

## 5 Containers netwerken en data volumes

### 5.1 Inleiding

In dit hoofdstuk bekijken we hoe netwerken werken in Podman, en hoe je standaard- en aangepaste netwerken kunt opzetten. We leren hoe je poorten publiceert en services blootstelt, zodat containers zowel onderling als met externe systemen kunnen communiceren.

Daarnaast bekijken we hoe je volumes koppelt aan containers voor het opslaan en beheren van gegevens – zowel lokaal als extern.

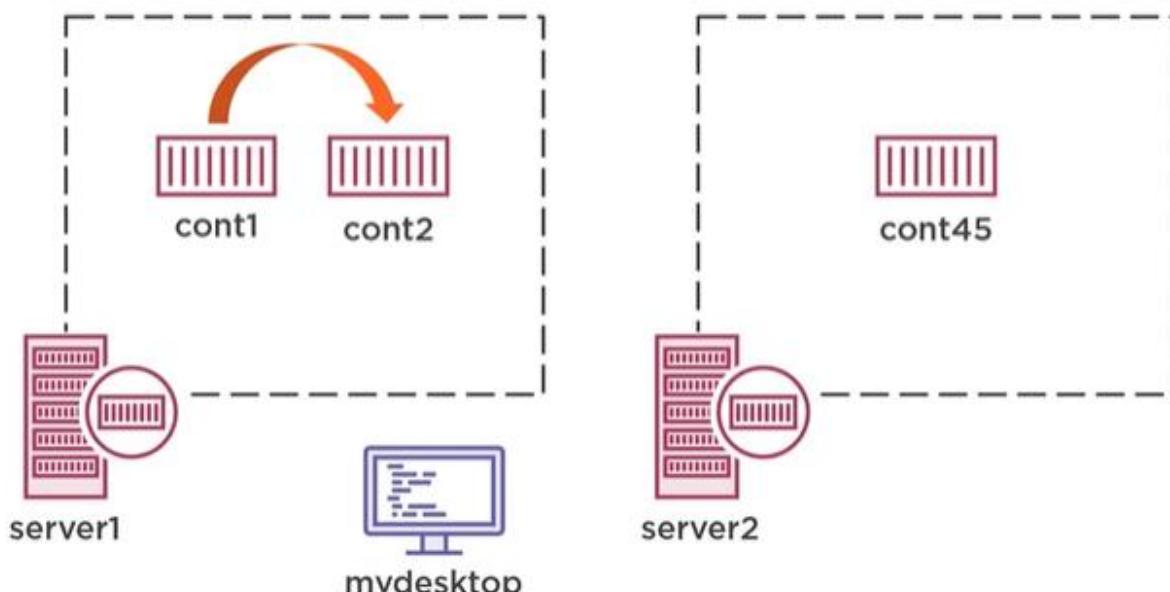
### 5.2 Container netwerken

#### 5.2.1 Inleiding

In een productieomgeving heb je vaak meerdere containerhosts (servers waarop containers draaien). Applicaties in containers moeten bereikbaar zijn voor gebruikers en andere services. Het is dus cruciaal om de netwerkconfiguratie correct op te zetten.

Hier overlopen we theoretisch de verschillende mogelijkheden.

#### 5.2.2 Eerste mogelijkheid: Twee Containers op de Zelfde Host



In dit scenario bevinden twee containers zich op dezelfde host en communiceren zij via een gedeeld netwerk, zoals een NAT-netwerk.

Voordelen:

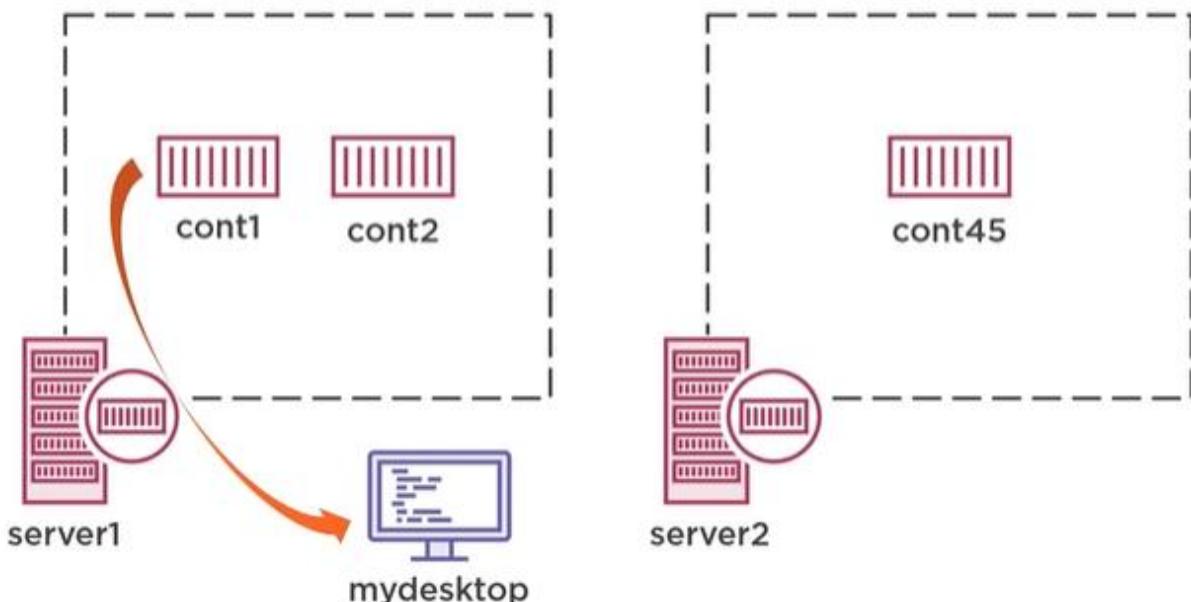
- Lage latentie omdat beide containers op dezelfde fysieke host draaien.

- Eenvoudige configuratie doordat beide containers dezelfde netwerkomgeving delen.

Voorbeeldconfiguratie:

Stel, er zijn twee containers, Cont1 en Cont2, die draaien op server1.

### 5.2.3 Tweede mogelijkheid: Een Container Communiceert met een Externe Service



In dit scenario moet een container verbinding maken met een service buiten de containeromgeving, bijvoorbeeld een externe database of een API-service.

