6 Podman images

6.1 Inleiding

In dit hoofdstuk zullen verschillende images bespreken die vaak gebruikt worden. Let op: niet alle container images zijn gebaseerd op RHEL. Het kan dus zijn dat je andere commando's moet gebruiken in de container dan degene die je gewend bent…

6.2 Webserver

Met podman search kan je zoeken naar container images die gespecifieerd zijn in /etc/containers/registries.conf:

```
student@serverXX:~$ less /etc/containers/registries.conf

…

unqualified-search-registries = ["registry.access.redhat.com",
"registry.redhat.io", "docker.io"]

…
```

Je ziet heel veel commentaar (regels beginnend met #) maar je ziet rond regel 20 een regel zonder #. Deze heb ik hierboven weergegeven.

unqualified-search-registries wil zeggen dat wanneer je zoekt zonder registry podman gaat zoeken op de 3 plaatsen die hier staan aangegeven.

We zoeken eens naar een nginx-container.

```
student@serverXX:~$ podman search nginx

NAME                                    DESCRIPTION

…
```

Je ziet zeer veel images.

We verfijnen dit door te zoeken naar regels waarin ubi10 voorkomt.

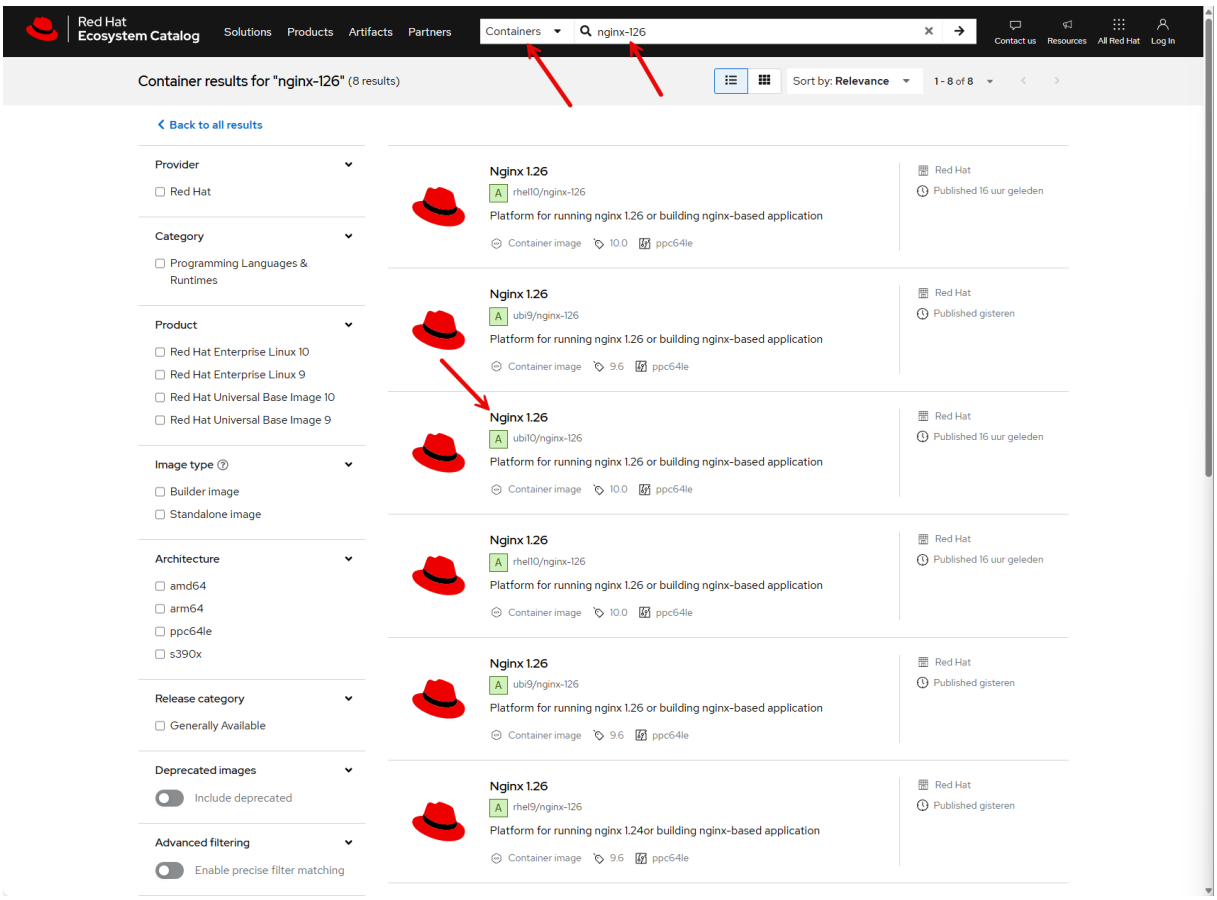
```
student@serverXX:~$ podman search nginx | grep ubi10

registry.access.redhat.com/ubi10/nginx-126      …

registry.redhat.io/ubi10/nginx-126              …
```

Zoals jullie weten is authenticatie vereist voor registry.redhat.io/ubi10/nginx-126 en niet voor registry.access.redhat.com/ubi10/nginx-126.

In de Red Hat Ecosystem Catalog is ook info over UBI-images opgenomen.

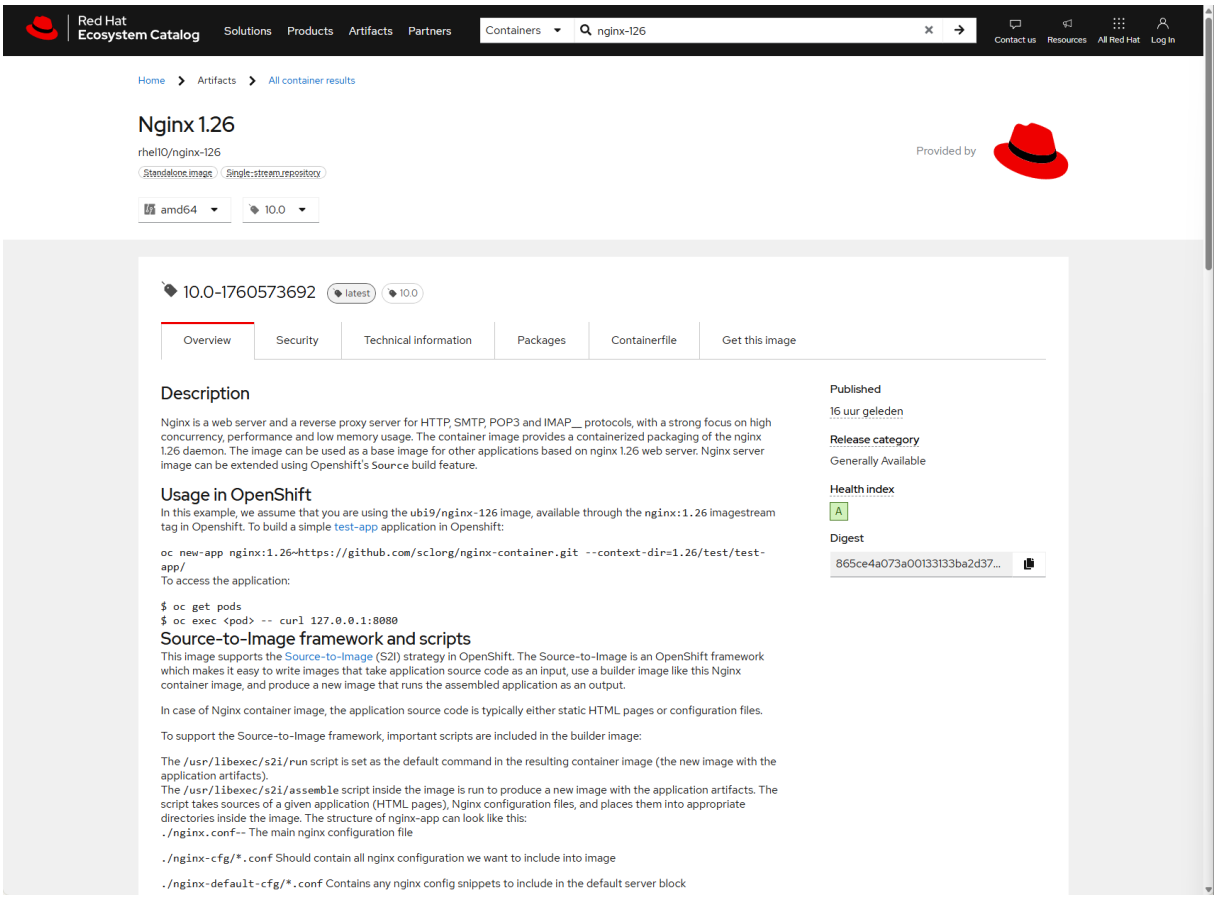
We surfen naar [undefined - Red Hat Ecosystem Catalog].

