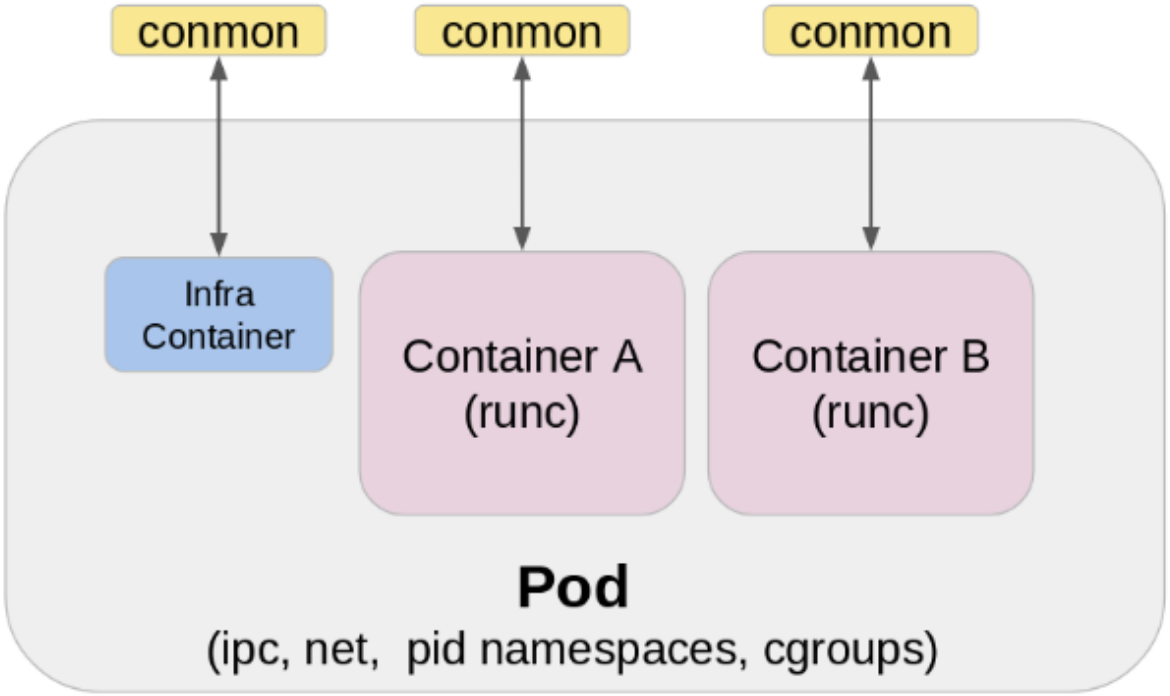
9 Podman pods

9.1 Inleiding

Containers zijn de kleinste eenheden die u kunt beheren met de containertools van Podman.

Een Podman-pod is een groep van een of meer containers. Elke Podman-pod bevat altijd een infracontainer.



Deze container is niet bedoeld om applicaties te draaien, maar fungeert als een technische ruggengraat voor de andere containers in de pod. De infra-container in Podman is een soort stille kracht: hij doet zelf niets functioneels, maar is essentieel voor het functioneren van een pod. Hier is een overzicht van zijn belangrijkste functies (leer dit zeker niet van buiten):

Functie	**Uitleg**
Namespacebeheer	Beheert de kernel namespaces voor de pod: netwerk, PID, IPC, mount enz.
Poortbindingen	Alle poortbindingen (zoals 8080:80) worden aan de infra-container gekoppeld.
IPC-namespace	Zorgt dat containers in de pod IPC-objecten kunnen delen (shared memory, semaforen).
PID-namespace	Maakt het mogelijk dat containers elkaars processen kunnen zien (bijv. voor monitoring).
Netwerkinterface	Verzorgt de gedeelde netwerkstack voor alle containers in de pod.

cgroup-parent toewijzing	Regelt resourcebeheer via cgroups voor de hele pod.
Pod persistentie	Houdt de pod “levend” zelfs als alle andere containers stoppen.
Pause-functionaliteit	Draait een ‘pauze’-proces dat letterlijk niets doet, maar de namespaces actief houdt.
Beveiliging en isolatie	Zorgt voor consistente isolatie tussen pods en de rest van het systeem.