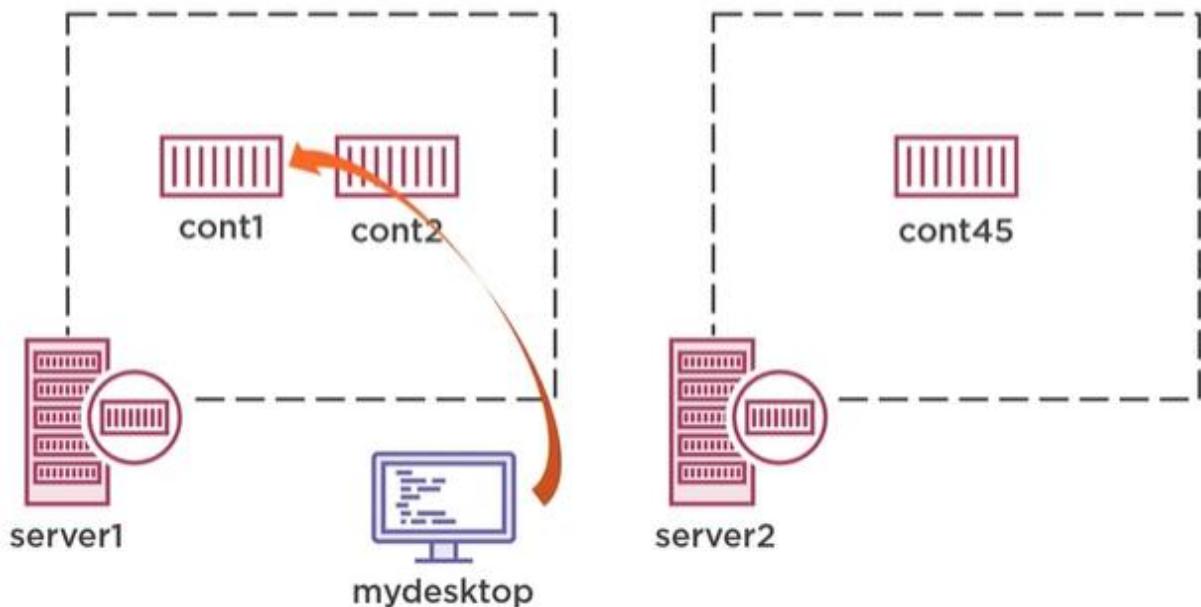
Voordelen:

- Biedt de mogelijkheid om externe diensten te integreren in een containergebaseerde toepassing.
- Flexibiliteit om containers te verbinden met verschillende externe services.

Voorbeeldconfiguratie:

Stel, Cont1 op Server1 moet verbinding maken met een externe database op DatabaseServer op mydesktop.

#### 5.2.4 Derde mogelijkheid: Een Externe Omgeving Communiceert met een Container



Hier moet een externe client, bijvoorbeeld een desktop-pc of een ander systeem, verbinding maken met een service die draait in een container, zoals een webapplicatie.

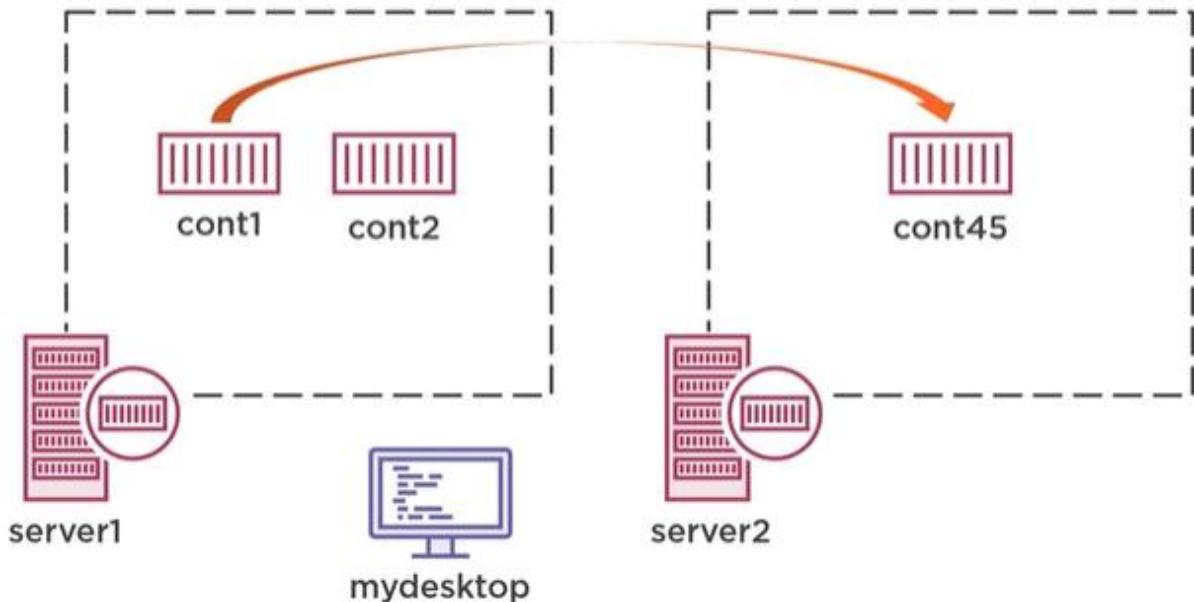
Voordelen:

- Externe systemen kunnen gemakkelijk toegang krijgen tot de services binnen containers.
- Maakt het mogelijk om containers te gebruiken als volwaardige servers voor externe gebruikers.

Voorbeeldconfiguratie:

Wanneer een externe gebruiker toegang wil krijgen tot een webserver die draait binnen Cont1.

### 5.2.5 Vierde mogelijkheid: Communicatie tussen Containers op Verschillende Hosts



Dit scenario wordt gebruikt in een multi-server-omgeving, waarbij containers op verschillende hosts met elkaar moeten communiceren.

Voordelen:

- Schaalbaarheid: meerdere hosts kunnen samen één logische netwerkomgeving creëren.
- Netwerkvirtualisatie: containers op verschillende hosts kunnen eenvoudig communiceren zonder dat er complexe routering nodig is.

Voorbeeldconfiguratie:

Stel dat Cont1 op Server1 moet communiceren met Cont45 op Server2.

## 5.3 Rootless netwerktoegang

### 5.3.1 Vooraf

We zullen eerst alle Podman-containers te verwijderen om het overzicht te bewaren.

```
student@serverXX:~$ podman rm --all -f  
...
```

Na het uitvoeren van deze stap kan je controleren of er geen containers meer aanwezig zijn door de lijst van containers weer te geven:

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

CONTAINER	ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES

### 5.3.2 Check instellingen podman

Rootless betekent dat de containers draaien zonder rootrechten op de host. Daarom gebruikt Podman speciale user-space tools: vroeger slirp4netns, nu pasta.

We checken eerst de versie van podman.

```
student@serverXX:~$ podman version

Client:          Podman Engine
Version:         5.4.0
API Version:    5.4.0
Go Version:     go1.23.10 (Red Hat 1.23.10-1.el10_0)
Built:           Wed Jun 25 02:00:00 2025
Build Origin:   Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
OS/Arch:        linux/amd64
```

Met onderstaande kan je zien hoe podman geconfigureerd is betreffende rootless netwerk.

```
student@serverXX:~$ podman info --debug | grep rootless

rootlessNetworkCmd: pasta
rootless: true
```

Wij gebruiken dus pasta voor rootless netwerken. Pasta is een high performance opvolger van slirp4netns.

Verschil tussen past en slirp4netns:

Hieronder vind je de verschillen tussen pasta en slirp4netns in rootless Podman. Leer dit niet van buiten. Misschien is het niet heel duidelijk voor je. We leren pasta in de volgende paragrafen kennen.