Kies nu voor containers bovenaan en typen in het zoekvenster nginx-126 in.



Klik op Nginx 1.26 bij ubi10 zoals hierboven staat aangegeven.

We krijgen nu onderstaande pagina.



Bij nieuwe packages is de info vaak niet beschikbaar.

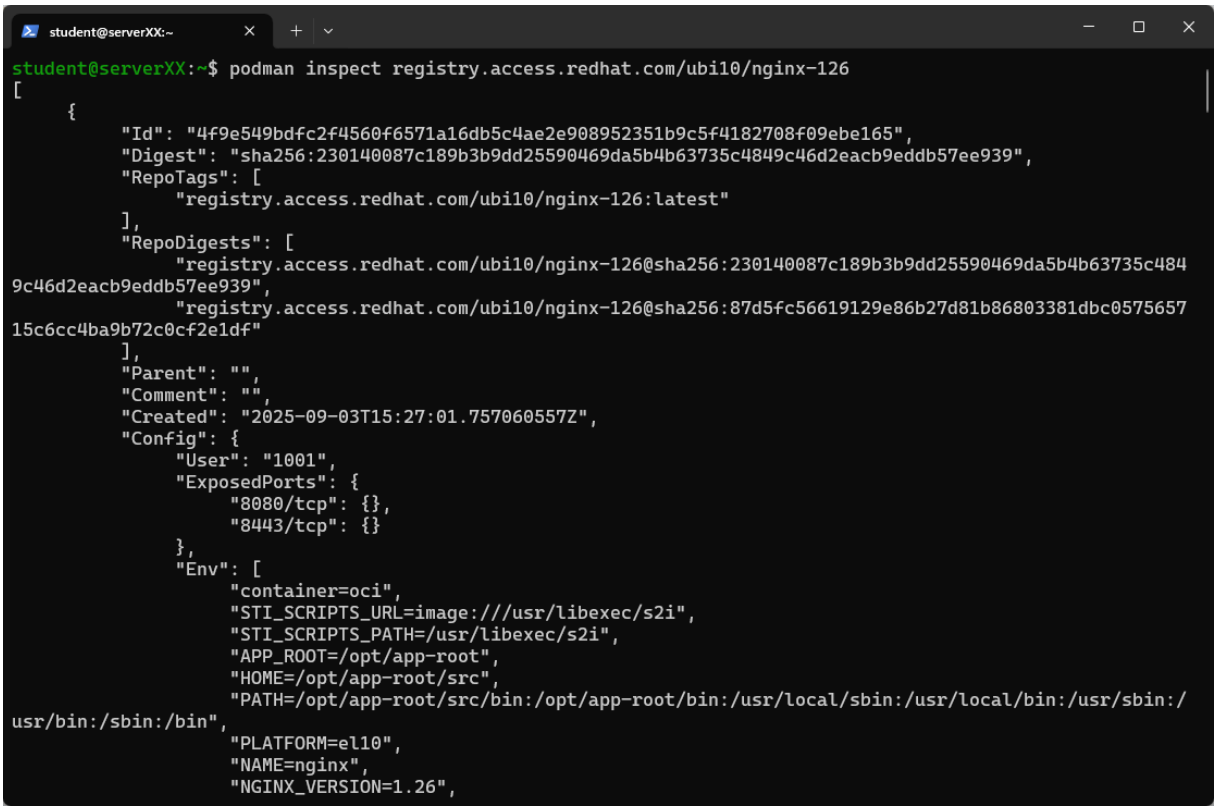
We pullen het image van `registry.access.redhat.com`.

```
student@serverXX:~$ podman pull registry.access.redhat.com/ubi10/nginx-126
```

Erna inspecteren we `ubi10/nginx-126`.

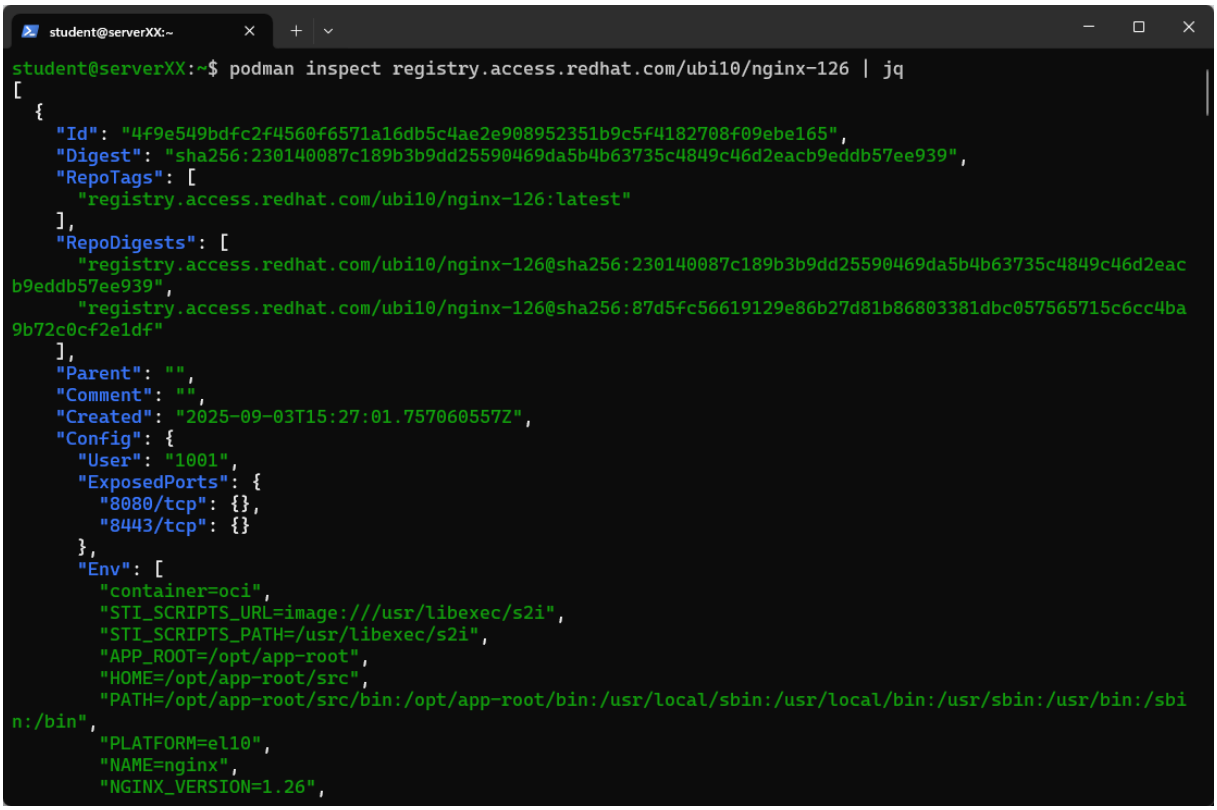
```
student@serverXX:~$ podman inspect ubi10/nginx-126
```

Je krijgt nu nogal onoverzichtelijke output.



Dit komt omdat het JSON-formaat is.

Je kan dit verbeteren door jq te gebruiken. Dit maakt JSON mooi ingesprongen met kleuren.