Samenvattend gezegd:

Een pod in Podman is een groep containers die samenwerken alsof het één toepassing is.

Alle containers in een pod delen hetzelfde netwerk, opslag (volumes) en resource-instellingen.

PS Podman-pods zijn vergelijkbaar met de Kubernetes-definitie. Hierop komen we later terug.

9.2 Werken met pods

Hier maken we een lege pod aan.

```
student@serverXX:~$ podman pod create --name mypod

a518b698331a040cfb62308dad93e56a415bbe9f81d85f8e0407c91377e9f6d6
```

De pod bevindt zich in de beginstatus 'Gemaakt'.

We listen nu alle pods.

```
student@serverXX:~$ podman pod ps

POD ID        NAME        STATUS      CREATED         INFRA ID      # OF
CONTAINERS

a518b698331a  mypod       Created     55 seconds ago  3136cbd2b750  1
```

Zoals je ziet is er 1 container: dat is de infra-container.

De container heeft als doel:

- het netwerk en de namespaces van de pod “vasthouden”;
- een stabiele PID 1 leveren in de pod;
- te zorgen dat de pod blijft bestaan, ook als er (nog) geen andere containers draaien.

In een gewone container is PID 1 het proces dat bepaalt of de container “leeft”. Als dat proces stopt, stopt de container zoals we al besproken hebben.

Je kan ook een lijst opvragen van alle pods en de bijhorende containers.

```
student@serverXX:~$ podman ps -a --pod

CONTAINER ID  IMAGE                                         …        PODNAME
```

```
3136cbd2b750  localhost/podman-pause:5.4.0-1750809600     …        mypod
```

De standaard infracontainer is gebaseerd op de registry.access.redhat.com/ubi10/pause image.

We zullen nu een container met de naam myubi in de bestaande pod mypod uitvoeren.

```
student@serverXX:~$ podman run -dt --name myubi --pod mypod
registry.access.redhat.com/ubi10/ubi /bin/bash
```

We listen nu terug alle pods.

```
student@serverXX:~$ podman pod ps

POD ID        NAME        STATUS      … # OF CONTAINERS

a518b698331a  mypod       Running     … 2
```

Zoals je ziet heeft de pod mypod nu 2 containers.

We zullen nu containers listen met info over pod.

```
student@serverXX:~$ podman ps -a --pod

CONTAINER ID  IMAGE  …                                                PODNAME

3136cbd2b750  localhost/podman-pause:5.4.0-1750809600 …                mypod

6f297a9b39d3  registry.access.redhat.com/ubi10/ubi:latest  /bin/bash…  mypod
```

Meer info over het aanmaken van een pod in de manual-page.

```
student@serverXX:~$ man podman-pod-create
```

Om de actieve processen van containers in een pod weer te geven, voert u het volgende in:

```
student@serverXX:~$ podman pod top mypod

USER …        %CPU        ELAPSED         TTY         TIME        COMMAND

0 …           0.000       6m5.964458803s  ?           0s          /catatonit -P

Root …        0.000       6m5.966108961s  pts/0       0s          /bin/bash
```

Om resourcestatistieken te zien van containers in een op meer pods gebruik je onderstaande:

```
student@serverXX:~$ podman pod stats -a --no-stream

POD           CID           NAME                CPU %       MEM USAGE/ LIMIT …

a518b698331a  3136cbd2b750  a518b698331a-infra  0.00%       49.15kB / 3.795GB …

a518b698331a  6f297a9b39d3  myubi               0.01%       647.2kB / 3.795GB …
```

Je kan een pod ook inspecteren.

```
student@serverXX:~$ podman pod inspect mypod

…
```

Hier zie je dat er 2 containers in de pod mypod zijn.

