkt dan uiteraard niet!!!

7.3.7 Folded block en literal block

Een folded block (symbool > in script) voegt regels samen tot één string, waarbij nieuwe regels worden vervangen door spaties.

Een literal block (symbool | in script) behoudt alle regelafbrekingen exact zoals ze zijn — elke nieuwe regel blijft staan.

Voorbeeld folded block:

```
beschrijving: >
  Dit is regel één.
  Dit is regel twee.
  Dit is regel drie.
```

Dit wordt geïnterpreteert als volgd:

```
  Dit is regel één. Dit is regel twee. Dit is regel drie.
```

Voorbeeld literal block:

```
beschrijving: |
  Dit is regel één.
  Dit is regel twee.
  Dit is regel drie.
```

Dit wordt geïnterpreteerd als volgt:

```
Dit is regel één.
Dit is regel twee.
Dit is regel drie.
```

7.4 Voorbeeld 1: gedeelde map

7.4.1 .yml-bestand

Genoeg uitleg hierover…

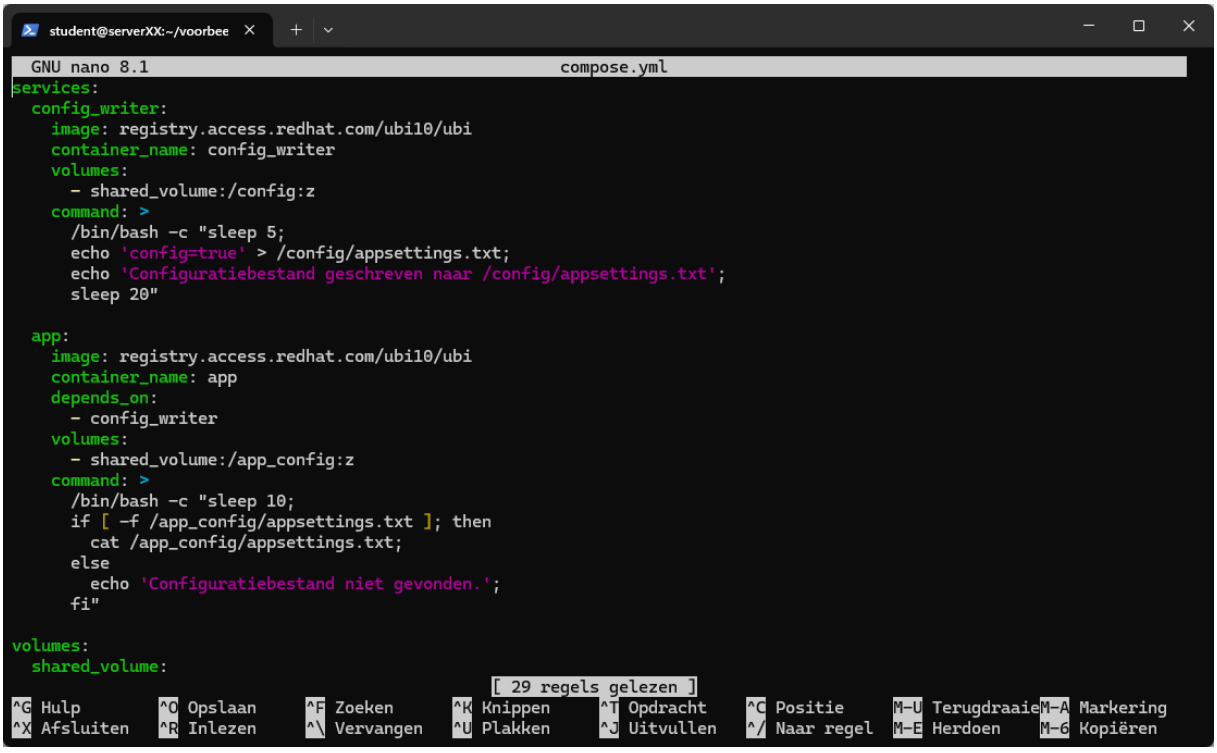
Hier is een voorbeeld van een compose.yml-bestand dat twee containers definieert: een container config_writer die een configuratiebestand schrijft, en een container app die dat configuratiebestand leest.

```
services:
  config_writer:
    image: registry.access.redhat.com/ubi10/ubi
    container_name: config_writer
    volumes:
      - shared_volume:/config:z
    command: >
      /bin/bash -c "sleep 5;
      echo 'config=true' > /config/appsettings.txt;
      echo 'Configuratiebestand geschreven naar /config/appsettings.txt';
      sleep 20"

  app:
    image: registry.access.redhat.com/ubi10/ubi
    container_name: app
    depends_on:
      - config_writer
    volumes:
```