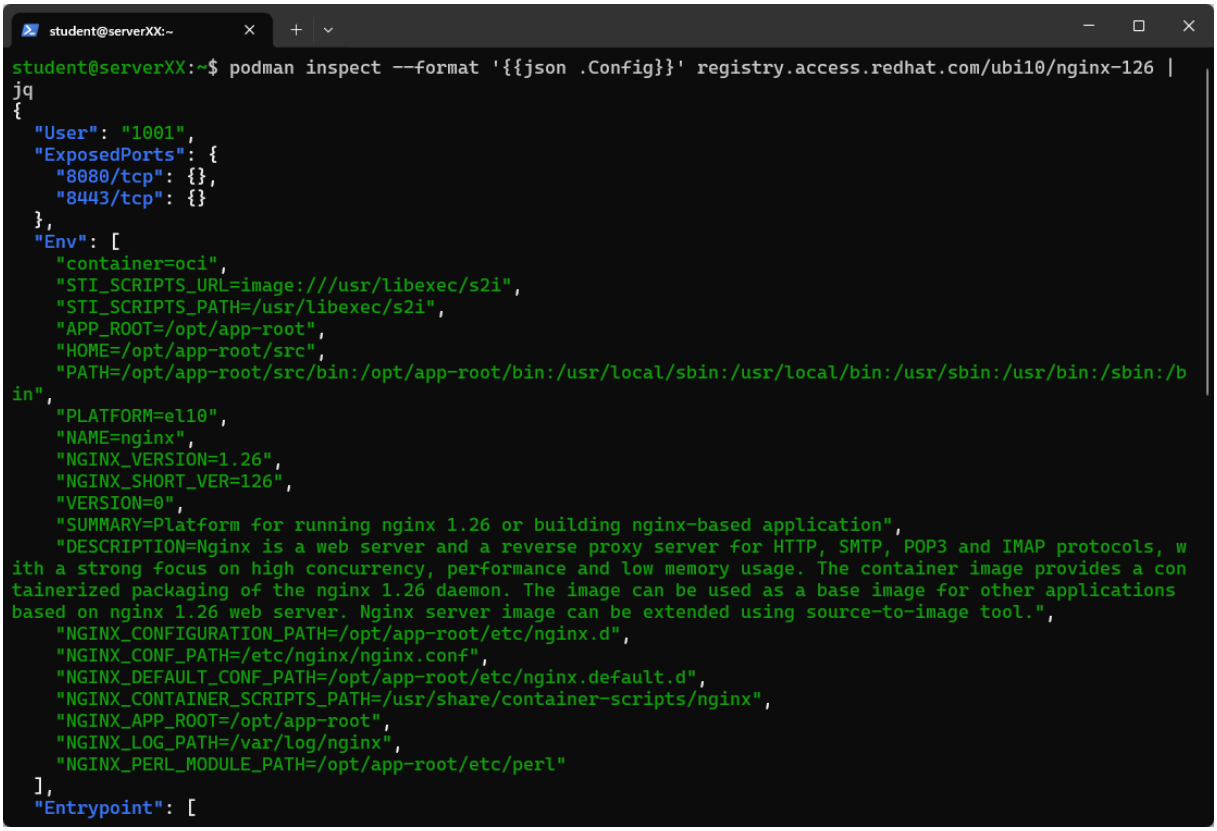


Podman heeft ook een mogelijkheid om het verder te filteren.

```
student@serverXX:~$ podman inspect --format '{{json .Config}}'
registry.access.redhat.com/ubi10/nginx-126 | jq
```

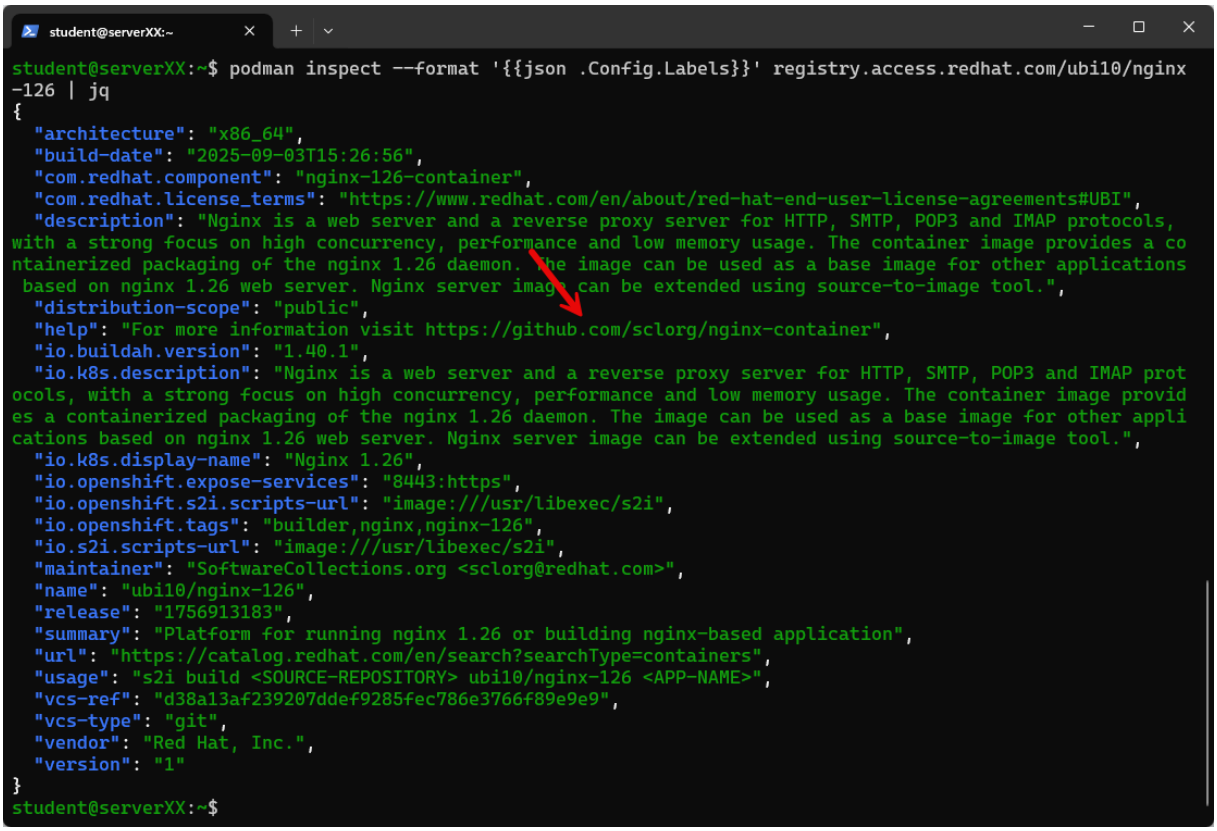
Met de --format-optie gebruik je een Go template om een specifiek deel uit de inspectie te filteren.

- .Config verwijst naar het veld Config in de JSON-structuur van de inspect-output.
- {{json .Config}} zegt: geef dit veld terug als JSON.



Je kan verder verfijnen naar .Config.Labels.

```
student@serverXX:~$ podman inspect --format '{{json .Config.Labels}}'
registry.access.redhat.com/ubi10/nginx-126 | jq
```



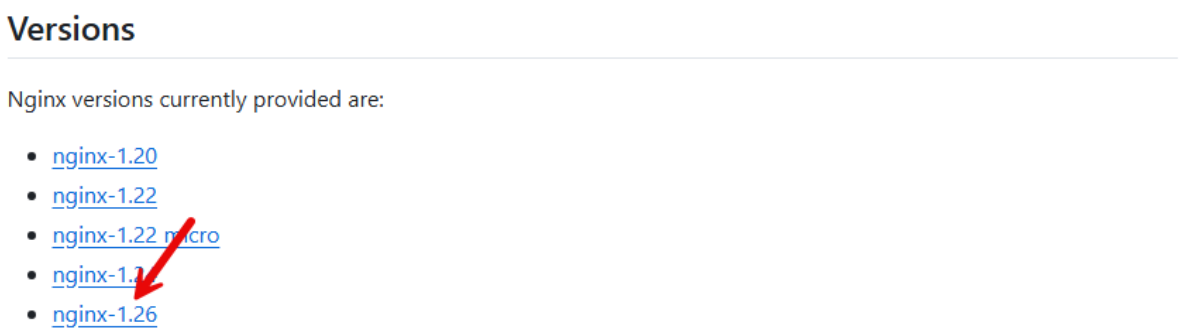
Hier zie je dus waar je meer info vindt over het container image.