Stop nu de pod mypod.

```
student@serverXX:~$ podman pod stop mypod

WARN[0010] StopSignal SIGTERM failed to stop container myubi in 10 seconds,
resorting to SIGKILL

mypod
```

We maken nu een lijst van alle pods met de bijhorende containers.

```
student@serverXX:~$ podman ps –a ––pod

…
```

Je kan zien dat de pod mypod en de container myubide status 'Verlaten' (exited) hebben.

Je kan één of meer gestopte pods en containers verwijderen met de podman pod rmopdracht.

```
student@serverXX:~$ podman pod rm mypod

a518b698331a040cfb62308dad93e56a415bbe9f81d85f8e0407c91377e9f6d6
```

Houd er rekening mee dat wanneer u de pod verwijdert, ook alle containers die zich daarin bevinden, worden verwijderd!

We controleren of alle containers van de pod en de pod verwijderd zijn.

```
student@serverXX:~$ podman ps –a

CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES

student@serverXX:~$ podman pod ps

POD ID      NAME        STATUS      CREATED     INFRA ID    # OF CONTAINERS
```

Opmerking: Aan podman pod ps kan je de optie -a niet toevoegen! Met podman pod ps toen je zowel actieve als gestopte pods.

9.3 Wordpress

9.3.1 Zonder gegevens op ServerXX

We hebben Wordpress tot nu toe op 2 keer verschillende manieren geïnstalleerd:

- Via CLI 2 containers en een user-defined netwerk opgezet
- Via podman compose

Nu zullen we een derde manier behandelen via een pod.

De belangrijkste voordelen hiervan:

- Eén gedeelde netwerknamespace
  Containers in dezelfde pod praten rechtstreeks met elkaar via localhost en gedeelde poorten. Je hoeft dus géén apart user-defined netwerk meer te maken of linken.
- Eenvoudiger lifecycle management
  Veel handiger dan containers individueel beheren.
- Infra-container houdt de pod stabiel
  Zoals reeds besproken: de pod blijft actief zolang de infra-container draait, ook als WordPress of de database herstart.
- Dichter bij Kubernetes-concepten
  Pods zijn het basisconcept van Kubernetes. Werken met Podman pods helpt dus om ervaring op te doen met hoe Kubernetes applicaties structureert. En jawel, dat zien we binnenkort!

We maken nu eerst de pod aan.

```
student@serverXX:~$ podman pod create --name wp-pod -p 8080:80
```

De pod heeft, zoals je verwacht, als naam wp-pod.

We maken hier ook een poortverbinding. Poort 8080 op de host wordt verbonden met poort 80 in de pod. De poortbinding wordt toegewezen aan de infra-container van de pod. Alle containers in de pod delen dezelfde netwerkstack, dus ze kunnen intern communiceren en zijn extern bereikbaar via die ene poort.

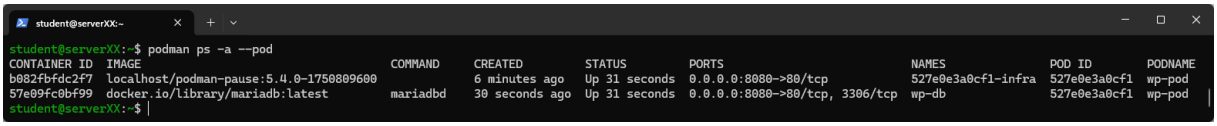
We starten nu een container in de pod op van mariadb. In feite komt dat overeen met hetgeen we al gedaan hebben. Alleen zie je dat de container in de pod wp-pod wordt gestart.

```
student@serverXX:~$ podman run -d --restart=always --pod=wp-pod -e
MYSQL_ROOT_PASSWORD="dbpass" -e MYSQL_DATABASE="wp" -eMYSQL_USER="wordpress" -e
MYSQL_PASSWORD="wppass" --name=wp-db mariadb

57e09fc0bf994b418a68857d5521958a877024b7ca1a86186a6b08d348063a84
```

We kijken nu naar de containers die draaien in deze pod.

```
student@serverXX:~$ podman ps -a --pod
```

…



Zoals je ziet draait de infra-container en de mariadb-container.