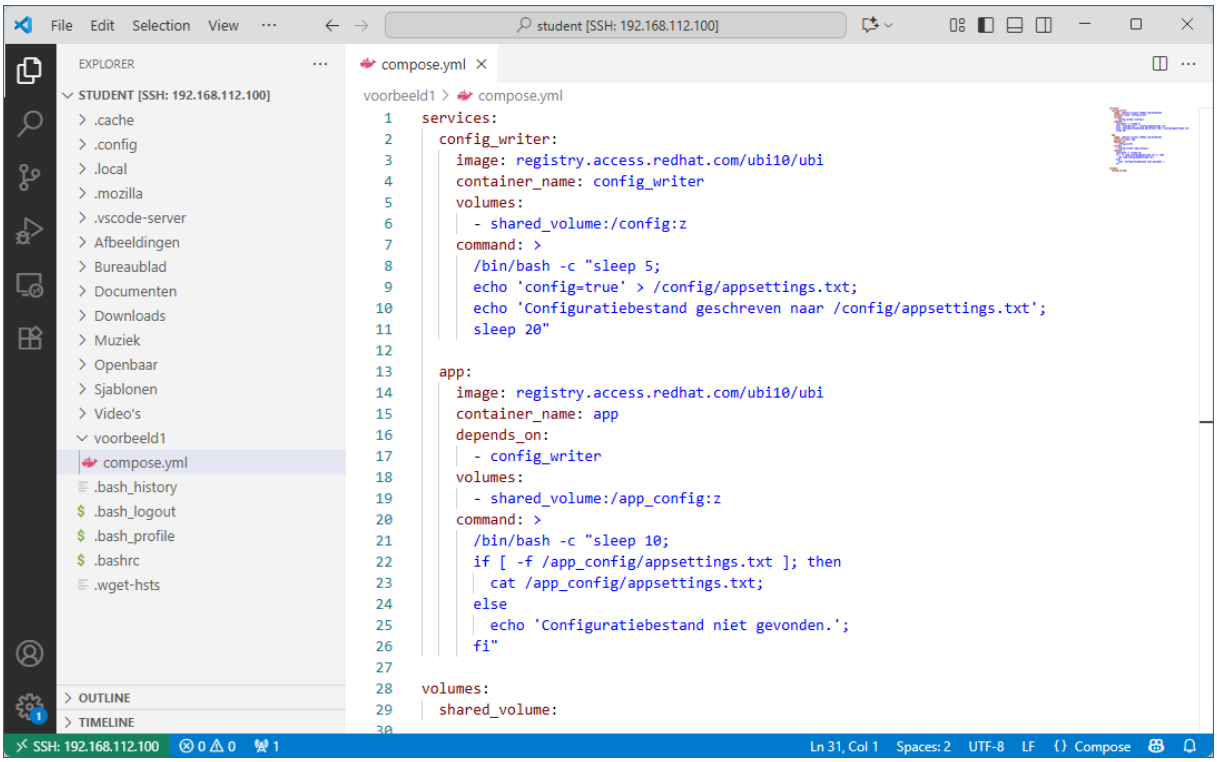
```
      - shared_volume:/app_config:z
    command: >
      /bin/bash -c "sleep 10;
      if [ -f /app_config/appsettings.txt ]; then
        cat /app_config/appsettings.txt;
      else
        echo 'Configuratiebestand niet gevonden.';
      fi"

volumes:
  shared_volume:
```

Maak het bestand compose.yml aan in ~/voorbeeld1.

```
student@serverXX:~$ mkdir voorbeeld1

student@serverXX:~$ nano voorbeeld1/compose.yml
```



Je kan uiteraard ook gebruik maken van Visual Studio Code.



Dit compose.yml-bestand definieert twee ubi10/ubi-containers:

- config_writer: Deze container maakt een configuratiebestand aan in een gedeeld volume (shared_volume) na 5 seconden wachten. Het bestand wordt opgeslagen in de gedeelde opslag en de container blijft vervolgens actief voor 20 seconden.
- app: Deze container start na de config_writer dankzij de depends_on-parameter. Het controleert of het configuratiebestand aanwezig is in het gedeelde volume en leest de inhoud, of geeft een melding als het bestand niet gevonden is.