Ik had het natuurlijk ook kunnen opvragen door te verfijnen naar verder verfijnen naar .Config.Labels.help.

```
student@serverXX:~$ podman inspect --format '{{json .Config.Labels.help}}'
registry.access.redhat.com/ubi10/nginx-126 | jq

"For more information visit https://github.com/sclorg/nginx-container"
```

We gaan nu naar https://github.com/sclorg/nginx-container.

Klik nu in die pagina op nginx-1.26 bij Versions.



Hier vind je zéér interessante info terug.

Je vindt info over hoe je Containerfile opzet voor deze image.

3. Prepare an application inside a container

This step usually consists of at least these parts:

- putting the application source into the container
- moving configuration files to the correct place (if available in the application source code)
- setting the default command in the resulting image

For all these three parts, you can either set up all manually and use the `nginx` command explicitly in the Dockerfile (3.1.), or you can use the Source-to-Image scripts inside the image (3.2.; see more about these scripts in the section "Source-to-Image framework and scripts" above), that already know how to set-up and run some common Nginx applications.

3.1. To use your own setup, create a Dockerfile with this content:

```
FROM registry.access.redhat.com/ubi9/nginx-126

# Add application sources
ADD test-app/nginx.conf "${NGINX_CONF_PATH}"
ADD test-app/nginx-default-cfg/*.conf "${NGINX_DEFAULT_CONF_PATH}"
ADD test-app/nginx-cfg/*.conf "${NGINX_CONFIGURATION_PATH}"
ADD test-app/*.html .

# Run script uses standard ways to run the application
CMD nginx -g "daemon off;"
```

3.2 is niet interessant voor ons op de website.

Onderstaande stappen komen ons bekend voor…

4. Build a new image from a Dockerfile prepared in the previous step

```
podman build -t nginx-app .
```

5. Run the resulting image with the final application

```
podman run -d nginx-app
```

Eronder staat ook hoe je onmiddellijk de image kan gebruiken via een gekoppelde directory.

Direct usage with a mounted directory

An example of the data on the host for the following example:

```
$ ls -lZ /wwwdata/html
-rw-r--r--. 1 1001 1001 54321 Jan 01 12:34 index.html
-rw-r--r--. 1 1001 1001  5678 Jan 01 12:34 page.html
```

If you want to run the image directly and mount the static pages available in the `/wwwdata/` directory on the host as a container volume, execute the following command:

```
$ podman run -d --name nginx -p 8080:8080 -v /wwwdata:/opt/app-root/src:Z ubi9/nginx-126 nginx -g "daemon off;"
```

This creates a container named `nginx` running the Nginx server, serving data from the `/wwwdata/` directory. Port 8080 is exposed and mapped to the host. You can pull the data from the nginx container using this command:

```
$ curl -Lk 127.0.0.1:8080
```

You can replace `/wwwdata/` with location of your web root. Please note that this has to be an **absolute** path, due to podman requirements.

Met de kennis die je al hebt opgedaan kan je beiden instellen maar we zullen ons beperken tot het gebruik met een gekoppelde directory.

Maak eerst de directorystructuur aan.

```
student@serverXX:~$ sudo mkdir -p /wwwdata/html
```

Maak de startpagina index.html aan.

```
student@serverXX:~$ sudo echo "Dit is een eenvoudige website" | sudo tee
/wwwdata/html/index.html

Dit is een eenvoudige website
```

We controleren de webpagina.

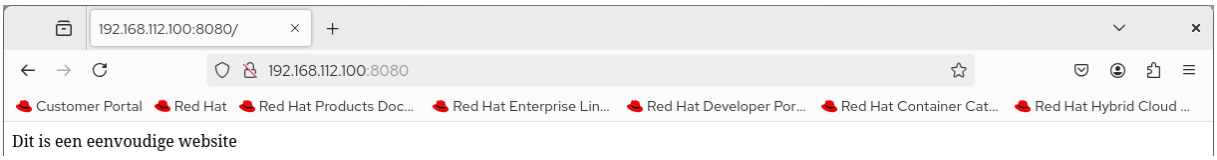
```
student@serverXX:~$ cat /wwwdata/html/index.html

Dit is een eenvoudige website
```

We starten nu een container op aan de hand van het image.

```
student@serverXX:~$ sudo podman run -d --name nginx -p 8080:8080 -v
/wwwdata/html:/opt/app-root/src:Z ubi10/nginx-126 nginx -g "daemon off;"

…

45bc04da9ccfc27077875faaad64e09499ab587adb340b06fbb7846b75a2f418
```

Er staat een fout in de beschrijving (/wwwdata**/html** i.p.v. /wwwdata) op de webpagina maar het principe is duidelijk.

Ga nu naar http://192.168.112.100:8080/.



6.3 Database-server

We zoeken nu eens een mariadb-image.

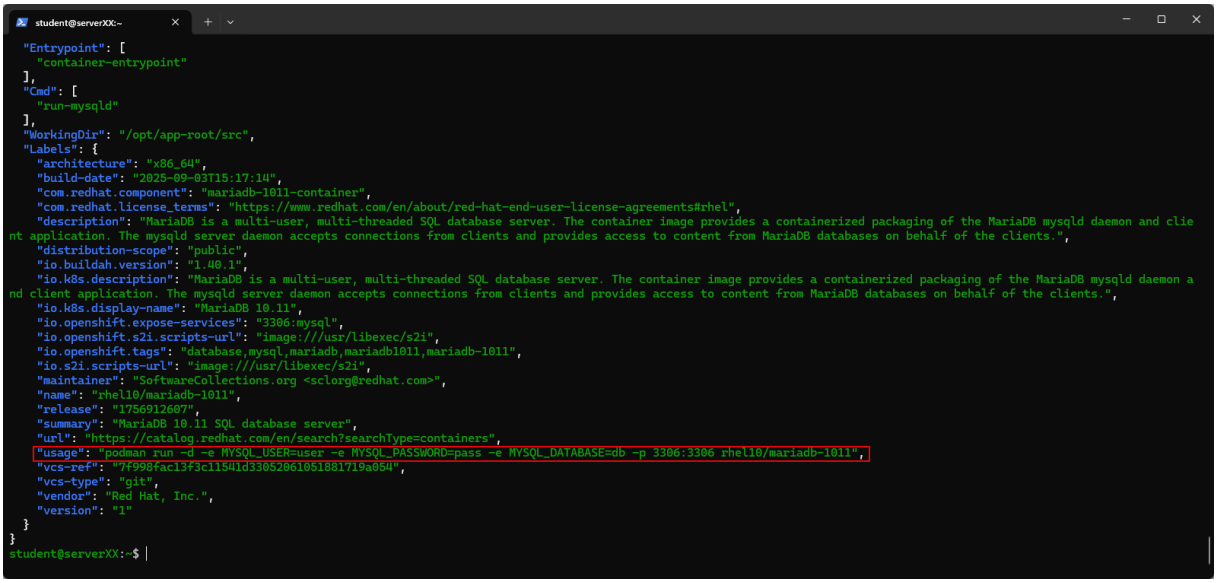
```
student@serverXX:~$ podman search mariadb

…

registry.redhat.io/rhel10/mariadb-1011

…
```

Zoals je ziet staat er een container in gebaseerd op rhel10.

We zoeken weer informatie op over deze container image na het inloggen (nodig voor registry.redhat.io) en pullen.

```
student@serverXX:~$ podman login registry.redhat.io
```

```
Username: stijnjacobs

Password:

Login Succeeded!

student@serverXX:~$ podman pull registry.redhat.io/rhel10/mariadb-1011

…

69da3594e8f55d55013d1307b53e0c2e5cde7e19147fcf4270aa9fdb4935120b

student@serverXX:~$ podman inspect --format '{{json .Config}}'
registry.redhat.io/rhel10/mariadb-1011 | jq
```



Hier staat heel duidelijk hoe je de containerimage moet gebruiken!