We starten nu ook de wordpress-container op gekoppeld aan dezelfde pod.

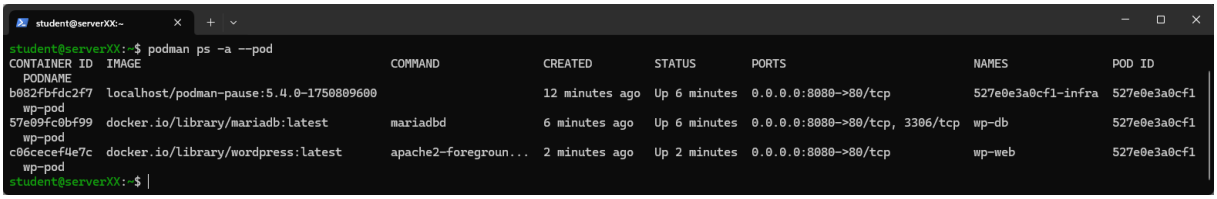
```
student@serverXX:~$ podman run -d --restart=always --pod=wp-pod -e
WORDPRESS_DB_NAME="wp" -e WORDPRESS_DB_USER="wordpress" -e
WORDPRESS_DB_PASSWORD="wppass" -e WORDPRESS_DB_HOST="127.0.0.1" --name wp-web
wordpress

c06cecef4e7ce1317c04fc310dd5fd8aca0443b14488ea0f6ed5f0455cbec2fe
```

Je ziet hier dat, zoals beschreven hierboven, er connectie kan gemaakt worden met Wordpress via localhost (127.0.0.1).

We kijken nu naar de containers die draaien in deze pod.

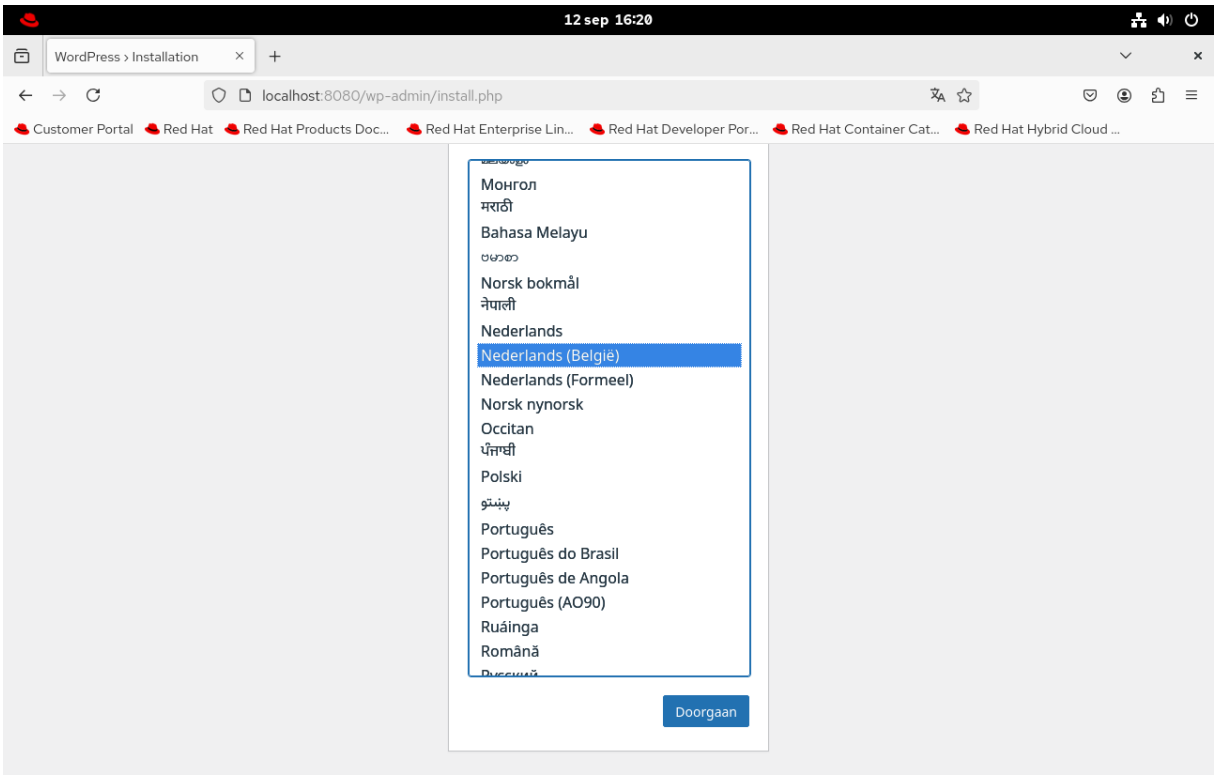
```
student@serverXX:~$ podman ps -a --pod
```