- IP-adres
  - o pasta: container gebruikt hetzelfde IP-adres als de host.
  - o slirp4netns: container krijgt een eigen virtueel IP (bijv. 10.0.2.100).
- Gateway/NAT
  - o pasta: gebruikt de host-gateway, geen NAT.
  - o slirp4netns: maakt eigen gateway en doet NAT.
- Interface
  - o pasta: neemt de host-interface over (bijv. ens160).
  - o slirp4netns: creëert een virtuele tap0 interface.
- IPv6
  - o pasta: ondersteunt IPv6-port forwarding.

- slirp4netns: geen IPv6-forwarding.
- Performance
  - pasta: efficiënter en sneller.
  - slirp4netns: meer overhead, trager.

### 5.3.2.1 Uitbreiding: slirp4netns instellen

Je kan dit instellen voor alle gebruikers in /usr/share/containers/containers.conf.

```
student@serverXX:~$ sudo nano /usr/share/containers/containers.conf
```

...

```
#default_rootless_network_cmd = "pasta"
```

...

Stel dat je slirp4netns wil gebruiken dan verander je dit in.

...

```
default_rootless_network_cmd = "slirp4netns"
```

...

In RHEL 10 is slirp4netns verouderd omdat pasta nu de standaard is voor rootless netwerken.  
Installatie kan nog als volgt:

```
student@serverXX:~$ sudo dnf install slirp4netns
```

### 5.3.3 Standaard rootless netwerk

#### 5.3.3.1 Communicatie container met internet

Als je geen netwerk specificeert wordt de rootless-netwerkstack gebruikt.

We starten één container op.

```
student@serverXX:~$ podman run -d --name C1 --hostname C1 ubi10/ubi-init
c7761e4e2b17ae79f7dec2d5b332edc92cea9858ab8c743dac174d0f0f9587cd
```

We controleren of de container gestart is.

```
student@serverXX:~$ podman ps
CONTAINER ID  IMAGE                                     COMMAND ...
c7761e4e2b17  registry.access.redhat.com/ubi10/ubi-init:latest /sbin/init ...
```

We gaan in de container:

```
student@serverXX:~$ podman exec -it C1 bash  
[root@C1 /]#
```

In de container installeren we essentiële netwerktools.

```
[root@C1 /]# dnf install -y iputils iproute curl  
...
```

We zullen nu het netwerk testen.

We checken of je ICMP echo-packages met ping (in package iputils) kan sturen naar en ontvangen van www.google.be.

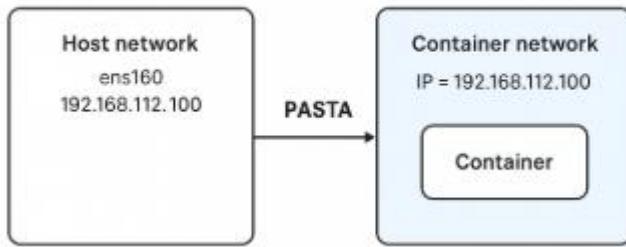
```
[root@C1 /]# ping -c 1 www.google.be  
PING www.google.be (216.58.214.163) 56(84) bytes of data.  
64 bytes from mad01s26-in-f3.1e100.net (216.58.214.163): icmp_seq=1 ttl=255  
time=1.58 ms  
--- www.google.be ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 1.583/1.583/1.583/0.000 ms
```

Met ip (in package iproute) checken we het IP-nummer.

```
[root@C1 /]# ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group  
default qlen 1000  
...  
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state  
UNKNOWN group default qlen 1000  
    link/ether 06:89:97:11:96:f4 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.112.100/24 brd 192.168.112.255 scope global noprefixroute  
ens160  
        valid_lft forever preferred_lft forever  
    inet6 fe80::489:97ff:fe11:96f4/64 scope link proto kernel_ll  
        valid_lft forever preferred_lft forever
```

Zoals verwacht heeft de container hetzelfde IP-adres als de host (192.168.112.100).

## ROOTLESS NETWORKING PASTA



Met curl (in package curl) vragen we een webpagina op.

```
[root@C1 /]# curl www.google.be
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

Zoals je ziet kan de container dus op internet enz. zonder specifiek instellingen hiervoor te hebben gedaan.

### 5.3.3.2 Communicatie container met container host

We gaan hiervoor eerst uit container.

```
[root@C1 /]# exit
exit
student@serverXX:~$
```

Installeer een webserver op je container host.

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

We starten httpd.

```
student@serverXX:~$ sudo systemctl start httpd
```

Uiteraard is de website nu beschikbaar vanaf de container host.

```
student@serverXX:~$ curl 127.0.0.1
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...

```

De website is uiteraard beschikbaar vanaf ServerXX.

We gaan nu terug in de container.

```
student@serverXX:~$ podman exec -it C1 bash
```

We vragen nu de webpagina op.

```
[root@C1 /]# curl 127.0.0.1
curl: (7) Failed to connect to 127.0.0.1 port 80 after 0 ms: Could not connect
to server

[root@C1 /]# curl 192.168.112.100
curl: (7) Failed to connect to 192.168.112.100 port 80 after 0 ms: Could not
connect to server

[root@C1 /]# curl localhost
curl: (7) Failed to connect to localhost port 80 after 0 ms: Could not connect
to server
```

Het is duidelijk dat een verbinding met de website op de host standaard niet werkt...

Pasta werkt standaard als volgt:

- 127.0.0.1 / localhost in de container  
De loopback van de container zelf, dus niet de host.
- 192.168.112.100  
Het externe LAN-IP dat pasta aan de container geeft, maar dat wordt ook als "eigen adres" gezien → een connect daarop gaat terug naar de container zelf (en faalt als daar geen service draait).

Je kan dit oplossen door gebruik te maken van host.containers.internal.

```
[root@C1 /]# curl http://host.containers.internal
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...

```

Via host.containers.internal kan je altijd de host bereiken vanuit de container.

Het wordt vertaald naar een IP-adres via /etc/hosts.

```
[root@C1 /]# cat /etc/hosts  
...  
169.254.1.2      host.containers.internal host.docker.internal  
...
```

Je kan dit ook de gateway van de host te gebruiken om vanuit de container verbinding te maken met de host.

```
student@serverXX:~$ podman run -d --network 'pasta:--map-gw' --name C2 --  
hostname C2 ubi10/ubi-init
```

Met de optie --map-gw zegt Podman tegen pasta:

“maak de gateway van de host (het default gateway-adres van ens160) bereikbaar in de container en gebruik dat als alias voor de host-loopback.”

We gaan in de container.

```
student@serverXX:~$ podman exec -it C2 bash
```

We maken nu verbinding met de webserver op de host door de IP-nummer van de gateway in te geven.

```
[root@C2 /]# curl 192.168.112.2  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">  
...
```

Je kan ook een specifiek IP-adres instellen om verbinding te maken met de host.