Je kan het onderdeel “usage” natuurlijk ook onmiddellijk opvragen.

```
student@serverXX:~$ podman inspect --format '{{json .Config.Labels.usage}}'
registry.redhat.io/rhel10/mariadb-1011 | jq

"podman run -d -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=db -
p 3306:3306 rhel10/mariadb-1011"
```

Aan de hand hiervan kunnen we onderstaande opbouwen.

```
student@serverXX:~$ podman run -d -e MYSQL_USER=student -e MYSQL_PASSWORD=abc -
e MYSQL_DATABASE=testdb -p 3306:3306 rhel10/mariadb-1011

17c89d9b9ffdae9508c46b33b39e5d7a4bc2c8b9f8f3becf3a2950fb7ba1e01b
```

De database-server is gestart en blijft draaien.

```
student@serverXX:~$ podman ps

CONTAINER ID  IMAGE                                                COMMAND     …
```

```
17c89d9b9ffd  registry.redhat.io/rhel10/mariadb-1011:latest run-mysqld  …
```

Dit komt omdat bij COMMAND run-mysqld staat. Zolang dit commando actief blijft draait de database-server.

Je kan nu uiteraard verbinding maken met deze database vanaf ServerXX. Je moet wel de client installeren.

```
student@serverXX:~$ sudo dnf install mariadb
```

We kunnen nu verbinding maken met het wachtwoord abc.

```
student@serverXX:~$ mysql -h 127.0.0.1 -P 3306 -u student -p testdb
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.11.11-MariaDB MariaDB Server
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [testdb]>
```

We gaan niet zelf de database configureren en verlaten MariaDB.

```
MariaDB [testdb]> exit
Bye
```

6.4 Wordpress

6.4.1 Images die we gebruiken

Vorig academiejaar hebben jullie Wordpress leren installeren in de OLOD Linux Advanced.

Jullie hebben toen daarvoor:

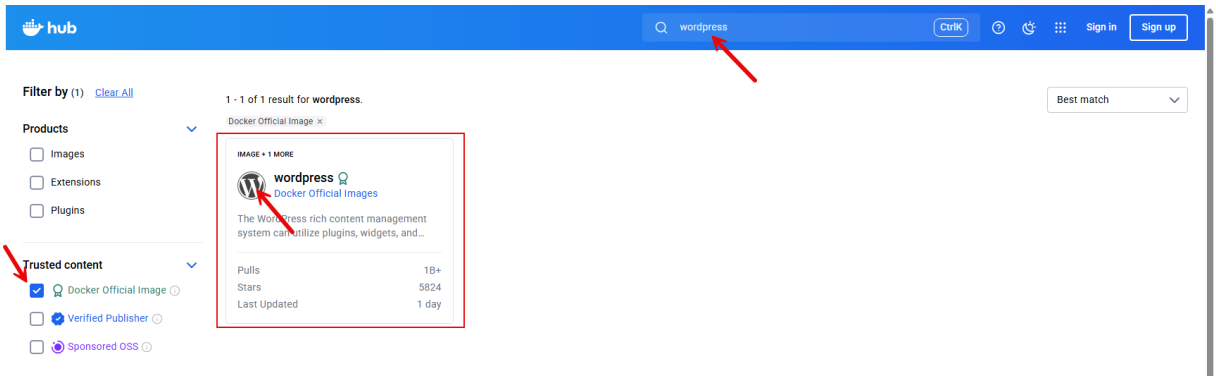
- Een webserver (httpd) geïnstalleerd en geconfigureerd
    - Httpd gestart.
    - Bestanden Wordpress geplaatst in /var/www/html.
    - Rechten en eigenaar aangepast van de Wordpress-bestanden.
- Een database management system geïnstalleerd en geconfigureerd.
    - Mariadb gestart.
    - Een database met een user en een wachtwoord aangemaakt.
- PHP geïnstalleerd en geconfigureerd.
- Wordpress geconfigureerd zodat die met de database overweg kan.

- Wp-config.php geconfigureerd voor database.

We gaan Wordpress installeren en configureren gebruik makend van containers.

We gaan nu gebruik maken van de officiële image van Wordpress op de Docker Hub: https://hub.docker.com.

Zoek bovenaan naar wordpress en selecteer links Docker Official Image.



Je ziet nu het officiële image van Wordpress staan. Klik erop.

Je vindt nu info over hoe je het image gebruikt. Hieronder vind je de voor ons belangrijkste regel.

How to use this image

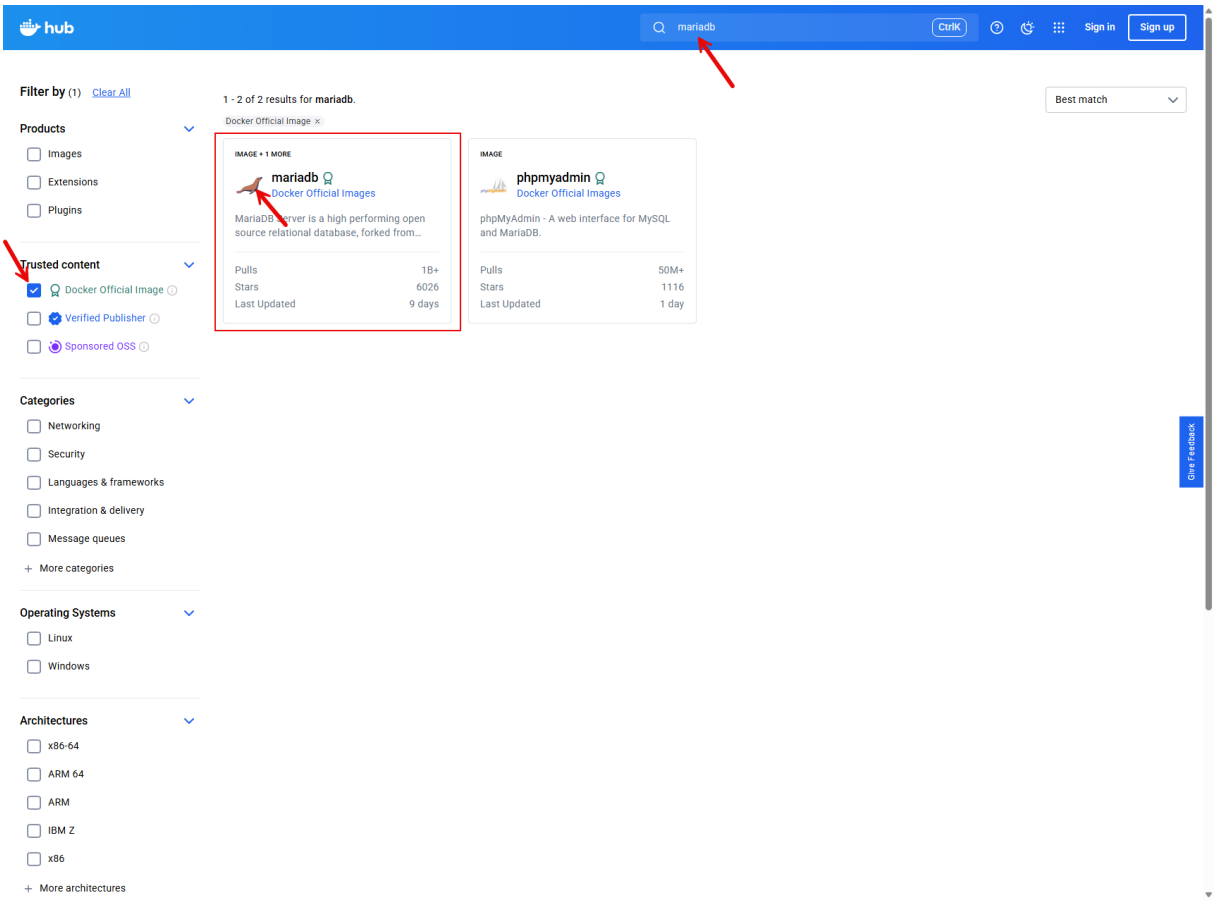
```
$ docker run --name some-wordpress --network some-network -d wordpress
```

Copy

- De officiële Wordpress Docker image bevat standaard PHP, omdat Wordpress in PHP is geschreven.
- De officiële Wordpress Docker image bevat wel geen databaseserver… De database wordt dus gebruikelijk in een aparte container gedraaid.