Het volume shared_volume zorgt ervoor dat beide containers toegang hebben tot hetzelfde configuratiebestand.

7.4.2 Podman-compose up

Zorg ervoor dat je terminal of command prompt is geopend in de directory waar je compose.yml-bestand zich bevindt.

```
student@serverXX:~$ cd voorbeeld1/
```

Voer het volgende commando uit:

```
student@serverXX:~/voorbeeld1$ podman-compose up

c9f5c430823dcf4e1f090bd53282efbfe17f983c29cfcc3bd0ddb53b895ecc0b

9ae589470b1a4334261ccc37796aa1eb996215cb82edef29255962578c348e72

9a29ac43fad3c6ab1b8f6f28e65c87be3ccb3455afb2dca317cdf9bb7a49f678
```

```
[config_writer] | Configuratiebestand geschreven naar /config/appsettings.txt

[app]           | config=true
```

Het eerste dat “podman-compose up” doet, is het lezen van de configuratie in het compose.yml bestand.

> Dit bestand definieert de services, netwerken, volumes, en andere instellingen die nodig zijn om de applicatie te draaien zoals we gezien hebben.

Op basis van de services die zijn gedefinieerd in het compose.yml bestand, start Podman Compose alle benodigde containers. Dit omvat:

- Het maken van nieuwe containers op basis van de gespecificeerde Docker-images.
- Het configureren van netwerken, volumes en omgevingsvariabelen zoals gedefinieerd in het bestand.
- Het uitvoeren van elk gedefinieerd command of entrypoint.

Je krijgt onderstaande output als je podman-compose up een tweede keer uitvoert.

```
student@serverXX:~/voorbeeld1$ podman-compose up

[config_writer] | Configuratiebestand geschreven naar /config/appsettings.txt

[app]           | config=true

student@serverXX:~/voorbeeld1$
```

We leggen dit nu gedetailleerd uit.

1. Lezen en verwerken van compose.yml

In dit geval detecteert Podman Compose twee services:

- config_writer: Een service die een configuratiebestand aanmaakt.
- app: Een service die afhankelijk is van config_writer en het configuratiebestand controleert.

Docker Compose maakt een volume genaamd shared_volume. Dit volume wordt gedeeld tussen de containers config_writer en app, en is toegankelijk op verschillende paden binnen de containers (vandaar z i.p.v. Z, anders zou maar 1 container toegang hebben):

- In config_writer wordt het gekoppeld aan /config.
- In app wordt het gekoppeld aan /app_config.

2. Starten van de config_writer service

Docker Compose start de config_writer container met de volgende instellingen:

- De container gebruikt de Docker-image registry.access.redhat.com/ubi10/ubi.
- Het volume shared_volume wordt gemount naar de map /config in de container.

- Het opgegeven command wordt uitgevoerd, namelijk een Bash-script.
    - /bin/bash -c "sleep 5;
      De container wacht 5 seconden voordat hij verder gaat.
    - echo 'config=true' > /config/appsettings.txt;
      Daarna schrijft hij de string 'config=true' naar een bestand genaamd appsettings.txt in de map /config, die wordt gedeeld via het volume.
    - echo 'Configuratiebestand geschreven naar /config/appsettings.txt': Deze regel wordt naar de container-uitvoer geschreven om aan te geven dat het configuratiebestand succesvol is geschreven.
    - sleep 20: Ten slotte wacht de container nog eens 20 seconden voordat hij volledig stopt.

3. Starten van de app service (na depends_on)

Docker Compose wacht met het starten van de app container totdat de config_writer container is gestart, vanwege de depends_on configuratie.

Let op: depends_on zorgt er alleen voor dat de config_writer container is gestart, niet dat het proces binnen de container (zoals het schrijven van het bestand) is voltooid. Dit betekent dat app mogelijk eerder kan beginnen dan dat config_writer zijn taken heeft voltooid, maar er is wel enige overlap door de ingebouwde vertragingen (zie de sleep-opdrachten in beide containers).