```
student@serverXX:~$ podman run -d --network 'pasta:--map-host-  
loopback=11.11.11.11' --name C3 --hostname C3 ubi10/ubi-init  
83756754ce2527c473c934897d269bba73c9d917f2cd9a0e0e13077ac40d308b
```

We gaan in de container.

```
student@serverXX:~$ podman exec -it C3 bash
```

We maken nu verbinding met de webserver op de host door de IP-nummer van de gateway in te geven.

```
student@serverXX:~$ podman exec -it C3 bash
```

```
[root@C3 /]# curl 11.11.11.11
```

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...
--map-host-loopback=11.11.11.11 = creëert in de container een extra IP (11.11.11.11) dat
wordt doorgestuurd naar de loopback van de host (127.0.0.1 op de host).

```

Je kiest bij na “--map-host-loopback=” een IP-adres met volgende eisen:

- Het mag nog niet in gebruik zijn op de host of in je LAN.
- Het moet binnen een geldig privé-/gereserveerd bereik vallen, zodat het geen echte internetroute heeft.
- Het IP-adres wordt alleen in de container zichtbaar

Ga nu uit de container.

```
[root@C3 /]# exit
exit
```

Zet de webserver uit op ServerXX.

```
student@serverXX:~$ sudo systemctl stop httpd
```

### 5.3.3.3 Communicatie met containers van buitenaf

#### 5.3.3.3.1 Vooraf

Verwijder nu alle containers.

```
student@serverXX:~$ podman rm --all -f
...
```

#### 5.3.3.3.2 Vanaf de host

We zullen in dit hoofdstuk een alternatieve methode gebruiken om een webserver op UBI10 te installeren.

```
student@serverXX:~$ podman run -d --name myhttpd -p 8080:80 ubi10/ubi sh -c
"dnf install -y httpd && httpd -D FOREGROUND"
```

```
222f4f4b35a473a9bdाaaea6f2f6706008791e306584e3957d5730b4f8957cdb
```

-d zorgt ervoor dat de container op de achtergrond draait.

--name myhttpd zorgt er uiteraard voor dat myhttpd de naam van de container is.

-p 8080:80 zorgt voor portmapping (hostpoort 8080 → containerpoort 80).

sh -c "...": Het commando sh -c wordt gestart om één of meerdere commando's te starten.

dnf install -y httpd installeert de webserver.

&&: als de installatie succesvol is wordt de webserver gestart.

httpd -D FOREGROUND voert de webserver op de voorgrond uit .

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

CONTAINER ID	IMAGE	COMMAND ...
222f4f4b35a4	registry.access.redhat.com/ubi10/ubi:latest	sh -c dnf install... ...

Waarom moet -D FOREGROUND toegevoegd worden?

- sh -c "..." wordt het PID 1 proces in de container (het eerste en hoofdproces).
  - o sh voert jouw hele commando-string uit.
- Binnen dat sh -c gebeurt:
  - o Eerst: dnf install -y httpd → dit installeert Apache.
  - o Daarna, alleen als dat lukt, start httpd -D FOREGROUND.
- Zodra httpd -D FOREGROUND draait, neemt httpd de rol over als het actieve proces.

Omdat het met -D FOREGROUND draait, blijft het proces actief op de voorgrond en blokkeert de shell. Dat zorgt ervoor dat de container niet meteen afsluit.

We kunnen nu checken of de container beschikbaar is vanaf de container host. Wacht wel even om onderstaand commando uit te voeren! De webserver moet immers geïnstalleerd en gestart worden.

```
student@serverXX:~$ curl 192.168.112.100:8080
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

```
...
```

Op soortgelijke wijze kan je uiteraard andere poorten doorsturen maar, omdat je een gewone gebruiker bent, kan je **standaard geen poorten mappen naar een poort op de host die kleiner dan 1024**.

Er zijn mogelijkheden om dit te omzeilen maar sommige methodes zetten de beveiliging op losse schroeven. Dit gaan we uiteraard niet doen!

We zullen laten zien hoe je via de firewall toch ervoor kan zorgen dat de webserver beschikbaar is via poort 80 op de host maar ook hier zijn rootrechten voor nodig.

Je moet om te beginnen port forwarding aan hebben staan.

```
student@serverXX:~$ sudo sysctl -w net.ipv4.ip_forward=1
```

We maken nu een firewall “doorstuurregel”.

```
student@serverXX:~$ sudo firewall-cmd --direct --add-rule ipv4 nat OUTPUT 0 -p tcp --dport 80 -j REDIRECT --to-ports 8080
```

success

--direct: hierdoor vermijd je dat je met zones werkt

ipv4: de regel geldt voor ipv4

nat: NAT wordt gebruikt om verkeer door te sturen

OUTPUT: dit is de keten (OUTPUT heeft betrekking op lokaal verkeer van de host zelf)

0: Prioriteit (0 is hoog)

Om het anders uit te leggen: NAT (in de OUTPUT chain) grijpt in vóór de kernel het pakket verzendt.

De kernel herschrijft de bestemming: poort 80 → poort 8080 (op dezelfde host).

```
student@serverXX:~$ curl 192.168.112.100
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...

```

#### 5.3.3.3.3 Vanaf andere host

We willen nu instellen dat je vanaf ClientXX verbinding kan maken met de webserver op ServerXX op poort 80.

Hiervoor dien je op ServerXX volgende firewallregel in te stellen.

```
student@serverXX:~$ sudo firewall-cmd --direct --add-rule ipv4 nat PREROUTING 0
-p tcp --dport 80 -j DNAT --to-destination 192.168.112.100:8080
```

PREROUTING: dit is de keten (PREROUTING heeft betrekking op verkeer dat binnenkomt)

Verkeer dat binnenkomt op poort 80 wordt herschreven naar 192.168.112.100:8080.

We kunnen nu op ClientXX verbinding maken met de webserver op ServerXX.

```
student@clientXX:~$ curl 192.168.112.100
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

```
...
```

We verwijderen nu alle toegevoegde tijdelijke firewallregels door de firewall te herladen.

```
student@serverXX:~$ sudo systemctl reload firewalld
```

Uiteraard is er nu geen verbinding meer mogelijk vanaf ClientXX en is enkel verbinding mogelijk via poort 8080 op ServerXX.

```
student@clientXX:~$ curl 192.168.112.100
```

```
curl: (7) Failed to connect to 192.168.112.100 port 80 after 0 ms: Could not connect to server
```

```
student@serverXX:~$ curl 192.168.112.100
```

```
curl: (7) Failed to connect to 192.168.112.100 port 80 after 0 ms: Could not connect to server
```

```
student@serverXX:~$ curl 192.168.112.100:8080
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

```
...
```

#### 5.3.3.4 Communicatie tussen 2 containers

We gaan dat hier bespreken via hostpoort. Er zijn nog andere manieren, maar daar komen we later op terug.

Je draait nu onderstaande container nog.

```
podman run -d --name myhttpd -p 8080:80 ubi10/ubi sh -c "dnf install -y httpd
&& httpd -D FOREGROUND"
```

Omdat we de optie -p 8080:80 hebben gebruikt, wordt poort 8080 op de host door Podman doorgestuurd naar poort 80 in de container.

Je kan gebruik maken van host.container.internal in een andere container omdat de website beschikbaar is vanaf vanaf de host.