Ga terug naar de Docker Hub: https://hub.docker.com/.

Zoek nu naar mariadb.



Je ziet nu het officiële image van mariadb staan. Klik erop.

Je vindt nu info over hoe je het image gebruikt. Hieronder vind je de voor ons belangrijkste regel.

Start a `mariadb` server instance with user, password and database

Starting a MariaDB instance with a user, password, and a database:

```
$ docker run --detach --name some-mariadb --env MARIADB_USER=example-user --env MARIADB_PASSWORD=my_cool_secret --env MARIADB_DATABASE=exmple-database --env MARIADB_ROOT_PASSWORD=my-secret-pw mariadb:latest
```

Copy

6.4.2 Podman volumes aanmaken

Je wil natuurlijk niet dat je de gegevens van de database en de website verliest als een container zich afsluit. We maken hiervoor 2 named volumes aan voor de 2 containers die we zullen gebruiken.

```
student@serverXX:~$ podman volume create mysql-data

mysql-data

student@serverXX:~$ podman volume create wp-content

wp-content
```

6.4.3 Podman netwerk aanmaken

We willen uiteraard dat de 2 containers met elkaar kunnen communiceren. Hiervoor dien je, aangezien we rootless werken, een netwerk aan te maken. Dat is het meest betrouwbaar.

```
student@serverXX:~$ podman network create wpnet

wpnet
```

6.4.4 Container MariaDB

We zullen nu een container starten met MariaDB.

```
student@serverXX:~$ podman run –d ––name mariadb ––network wpnet –e
MARIADB_ROOT_PASSWORD='ServerXXdocker007' –e MARIADB_DATABASE='wordpress' –e
MARIADB_USER='wp' –e MARIADB_PASSWORD='ServerXXdocker1' –v mysql–
data:/var/lib/mysql:Z docker.io/library/mariadb:latest
9db3b5a582b291eeb216a8e5ac26c9402283aad7f751ca21b8654ee5d40a360e
```

-d zorgt ervoor dat de container detached start.

-- name mariadb geeft de container de naam mariadb.

--network wpnet verbindt de container met een bestaand netwerk dat wpnet heet.

-e is een environment variable.

MARIADB_ROOT_PASSWORD = wachtwoord van de databasebeheerder root.

MARIADB_DATABASE = er wordt automatisch een database gemaakt met de naam wordpress.

MARIADB_USER = er wordt een nieuwe gebruiker wp aangemaakt.

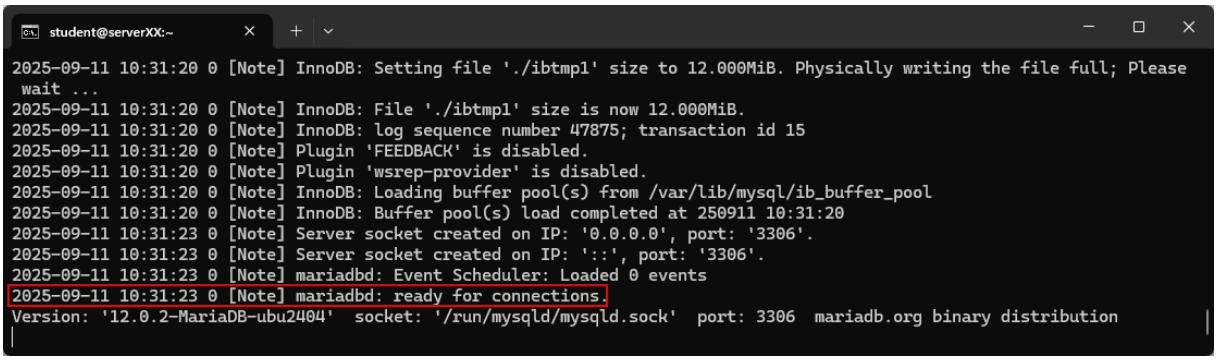
MARIADB_PASSWORD = wachtwoord van die gebruiker (wp).

De directory waarin de database bewaard wordt, wordt opgeslagen in mysql-data.

MariaDB wordt met andere woorden met het netwerk verbonden en er wordt een database aangemaakt met een gebruiker en wachtwoord die daarvan gebruik mogen maken.

Ter controle kan je kijken of MariaDB connecties aanvaardt.

```
student@serverXX:~$ podman logs –f mariadb
```



Ga eruit door <CTRL> + C te typen.

De container blijft draaien zolang mariadb draait.

```
student@serverXX:~$ podman ps
CONTAINER ID  IMAGE                                COMMAND …
9db3b5a582b2  docker.io/library/mariadb:latest  mariadbd …
```

6.4.5 Container Wordpress

We starten nu een container op met Wordpress. Aangezien we rootless werken publiceren we de poort naar 8080 op de host. We stellen ook de verbinding met de databaseserver in.

```
student@serverXX:~$ podman run -d --name wordpress --network wpnet -p 8080:80
-e WORDPRESS_DB_HOST='mariadb:3306' -e WORDPRESS_DB_USER='wp' -e
WORDPRESS_DB_PASSWORD='ServerXXdocker1' -e WORDPRESS_DB_NAME='wordpress' -v
wp-content:/var/www/html/wp-content:Z docker.io/library/wordpress:latest
…
143026fcf4b0516884bb68e014bc4131c23514d94496529e98c1c4c13f8c7700
```

-d zorgt ervoor dat de container detached start.
-- name some-wordpress geeft de container de naam wordpress.
--network wpnet verbindt de container met een bestaand netwerk dat wpnet heet.
-e is een environment variable.
Je ziet dat de gegevens over de database hier terugkomen.
De website wordt opgeslagen in het named volume wp-content.

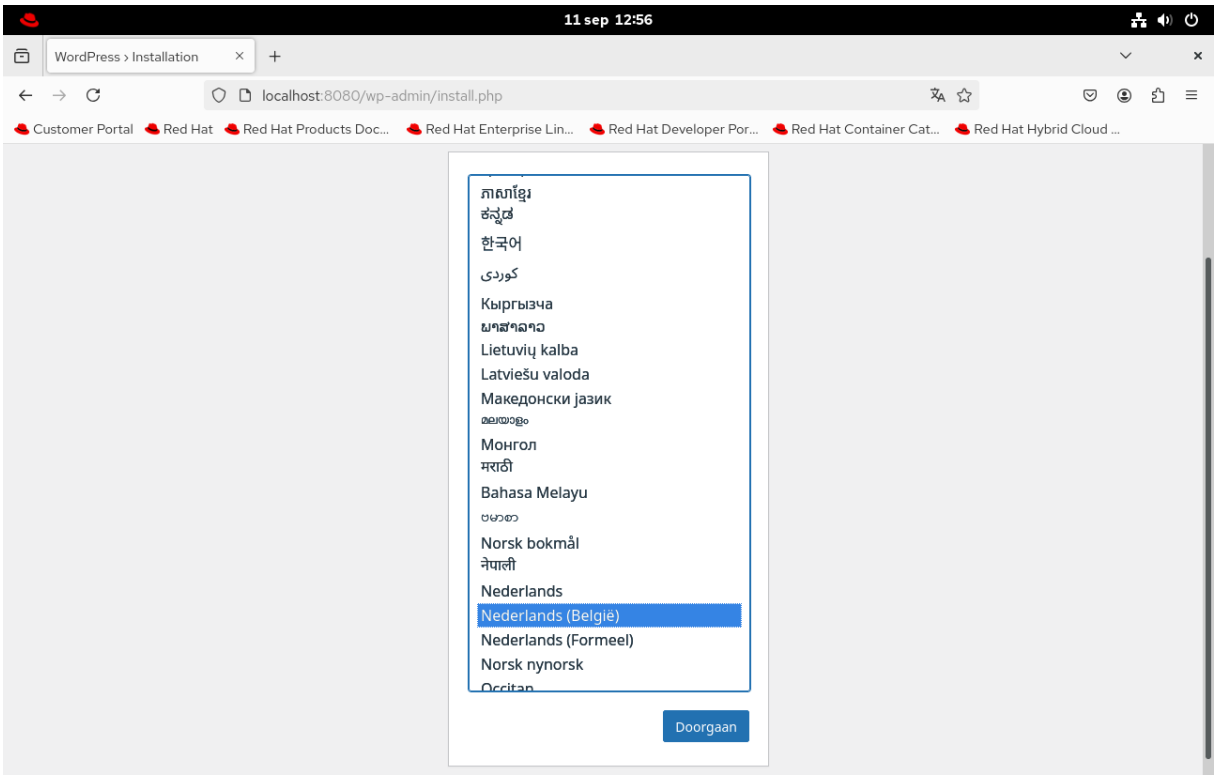
We kunnen al een kleine test doen of de website beschikbaar is.

```
student@serverXX:~$ netstat -tlpn | grep 8080
…
tcp6       0      0 :::8080      :::*         LISTEN      5279/rootlessport
```

Er is dus een website beschikbaar op poort 8080.