De app container wordt vervolgens gestart met de volgende instellingen:

- De container gebruikt ook de Docker-image registry.access.redhat.com/ubi10/ubi.
- Het volume shared_volume wordt gemount naar de map /app_config in de container.
- Het opgegeven script wordt uitgevoerd.
    - /bin/bash -c "sleep 10;
      De container wacht 10 seconden voordat hij verder gaat. Dit zorgt ervoor dat er enige tijd is verstreken, zodat config_writer mogelijk het configuratiebestand heeft kunnen schrijven.
    - if [-f /app_config/appsettings.txt]
      Controleert of het configuratiebestand appsettings.txt bestaat in de map /app_config (dat gekoppeld is aan hetzelfde volume als config_writer, maar in een andere map).
        - then
          cat /app_config/appsettings.txt;
          Als het bestand bestaat, worden de inhoud van het bestand (config=true) naar de container-uitvoer geschreven.
        - else
          echo 'Configuratiebestand niet gevonden.'
          Als het bestand niet wordt gevonden, wordt er een foutmelding naar de container-uitvoer geschreven.

4. Duur van het uitvoeren van de containers

- config_writer
  Deze container blijft draaien gedurende de “sleep 20” fase, maar zal daarna stoppen.

- App
  Deze container zal direct na het uitvoeren van het script stoppen.

7.4.3 Logs bekijken

Als je podman-compose up hebt uitgevoerd, kun je de log-uitvoer van beide containers zien.

```
student@serverXX:~/voorbeeld1$ podman logs config_writer

Configuratiebestand geschreven naar /config/appsettings.txt

Configuratiebestand geschreven naar /config/appsettings.txt
```

- Je ziet 2 keer het weggeschreven bericht omdat je 2x “podman-compose up” hebt uitgevoerd.

```
student@serverXX:~/voorbeeld1$ podman logs app

config=true

config=true
```

  - Als het bestand appsettings.txt bestaat, zal de inhoud (config=true) worden weergegeven.
  - Dat is hier 2x gebeurd omdat je 2x “podman-compose up” hebt uitgevoerd.

Naast de logs van de containers kan je ook gebruik maken van onderstaande.

```
student@serverXX:~/voorbeeld1$ podman-compose logs

9a29ac43fad3 config=true

9a29ac43fad3 config=true

9ae589470b1a Configuratiebestand geschreven naar /config/appsettings.txt

9ae589470b1a Configuratiebestand geschreven naar /config/appsettings.txt
```

7.4.4 Podman-compose down

Het commando podman-compose down wordt gebruikt om de hele Docker Compose-opstelling te verwijderen. Dit commando stopt (indien nodig) en verwijdert alle containers, netwerken, volumes en andere resources die zijn aangemaakt door podman-compose up.

```
student@serverXX:~/voorbeeld1$ podman-compose down

app

config_writer

app

config_writer

c9f5c430823dcf4e1f090bd53282efbfe17f983c29cfcc3bd0ddb53b895ecc0b
```

```
voorbeeld1_default
```

De containers en netwerken zijn nu verwijderd.

```
student@serverXX:~/voorbeeld1$ podman ps –a

…

student@serverXX:~/voorbeeld1$ podman network ls

NETWORK ID    NAME        DRIVER

2f259bab93aa  podman      bridge
```

Volumes worden niet verwijderd. In dit geval staat het volume in /home/student/.local/share/containers/storage/volumes/voorbeeld1_shared_volume.

```
student@serverXX:~/voorbeeld1$ ls
/home/student/.local/share/containers/storage/volumes/voorbeeld1_shared_volume/
_data

appsettings.txt
```

Optioneel worden ook volumes verwijderd als je de optie -v meegeeft met “podman -compose down -v”. Aangezien we al de containers en het netwerk hebben verwijderd zal enkel het volume verwijderd worden.

```
student@serverXX:~/voorbeeld1$ podman–compose up

…

student@serverXX:~/voorbeeld1$ podman–compose down –v

student@serverXX:~/voorbeeld1$ ls
/home/student/.local/share/containers/storage/volumes/

student@serverXX:~/voorbeeld1$
```

7.4.5 Podman-compose down versus podman-compose stop

- podman-compose down
  Dit stopt en verwijdert de containers, netwerken en eventueel volumes (met de -v vlag). Het is een volledig opruimingscommando.
- podman-compose stop
  Dit stopt alleen de containers, maar verwijdert ze niet. De containers blijven bestaan en kunnen later worden herstart met podman-compose start zonder opnieuw te worden gemaakt.

7.4.6 Podman-compose up -d

De optie -d bij podman-compose up staat voor detached mode net zoals -d bij het “podman run”.