```
student@serverXX:~$ podman run --rm -it ubi10/ubi curl -v
http://host.container.internal:8080

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

...

--rm: de container wordt definitief verwijderd na uitvoering.

### 5.3.4 User-space netwerk

#### 5.3.4.1 Voorbeeld 1

In sommige gevallen wil je de subnets definiëren waarop je containers zich bevinden of specifieke IP-adressen toewijzen aan de containers binnen een netwerk.

Je kunt een nieuw netwerk aanmaken met de gewenste subnetconfiguratie door gebruik te maken van de opdracht podman network create. Deze opdracht stelt je in staat om een aangepast user-mode netwerk te definiëren met specifieke subnet- en gatewayinstellingen.

Gebruik de volgende opdracht om een nieuw netwerk aan te maken:

```
student@serverXX:~$ podman network create --subnet 192.168.5.0/24 --gateway 192.168.5.1 mynat
```

Uitleg:

- podman network create: De opdracht om een nieuw Podman-netwerk aan te maken.
- --subnet 192.168.5.0/24: Specificeert het subnet dat het nieuwe netwerk moet gebruiken.
- --gateway 192.168.5.1: Stelt het gateway-adres in voor het netwerk.
- mynat: De naam van het nieuwe netwerk dat wordt aangemaakt.

Controleer na het aanmaken van het netwerk of het correct is ingesteld met:

```
student@serverXX:~$ podman network ls
```

The screenshot shows two terminal windows side-by-side. The left window is titled 'student@serverXX:~' and the right window is titled 'student@clientXX:~'. Both windows show the command 'podman network ls' being run. The server window shows a table of networks with one entry: NETWORK ID '2f259bab93aa' and NAME 'podman' under DRIVER 'bridge'. The client window shows a similar table with one entry: NETWORK ID '05112f49f255' and NAME 'mynat' under DRIVER 'bridge'. This indicates that the network 'mynat' was successfully created on the server.

```
student@serverXX:~$ podman network ls
NETWORK ID      NAME      DRIVER
2f259bab93aa   podman   bridge
student@serverXX:~$ podman network create --subnet 192.168.5.0/24 --gateway 192.168.5.1 mynat
mynat
student@serverXX:~$ podman network ls
NETWORK ID      NAME      DRIVER
05112f49f255   mynat    bridge
2f259bab93aa   podman   bridge
student@serverXX:~|
```

Als je gedetailleerde informatie wilt bekijken over het specifiek aangemaakte netwerk mynat, gebruik dan de volgende opdracht:

```
student@serverXX:~$ podman network inspect mynat
```

The screenshot shows two terminal windows side-by-side. The left window is titled 'student@serverXX:~' and the right window is titled 'student@clientXX:~'. Both windows show the command 'podman network inspect mynat' being run. The output is a JSON object describing the 'mynat' network. A red box highlights the 'name' field ('"name": "mynat"') and another red box highlights the subnet configuration ('"subnet": "192.168.5.0/24"', '"gateway": "192.168.5.1"').

```
student@serverXX:~$ podman network inspect mynat
[{"name": "mynat", "id": "05112f49f25554deb02778899a5abbf03eca8dfeedac3be804a7eb30c9c8bb3b0", "driver": "bridge", "network_interface": "podman1", "created": "2025-09-02T11:09:53.371806225+02:00", "subnets": [{"subnet": "192.168.5.0/24", "gateway": "192.168.5.1"}], "ipv6_enabled": false, "internal": false, "dns_enabled": true, "ipam_options": {"driver": "host-local"}, "containers": {}}]
student@serverXX:~$
```

Verwijder de bestaande containers van gebruiker student.

```
student@serverXX:~$ podman rm --all -f
```

Voer de volgende opdrachten uit om de containers C1 en C2 aan te maken, en koppel ze aan het eerder aangemaakte mynat-netwerk met specifieke IP-adressen:

- Container C1 aanmaken met IP-adres 192.168.5.150:

```
student@serverXX:~$ podman run -d --name C1 --hostname C1 --network mynat --ip 192.168.5.150 ubi10/ubi-init
```

```
4b2747c0eb2fddfc3b5263eb70668c8ca5f4f4cddb1203865b3132c770d6fef
```

- Container C2 aanmaken met IP-adres 192.168.5.151:

```
student@serverXX:~$ podman run -d --name C2 --hostname C2 --network mynat --ip 192.168.5.151 ubi10/ubi-init
```

```
90a401c67a1d63221925aa893690db03e54ecc4a1313234cdcf920f7510f4dc
```

We checken nu of de instellingen van C1 correct zijn. Gebruik de volgende opdracht om de Bash-interface te openen binnen container C1:

```
student@serverXX:~$ podman exec -it C1 bash
```

We voeren nu "ip a" uit.

```
[root@C1 /]# ip a
```

```
[root@C1 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host proto kernel_ll
        valid_lft forever preferred_lft forever
2: eth0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 82:34:5c:7b:c1:43 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.5.150/24 brd 192.168.5.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::8034:5cff:fe7b:c143/64 scope link proto kernel_ll
        valid_lft forever preferred_lft forever
[root@C1 /]#
```

Je ziet hier eth0@if4.

-eth0 is interface van de container.

-@if4 is koppeling met index 4 op host (bij rootless vind je niets hiervan terug op de host – zie verder!).

We voeren nu ping uit na installatie iputils.

```
[root@C1 /]# dnf install -y iputils
[root@C1 /]# ping -c 1 C2
```

```
[root@C1 /]# ping -c 1 C2
PING C2.dns.podman (192.168.5.151) 56(84) bytes of data.
64 bytes from c2 (192.168.5.151): icmp_seq=1 ttl=64 time=0.066 ms

--- C2.dns.podman ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.066/0.066/0.066/0.000 ms
[root@C1 /]#
```

Je kan nu dus rechtstreeks communiceren tussen C1 en C2 met de naam en niet enkel met IP-adres.

We bekijken gateway in container.

```
[root@C1 /]# ip route
default via 192.168.5.1 dev eth0 proto static metric 100
192.168.5.0/24 dev eth0 proto kernel scope link src 192.168.5.150
```

```
[root@C1 /]# ip route
default via 192.168.5.1 dev eth0 proto static metric 100
192.168.5.0/24 dev eth0 proto kernel scope link src 192.168.5.150
[root@C1 /]#
```

We bekijken nu de DNS-servers die gebruikt worden in de container.

```
[root@C1 /]# cat /etc/resolv.conf
search dns.podman
```

```
nameserver 192.168.5.1
```

Je kan op internet in de container.

```
[root@C1 /]# ping -c 1 www.google.be
PING www.google.be (216.58.214.163) 56(84) bytes of data.
64 bytes from mad01s26-in-f3.1e100.net (216.58.214.163): icmp_seq=1 ttl=254
time=25.5 ms

--- www.google.be ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.464/25.464/25.464/0.000 ms
```

Ga nu uit de container.

```
[root@C1 /]# exit
exit
```

Bij een rootless netwerk is er geen netwerkkaart op de host in het subnet van de container!