6.4.6 Website Wordpress configureren

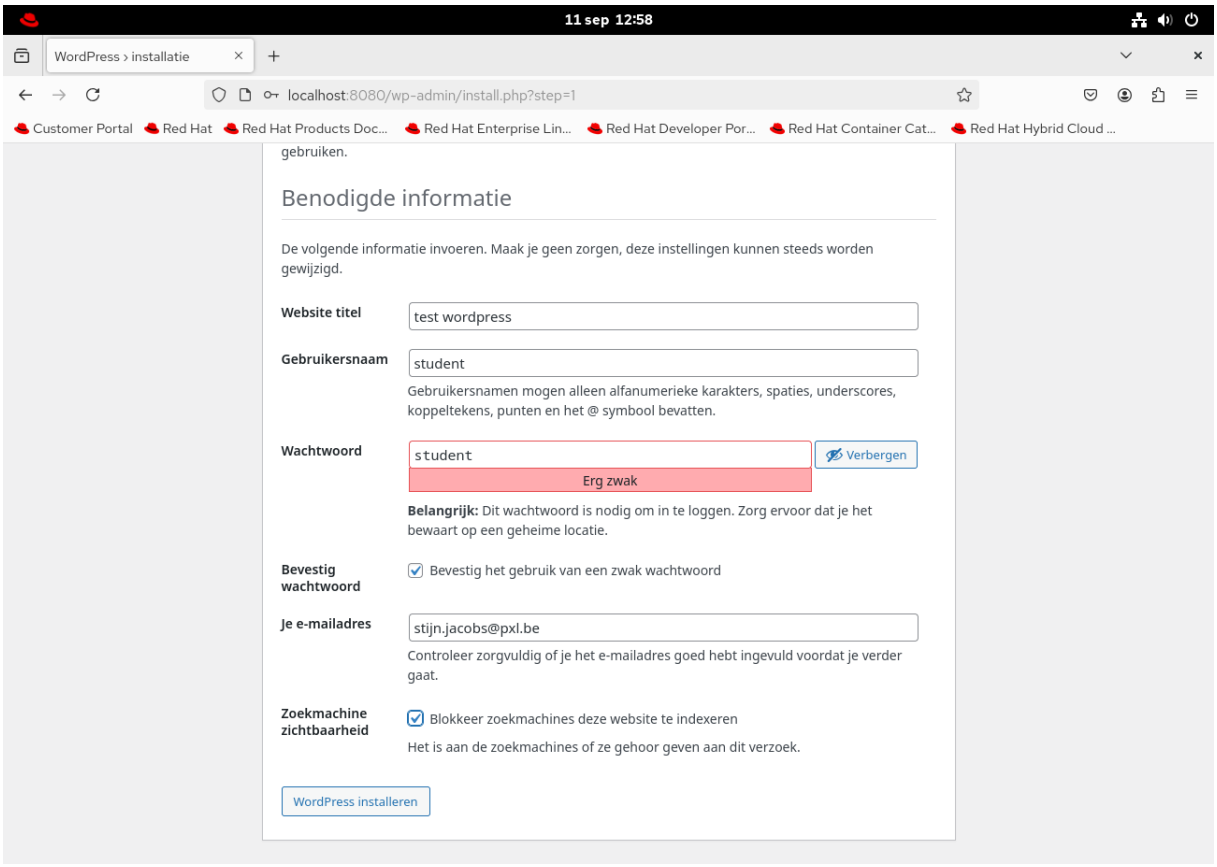
Ga naar de browser van ServerXX en typ http://localhost:8080 in.
Kies in het eerste vensster voor Nederlands (België) zoals je hieronder ziet.



Klik erna op Doorgaan.

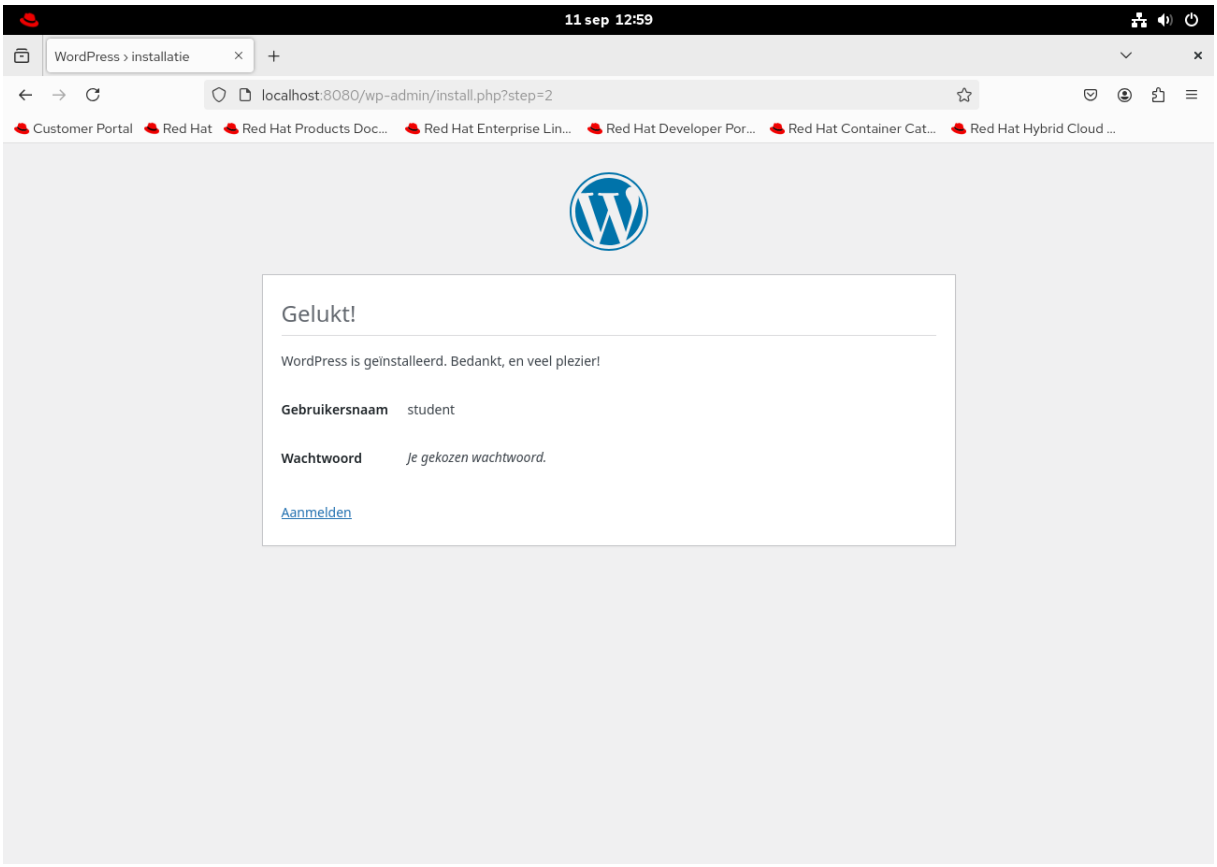
Je kan nu onmiddellijk de info over de website ingeven die je wenst. De database is immers al gekoppeld.