Containers worden opgestart op de achtergrond. Dit betekent dat ze onafhankelijk van de terminal draaien, en de terminal wordt onmiddellijk vrijgegeven nadat de containers zijn gestart.

```
student@serverXX:~/voorbeeld1$ podman-compose up -d

config_writer

app
```

Je zal nu de logboeken moeten raadplegen om te zien wat er is gebeurd. Het eenvoudigste is via podman-compose.

```
student@serverXX:~/voorbeeld1$ podman-compose logs

11284f1dc973 Configuratiebestand geschreven naar /config/appsettings.txt

d1e2f29cfa5e config=true
```

Je ziet nu de log entries van alle containers.

Wil je het log bekijken van een specifieke container voer je dit uit zoals hieronder staat weergegeven.

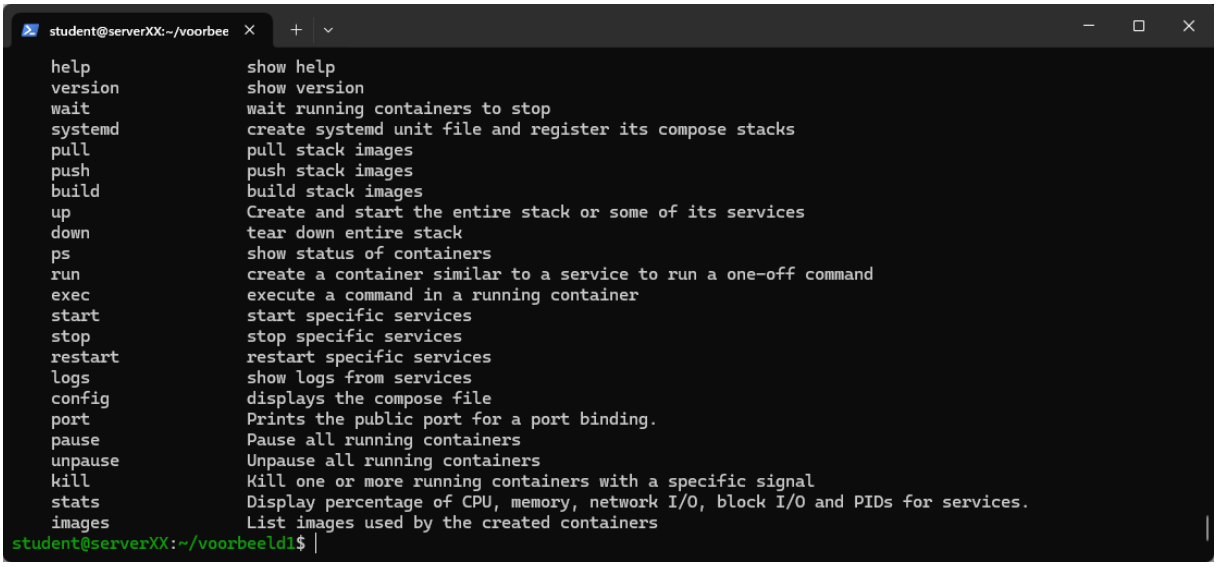
```
student@serverXX:~/voorbeeld1$ podman-compose logs config_writer

Configuratiebestand geschreven naar /config/appsettings.txt
```

7.4.7 Podman-compose help

Voer onderstaande hiervoor uit.

```
student@serverXX:~/voorbeeld1$ podman-compose --help
```

Je zal o.a. lijst zien met commando's die je kan gebruiken.



7.5 Voorbeeld 2: Wordpress

In vorig hoofdstuk hebben we Wordpress geïnstalleerd in 2 containers.