…



Je ziet nu dat de 3 containers draaien.

Je kan nu naar de browser gaan en de nodige instellingen doen zoals we al enkele keren gedaan hebben.



Na je test verwijderen we de pod nu met de bijhorende containers.

```
student@serverXX:~$ podman pod rm wp-pod -f

527e0e3a0cf164b224503feea89965f6457d507d8a0c313117a401b1dae6b7e0
```

9.3.2 Met gegevens op ServerXX

Wat we zojuist gedaan hebben is niet erg verstandig omdat de gegevens niet worden opgeslagen op de host zelf maar in de container.

- Als je geen volumes koppelt, worden gegevens in de container zelf opgeslagen en gaan verloren als je de container verwijdert. De gegevens worden nooit in de pod opgeslagen.
- Als je named volumes of bind volumes gebruikt (zoals we nu gaan doen), worden gegevens op de host opgeslagen, maar toegewezen aan een specifieke container.

We maken nu eerst de mappen aan op de host.

```
student@serverXX:~$ mkdir -p wpdata/{html,db}
```

We maken nu weer de pod aan.

```
student@serverXX:~$ podman pod create --name wp-pod -p 8080:80

f5fe87b906b6dcdef8e833a1cd379fa29c5cdf25b426c52e18b5db36dbb4466d
```

We starten nu de MariaDB-container met het bind mount volume op.

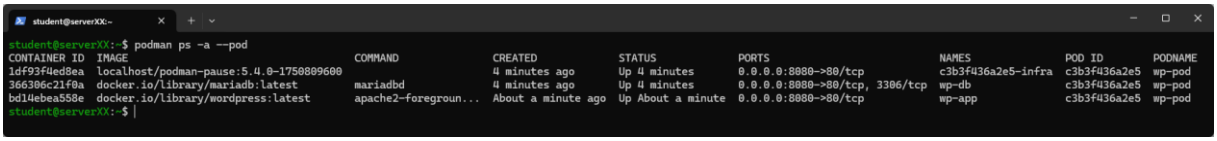
```
student@serverXX:~$ podman run -d   --pod wp-pod   --name wp-db   -e
MYSQL_ROOT_PASSWORD="admin123"   -e MYSQL_DATABASE="wordpress"   -e
MYSQL_USER="wpuser"   -e MYSQL_PASSWORD="user123"   -v
/home/student/wpdata/db:/var/lib/mysql:Z   docker.io/library/mariadb:latest

366306c21f0a7a4b0c2e582a91380ead1194a78f88a1f5082d2a0f478435700a
```