Vul nu gegevens in analoog met onderstaande.

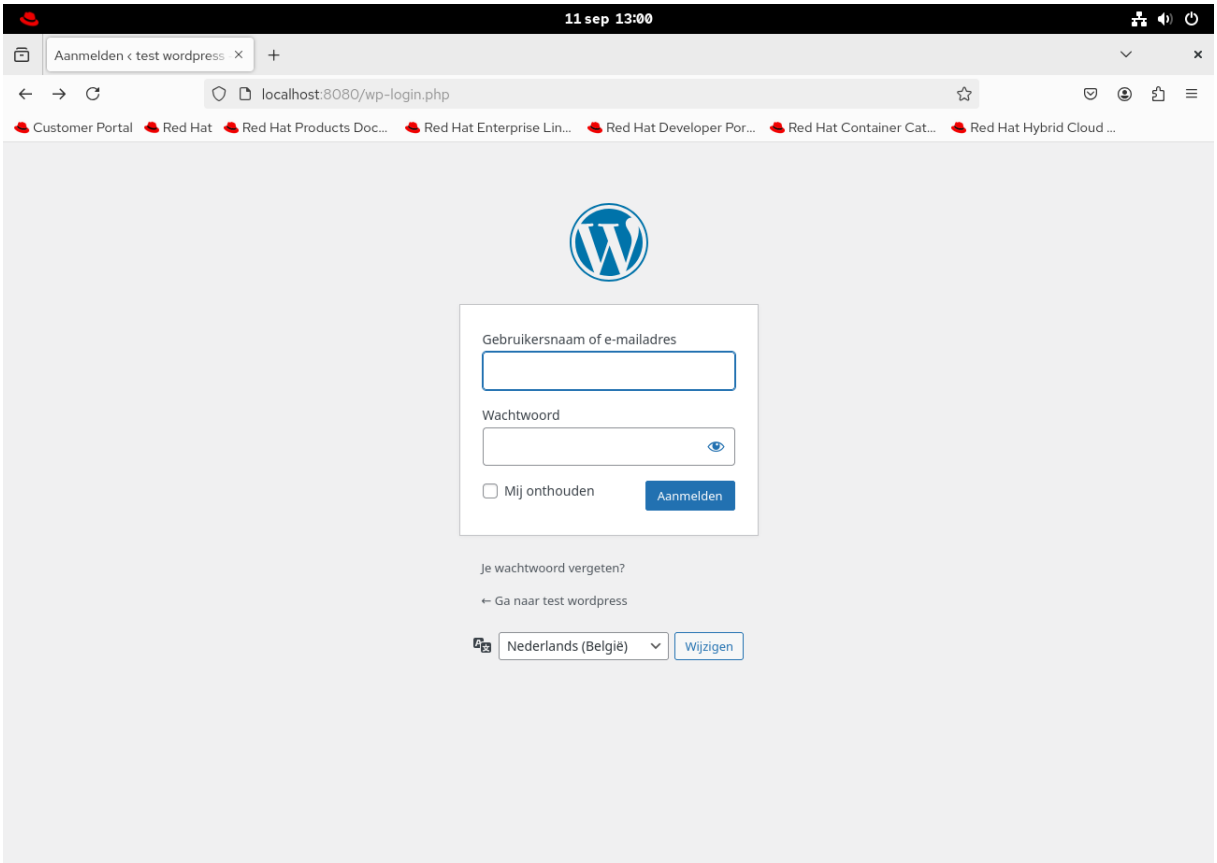


Klik erna uiteraard op WordPress installeren.

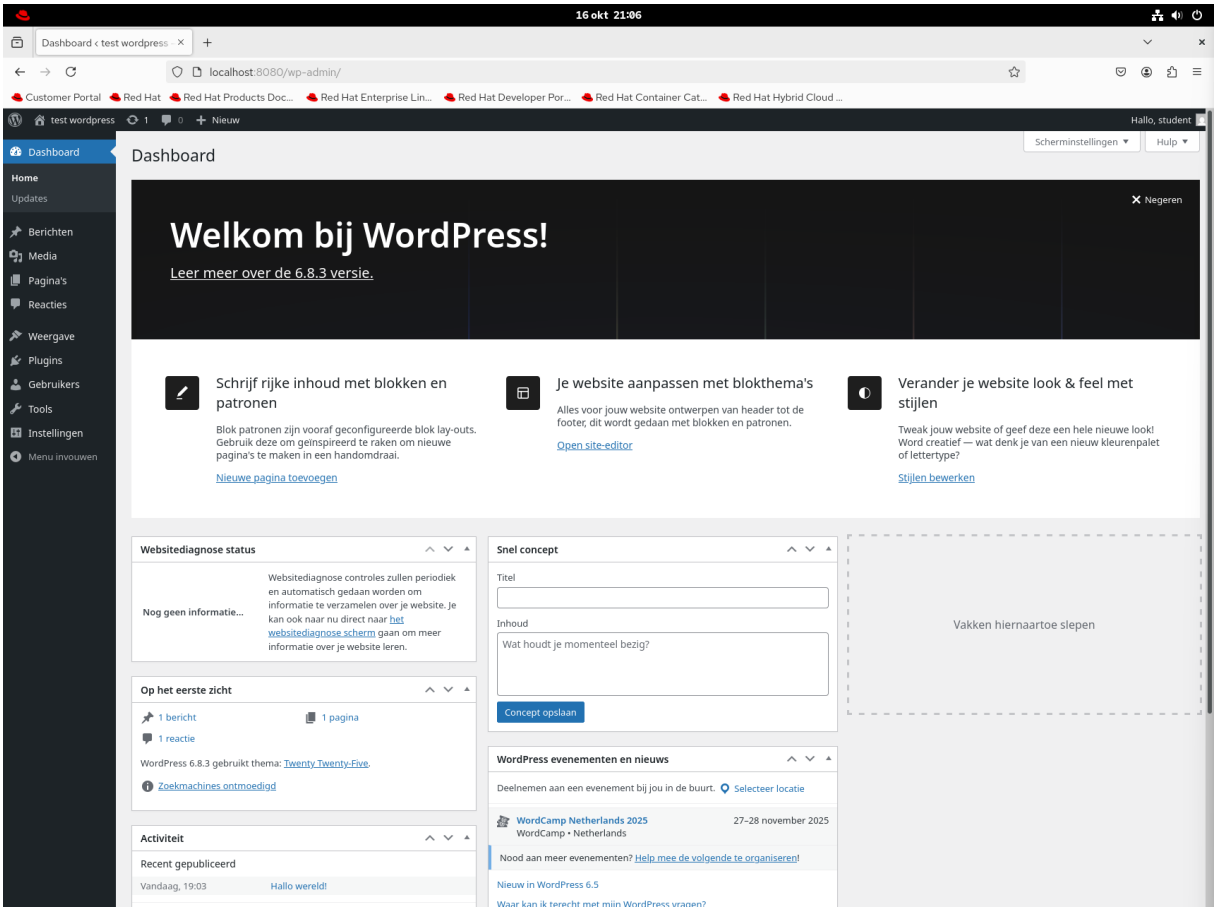
Je krijgt nu de melding dat WordPress is geïnstalleerd.



Klik uiteraard op Aanmelden.



Je kan de website beheren door eerst de Gebruikersnaam of e-mailadres en wachtwoord in te geven.



Je kan nu naar de website zelf gaan door rechtsboven te klikken op “test wordpress”, website bezoeken te klikken.

Je kan uiteraard ook http://localhost:8080 of http://192.168.112.100:8080/ intypen.
De configuratie van de website is altijd bereikbaar via http://localhost:8080/wp-admin/.

Verder gaan we hier niet op in. Dat is voor andere collega’s in de ICT-afdeling.

6.1 Tips

Elke officiële image heeft een uitgebreide beschrijving.

Daar vind je info over o.a.:

- Omgevingsvariabelen (zoals MARIADB_ROOT_PASSWORD, WORDPRESS_DB_HOST).
- Standaard poorten (bijv. MariaDB = 3306, WordPress/Apache = 80).
- Volumes (waar je data kan mounten).