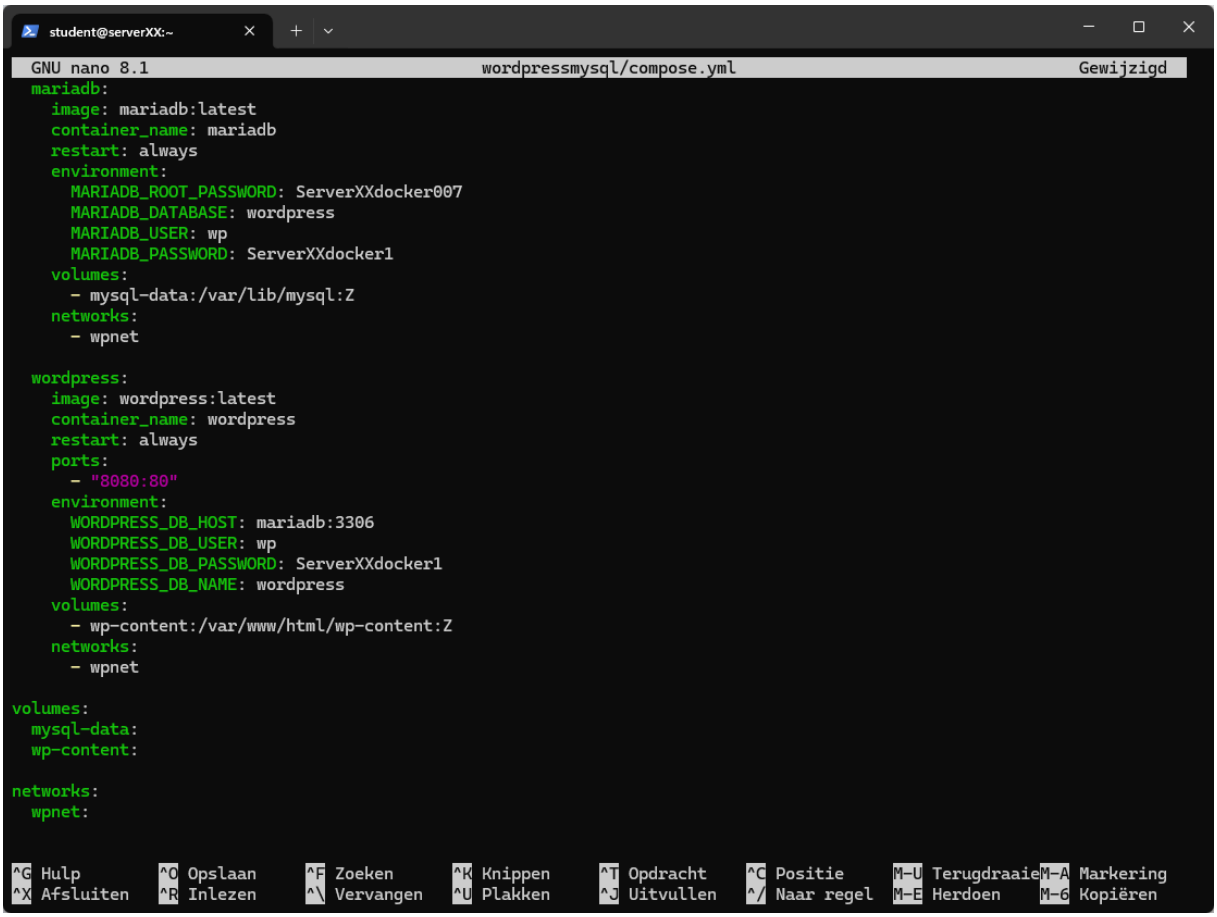
We kunnen dat ook opbouwen met Podman-compose.

7.5.1 .yml-bestand

We maken eerst weer een directory aan.

```
student@serverXX:~$ mkdir wordpressmysql
student@serverXX:~$ nano wordpressmysql/compose.yml
services:
  mariadb:
    image: mariadb:latest
    container_name: mariadb
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: ServerXXdocker007
      MARIADB_DATABASE: wordpress
      MARIADB_USER: wp
      MARIADB_PASSWORD: ServerXXdocker1
    volumes:
      - mysql-data:/var/lib/mysql:Z
    networks:
      - wpnet

  wordpress:
    image: wordpress:latest
    container_name: wordpress
    restart: always
```

```
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: mariadb:3306
      WORDPRESS_DB_USER: wp
      WORDPRESS_DB_PASSWORD: ServerXXdocker1
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wp-content:/var/www/html/wp-content:Z
    networks:
      - wpnet

volumes:
  mysql-data:
  wp-content:

networks:
  wpnet:
```



Of met Visual Studio Code natuurlijk...

compose.yml ●

wordpressmsql > compose.yml

```yaml
services:
  mariadb:
    image: mariadb:latest
    container_name: mariadb
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: ServerXXdocker007
      MARIADB_DATABASE: wordpress
      MARIADB_USER: wp
      MARIADB_PASSWORD: ServerXXdocker1
    volumes:
      - mysql-data:/var/lib/mysql:Z
    networks:
      - wpnet

  wordpress:
    image: wordpress:latest
    container_name: wordpress
    restart: always
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: mariadb:3306
      WORDPRESS_DB_USER: wp
      WORDPRESS_DB_PASSWORD: ServerXXdocker1
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wp-content:/var/www/html/wp-content:Z
    networks:
      - wpnet

volumes:
  mysql-data:
  wp-content:

networks:
  wpnet:
```