Als je docker hebt geïnstalleerd heb je wel docker0 maar dat heeft er niets mee te maken natuurlijk.

```
student@serverXX:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:0c:29:99:b5:3c brd ff:ff:ff:ff:ff:ff
    altname enp3s0
    altname enx000c2999b53c
    inet 192.168.112.100/24 brd 192.168.112.255 scope global noprefixroute ens160
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe99:b53c/64 scope link noprefixroute
```

```
valid_lft forever preferred_lft forever
```

### 5.3.4.2 Voorbeeld 2

Je kunt een Podman-netwerk aanmaken met extra opties, waarmee je aanvullende configuraties kunt opgeven.

De onderstaande opdracht maakt een nieuw NAT-netwerk aan met aangepaste DNS en gateway:

```
student@serverXX:~$ podman network create mynat2 --subnet 10.0.0.0/24 --gateway  
10.0.0.1 --dns 10.0.0.4  
  
mynat2
```

Extra netwerkopties:

- dns 10.0.0.4: Wijs een specifieke DNS-server toe (in dit voorbeeld 10.0.0.4).
- gateway 10.0.0.1: Dit stelt de DNS-server in.

We listen nu terug de docker networks.

```
student@serverXX:~$ podman network ls  
  
NETWORK ID      NAME        DRIVER  
05112f49f255   mynat       bridge  
2ad4d1aed622   mynat2      bridge  
2f259bab93aa   podman      bridge
```

Je kan, zoals je ziet, meerdere NAT-netwerken hebben ingesteld.

We gaan specifieke informatie opvragen over mynat2.

```
student@serverXX:~$ podman network inspect mynat2
```

The screenshot shows two terminal windows side-by-side. The left window is titled 'student@serverXX:~' and the right window is titled 'student@clientXX:~'. Both windows are running the command 'podman network inspect mynat2'. The output is a JSON object with several fields highlighted by red boxes:

- 'name': 'mynat2'
- 'id': '2ad4dlaed622828a1bc4cff7bebc31e500e0afc582a39484d1c0d6f6da4a7b41'
- 'driver': 'bridge'
- 'network\_interface': 'podman2'
- 'created': '2025-09-02T11:43:03.279982941+02:00'
- 'subnets': [ { 'subnet': '10.0.0.0/24', 'gateway': '10.0.0.1' } ]
- 'ipv6\_enabled': false
- 'internal': false
- 'dns\_enabled': true
- 'network\_dns\_servers': [ '10.0.0.4' ]
- 'ipam\_options': { 'driver': 'host-local' }
- 'containers': {}

At the bottom of the left window, there is a prompt: 'student@serverXX:~\$ |'

### 5.3.4.3 Verwijderen eigen netwerk

We verwijderen het user-mode netwerk mynat2.

```
student@serverXX:~$ podman network rm mynat2
mynat2
```

### 5.3.4.4 User-mode netwerken en poorttoewijzing

Een user-mode netwerk biedt bescherming tegen de buitenwereld door netwerkisolatie. Containers die verbonden zijn met een user-mode netwerk kunnen wel naar buiten communiceren, maar inkomend verkeer van buitenaf wordt standaard geblokkeerd. Om externe toegang te bieden tot bepaalde services die op containers draaien, moet je poorten publiceren zoals we reeds geleerd hebben.

Als je bijvoorbeeld een webservice draait in een container en je wilt dat externe clients deze kunnen benaderen via TCP-poort 80 (standaard HTTP-poort), moet je deze poort expliciet openzetten.

Om dit te demonstreren vertrekken we vanuit het netwerk mynat.

```
student@serverXX:~$ podman network ls
```

NETWORK ID	NAME	DRIVER
05112f49f255	mynat	bridge
2f259bab93aa	podman	bridge

We maken een nieuwe container genaamd myhttpd aan en deze verbinden met het mynat-netwerk, waarbij je poort 8080 van de host koppelt aan poort 80 van de container. Hierdoor wordt verkeer dat de host op poort 8080 bereikt, automatisch doorgestuurd naar de container.

```
student@serverXX:~$ podman run -d -it --name myhttpd -p 8080:80 --network mynat  
--ip 192.168.5.160 ubi10/ubi
```

```
c9c1bd721f28f9a8e96d057b836aae8226eadca70753bcfee6c267b576fdfaa5
```

-p 8080:80: Koppelt poort 8080 op de host aan poort 80 binnen de container. Verkeer dat de host bereikt op poort 8080, wordt doorgestuurd naar poort 80 van de container.

--ip 192.168.5.160: de container krijgt een specifiek IP-nummer

We installeren nu een webserver in de container.

```
student@serverXX:~$ podman exec -it myhttpd dnf install -y httpd
```

We checken de IP-nummer in de container.

```
student@serverXX:~$ podman exec -it myhttpd ip a
```

```
student@serverXX:~$ podman exec -it myhttpd ip a  
Complete!  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/Loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host proto kernel_lo  
        valid_lft forever preferred_lft forever  
2: eth0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000  
    link/ether 92:41:06:52:42:ba brd ff:ff:ff:ff:ff:ff link-netnsid 0  
    inet 192.168.5.160/24 brd 192.168.5.255 scope global eth0  
        valid_lft forever preferred_lft forever  
    inet6 fe80::9041:6ff:fe52:42ba/64 scope link proto kernel_ll  
        valid_lft forever preferred_lft forever  
student@serverXX:~ |
```

We starten nu de Apache webserver in de container.

```
student@serverXX:~$ podman exec -it myhttpd httpd
```

```
AH00558: httpd: Could not reliably determine the server's fully qualified  
domain name, using 192.168.5.160. Set the 'ServerName' directive globally to  
suppress this message
```

De container fungeert nu als een webserver. Poort 80 binnen de container (en dus op de host via de gepubliceerde poort 8080) is nu toegankelijk voor HTTP-verkeer.

Je hebt nu toegang tot de webserver op de containerhost.

```
student@serverXX:~$ curl 192.168.112.100:8080  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

...

Als je meerdere containers wilt laten draaien die elk gebruik maken van poort 80 binnen de container, kun je verschillende poorten op de host koppelen aan poort 80 in elke container. Hierdoor kun je meerdere containers toegankelijk maken via verschillende poorten op de host, terwijl ze binnen de container nog steeds poort 80 gebruiken.

```
student@serverXX:~$ podman run -d --name myhttpd2 --publish 1234:80 ubi10/ubi
sh -c "dnf install -y httpd && httpd -D FOREGROUND"

--publish 1234:80 of -p 1234:80 is hetzelfde
```

Hier installeer ik onmiddellijk de webserver in de container.

Dit betekent dat je de webserver van myhttpd2 in de container kunt benaderen via <http://<host-ip>:1234>.

We checken dit (wat wel even tot de webserver geïnstalleerd is en draait).