We starten nu de Wordpress-container met het gekoppeld volume.

```
student@serverXX:~$ podman run -d --pod wp-pod --name wp-app -e
WORDPRESS_DB_HOST="127.0.0.1" -e WORDPRESS_DB_NAME="wordpress" -e
WORDPRESS_DB_USER="wpuser" -e WORDPRESS_DB_PASSWORD="user123" -v
$HOME/wpdata/html:/var/www/html:Z docker.io/library/wordpress:latest

bd14ebea558e413f0f67ebb366f3204ba2309d8747fa537f23a635e898d9fef6
```

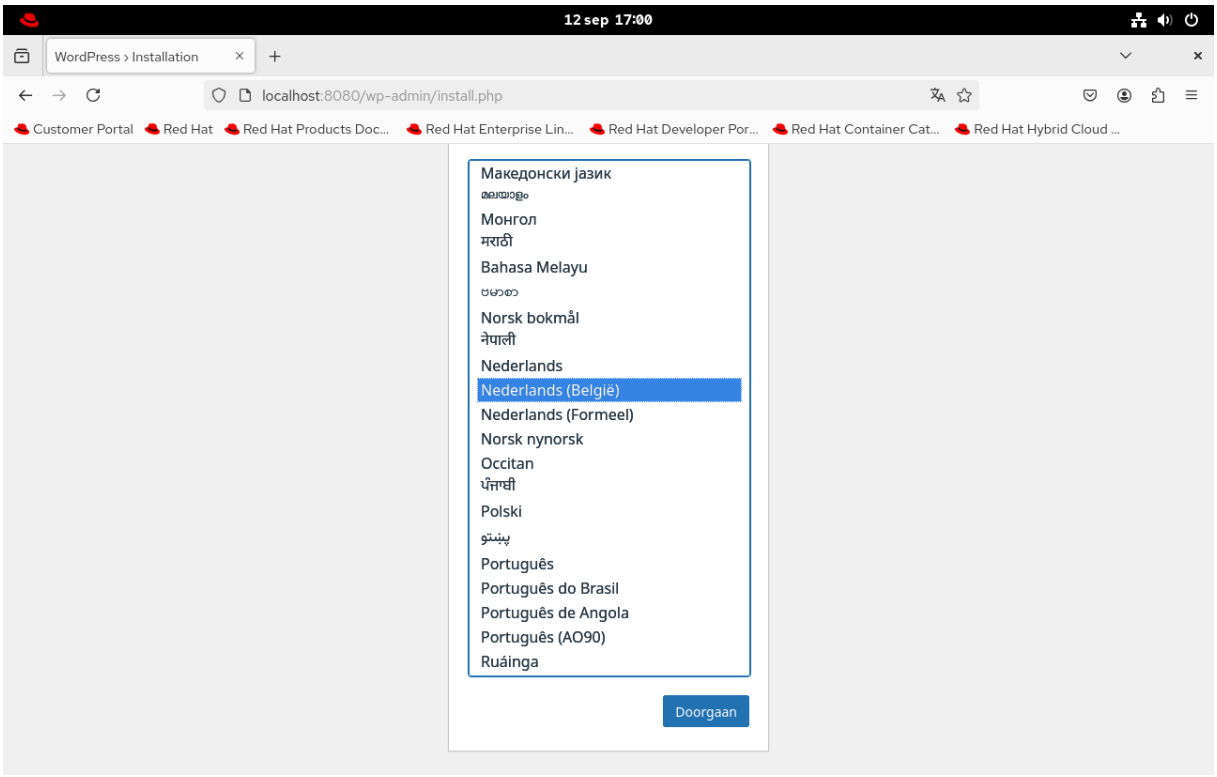
We kijken nu naar de containers die draaien in deze pod.

```
student@serverXX:~$ podman ps -a --pod

…
```