Met de kennis die je ondertussen vergaard hebt zou dit YML-bestand duidelijk moeten zijn. Let op het subtiele verschil met voorbeeld1 betreffende de volumes. Hier staat een hoofdletter Z i.p.v. kleine letter z omdat je hier wil dat niet beide containers toegang hebben tot mysql-data enwp-content.

Daarnaast zie je dat er mariadb staat bij image en niet docker.io/library/mariadb:latest. Dat hoeft niet… RHEL vraagt waar te downloaden zoals je weet als het image niet lokaal aanwezig is! Dat is al behandeld.

Je ziet ook de poortkoppeling bij Wordpress.

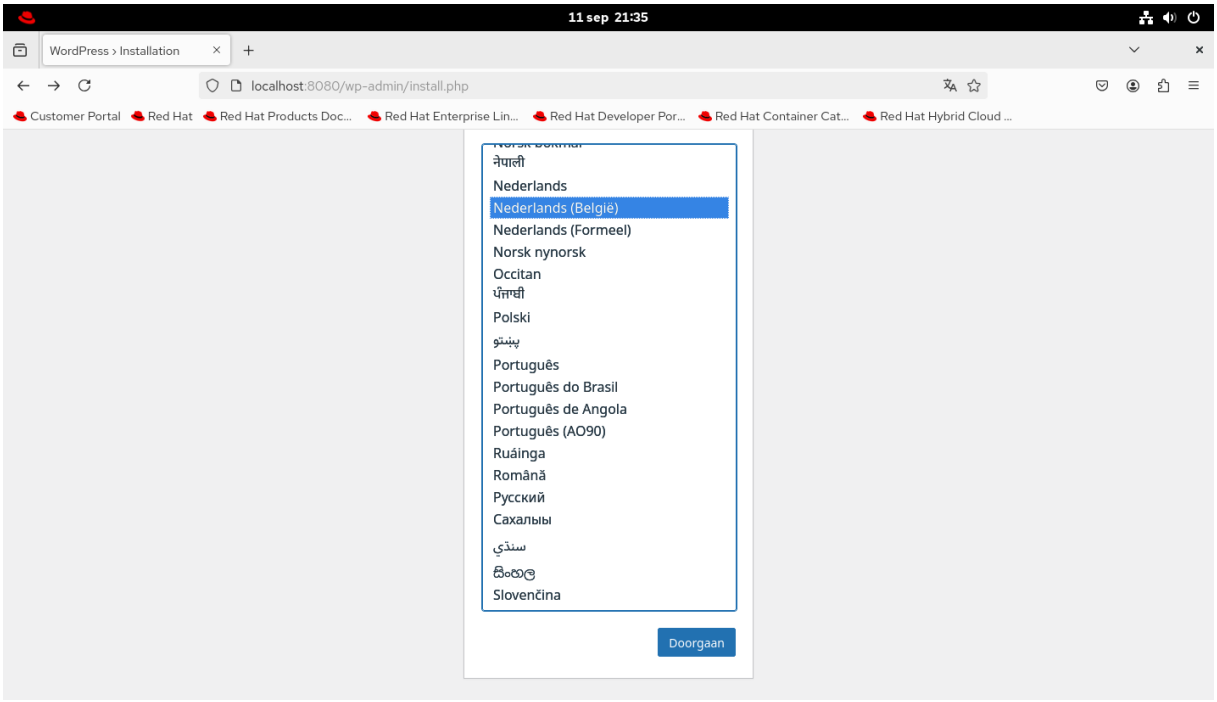
7.5.2 Podman-compose up

Ga weer naar de juiste directory (in dit geval wordpressmysql) en voer “podman-compose up” uit.

```
student@serverXX:~/wordpressmsql$ podman compose up
```

7.5.3 Website instellen

Ga naar de browser in je RHEL-VM en voer de gegevens voor je website in zoals je al in vorig hoofdstuk hebt gedaan.



Aangezien dit hetzelfde is dan in vorig hoofdstuk gaan we er hier niet verder op in.