```
student@serverXX:~$ curl 192.168.112.100:1234

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

"
```

## 5.4 Rootful netwerktoegang

We zullen bestuderen of er een verschil is tussen rootfull en rootless netwerk als je zelf geen netwerk specificeert.

We starten één container op als root.

```
student@serverXX:~$ sudo podman run -d --name C10 --hostname C10 ubi10/ubi-init
53eb6d6053e8735807fcf8a3a5137499b97dd6f22cfa560324db0bfd877004f1
```

We gaan in de container:

```
student@serverXX:~$ sudo podman exec -it C10 bash
[root@C10 /]#
```

In de container installeren we essentiële netwerktools.

```
[root@C10 /]# dnf install -y iputils iproute curl
"
```

We zullen nu testen hetgeen mogelijk is in de container.

We checken of je ICMP echo-packages met ping (in package iputils) kan sturen naar en ontvangen van www.google.be.

```
[root@C10 /]# ping -c 1 www.google.be
PING www.google.be (216.58.214.163) 56(84) bytes of data.
64 bytes from mad01s26-in-f3.1e100.net (216.58.214.163): icmp_seq=1 ttl=255
time=1.58 ms

--- www.google.be ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.583/1.583/1.583/0.000 ms
```

Met curl (in package curl) vragen we een webpagina op.

```
[root@C1 /]# curl www.google.be
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A href="http://www.google.com/">here</A>
</BODY></HTML>
```

Zoals je ziet kan de container dus op internet zonder specifiek instellingen hiervoor te hebben gedaan.

Met ip (in package iproute) checken we het IP-nummer.

```
[root@C10 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qdisc mq 0:0
    link/loopback brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 0.0.0.0 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 brd 0.0.0.0 scope host proto kernel
        valid_lft forever preferred_lft forever
2: eth0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default qlen 1000
    link/ether f6:84:42:67:fa:a7 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

```
inet 10.88.0.2/16 brd 10.88.255.255 scope global eth0
      valid_lft forever preferred_lft forever

inet6 fe80::f484:42ff:fe67:faa7/64 scope link proto kernel_ll
      valid_lft forever preferred_lft forever

[root@C10 /]# exit

exit
```

In tegenstelling tot bij rootless netwerktoegang krijg je container wel een eigen IP-nummer. In dit geval is dat 10.88.0.2.

**Dit komt omdat bij een rootful container Podman een echte Linux bridge kan aanmaken in tegenstelling tot bij rootless.**

Er wordt gebruik gemaakt van het standaard podman netwerk aangezien we geen netwerk gespecificeerd hebben.

```
student@serverXX:~$ sudo podman network ls

NETWORK ID      NAME        DRIVER
2f259bab93aa  podman      bridge
```

We inspecteren nu het podman-netwerk.

```
student@serverXX:~$ sudo podman network inspect podman
```

```

student@serverXX:~$ sudo podman network inspect podman
[
  {
    "name": "podman",
    "id": "2f259bab93aaaaa2542ba43ef33eb990d0999ee1b9924b557b7be53c0b7a1bb9",
    "driver": "bridge",
    "network_interface": "podman0",
    "created": "2025-09-10T11:47:26.018785055+02:00",
    "subnets": [
      {
        "subnet": "10.88.0.0/16",
        "gateway": "10.88.0.1"
      }
    ],
    "ipv6_enabled": false,
    "internal": false,
    "dns_enabled": false,
    "ipam_options": {
      "driver": "host-local"
    },
    "containers": [
      "53eb6d6053e8735807fcf8a3a5137499b97dd6f22cfa560324db0bfd877004f1": {
        "name": "C10",
        "interfaces": {
          "eth0": {
            "subnets": [
              {
                "ipnet": "10.88.0.2/16",
                "gateway": "10.88.0.1"
              }
            ],
            "mac_address": "f6:84:42:67:fa:a7"
          }
        }
      }
    ]
  }
]

```

Je ziet hier ook de netwerkinstellingen van de draaiende container C10.

We bekijken nu de IP-instellingen op de container host ServerXX.

```
student@serverXX:~$ ip a
```

```

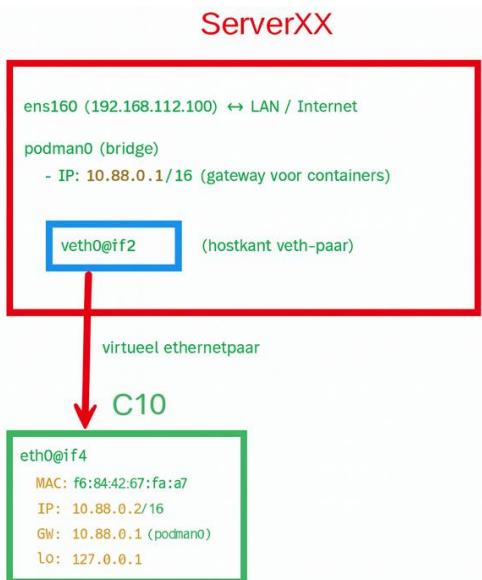
student@serverXX:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
      valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
  link/ether 00:0c:29:99:b5:3c brd ff:ff:ff:ff:ff:ff
  altname enp3s0
  altname enx000c2999b53c
  inet 192.168.112.100/24 brd 192.168.112.255 scope global noprefixroute ens160
    valid_lft forever preferred_lft forever
  inet6 fe80::20c:29ff:fe99:b53c/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
3: podman0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
  link/ether 7a:cd:fe:74:b2:0e brd ff:ff:ff:ff:ff:ff
  inet 10.88.0.1/16 brd 10.88.255.255 scope global podman0
    valid_lft forever preferred_lft forever
  inet6 fe80::78cd:feff:fe74:b20e/64 scope link proto kernel_ll
    valid_lft forever preferred_lft forever
4: veth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master podman0 state UP group default qlen 1000
  link/ether b2:1c:aa:f6:23:8b brd ff:ff:ff:ff:ff:ff link-netns netns-96b021ba-201f-430b-3798-b3f13b9c65f2
  inet6 fe80::b01c:aaff:fef6:238b/64 scope link proto kernel_ll
    valid_lft forever preferred_lft forever

```

Podman0 is de virtuele bridge. De gateway is 10.88.0.1.

veth0@if2 is de hostkant van de virtuele ethernet-kabel tussen bridge en container C10. Het is een switch-poort op level 2. De andere kant (eth0@if7) zit in de container.