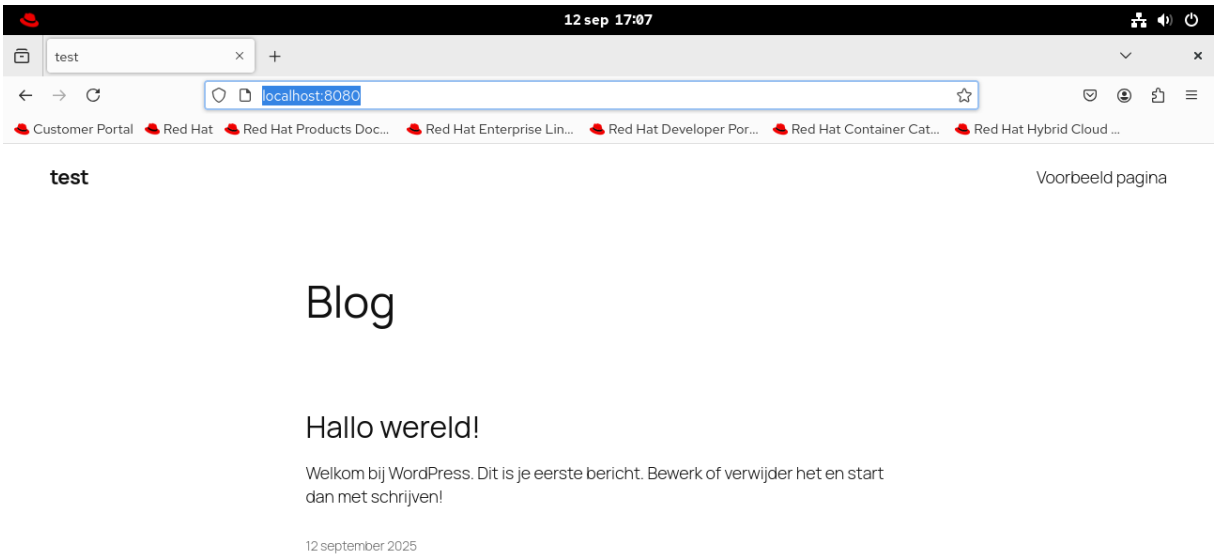
Je ziet nu dat de 3 containers draaien.

Je kan nu naar de browser gaan en de nodige instellingen doen zoals we al enkele keren gedaan hebben.



Stel in zoals gebruikelijk.

Na instellen krijg je dit bijvoorbeeld.



We stoppen en verwijderen nu de pod.

```
student@serverXX:~$ podman pod stop wp-pod

wp-pod

student@serverXX:~$ podman pod rm wp-pod -f
```

Het mooie is nu dat wanneer de pod verwijderd is je nog steeds terug een nieuwe pod kan opbouwen met de zelfde bind mounts en de website voorhanden blijft. Je moet dus niets meer opnieuw instellen omdat de database en website op ServerXX staan.

```
student@serverXX:~$ podman pod create --name wp-pod2 -p 8080:80

7ea8e4f8784a52e847024f57947b367e4f95bb4c3f4b20d73195d7a6d062ca0b

student@serverXX:~$ podman run -d   --pod wp-pod2   --name wp-db   -e
MYSQL_ROOT_PASSWORD="admin123"   -e MYSQL_DATABASE="wordpress"   -e
MYSQL_USER="wpuser"   -e MYSQL_PASSWORD="user123" -v
/home/student/wpdata/db:/var/lib/mysql:Z   docker.io/library/mariadb:latest

08360a21128d5d20797fe20cae0fdb019eb1806b4786ccc255c094e46addced1

student@serverXX:~$ podman run -d --pod wp-pod2 --name wp-app -e
WORDPRESS_DB_HOST="127.0.0.1" -e WORDPRESS_DB_NAME="wordpress" -e
WORDPRESS_DB_USER="wpuser" -e WORDPRESS_DB_PASSWORD="user123" -v
$HOME/wpdata/html:/var/www/html:Z docker.io/library/wordpress:latest

93d67641b7ff74a7283e8d3172d3744442aa2a8a6dc66dd31352e5505d9fadcb
```

Als je nu terug gaat naar http://localhost:8080 krijg je terug dezelfde website zonder dat je dus nog moet instellen.