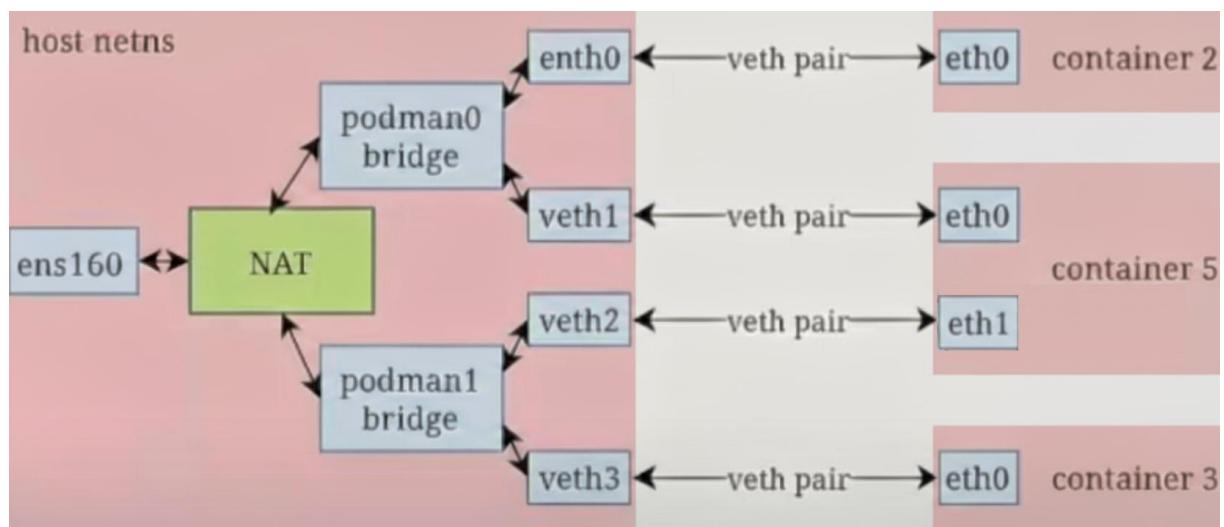
Hieronder zie je een schema van de situatie.



Hoe het werkt...

- Container C10 stuurt verkeer → eth0@if4 → veth-paar → veth0@if2 (host) → bridge podman0.
- Als het verkeer naar internet moet, gaat het via gateway 10.88.0.1 (bridge) → ens160 → LAN.
- Andere containers (indien er die zijn) in hetzelfde netwerk krijgen ook een veth naar podman0 en praten rechtstreeks via de bridge.

Hieronder vind je een algemener schema hierover.



Vanuit de container host kan je nu dus rechtstreeks ICMP-pakketten sturen naar de container.

```

student@serverXX:~$ ping -c 1 10.88.0.2
PING 10.88.0.2 (10.88.0.2) 56(84) bytes of data.
64 bytes from 10.88.0.2: icmp_seq=1 ttl=64 time=0.087 ms
--- 10.88.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.087/0.087/0.087/0.000 ms

```

Zonder port mapping is een container echter alleen vanaf de host bereikbaar, niet vanaf buiten! Logisch aangezien het bridge-netwerk niet voor de buitenwereld beschikbaar is.

Je kan, omdat je root bent, nu ook poorten publiceren kleiner dan 1024.

We zullen de standaardpoort 80 gebruiken. Stop dus zeker de webserver die draait op ServerXX zelf.

```
student@serverXX:~$ sudo systemctl stop httpd
```

Hier een voorbeeld.

```

student@serverXX:~$ sudo podman run -d --name myhttpd10 --publish 80:80
ubi10/ubi sh -c "dnf install -y httpd && httpd -D FOREGROUND"
19038ae548760cf6c978eb422641d2514974a40ef849020a221d7f9372b4aacc

```

Je kan de website nu bereiken niet enkel bereiken via het IP-nummer van de container maar ook via 192.168.112.100 (wacht weer even voordat je het uitvoert!):

```

student@serverXX:~$ curl 192.168.112.100
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...
```

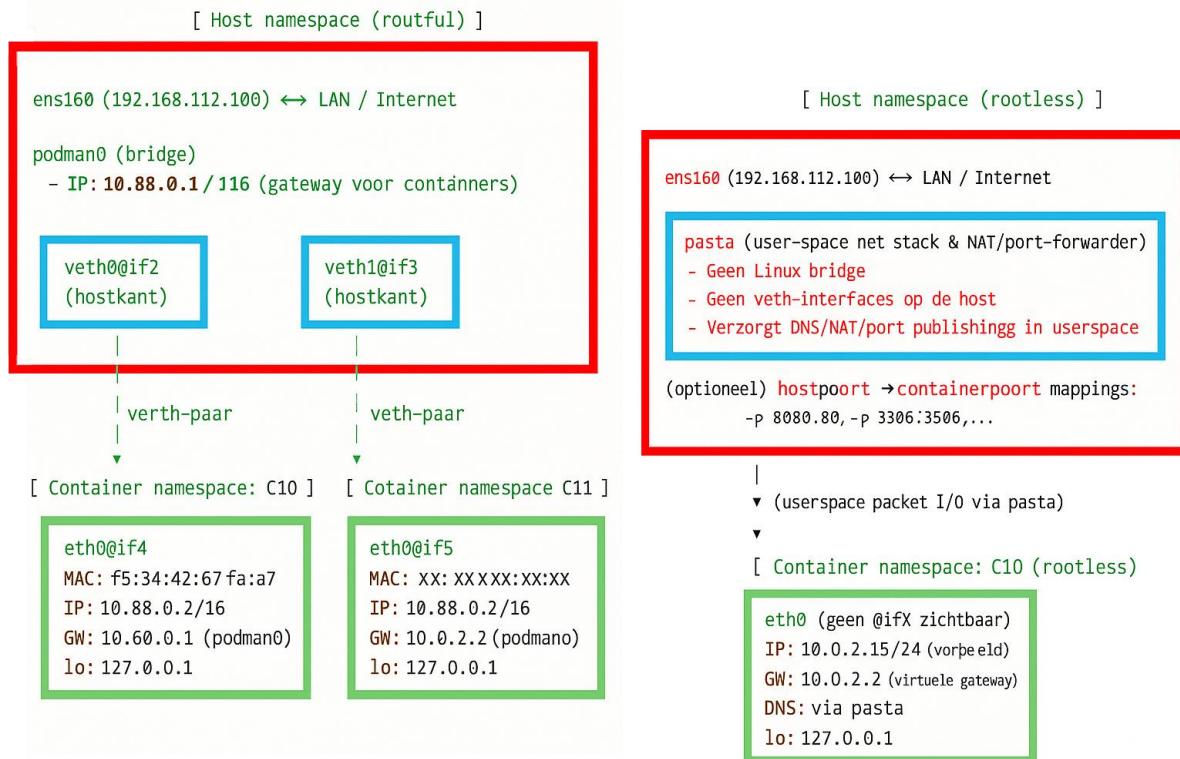
Je kan nu deze website beschikbaar maken via het netwerk.

## 5.5 User-space netwerken vs bridge netwerken

Hieronder vind je een vergelijkingstabel tussen rootful bridge en rootless user-space netwerken.

Eigenschap	Rootful (sudo podman)	Rootless (podman)
Netwerkdriver	bridge (podman0)	slirp4netns / pasta
IP-adressen	Echte kernel-bridge IP's	Virtueel (userspace)

Eigenschap	Rootful (sudo podman)	Rootless (podman)
Poorten <1024	Mogelijk	Niet mogelijk
Veiligheid	Minder geïsoleerd (root)	Veiliger (user)



## 5.6 Netwerkmodi

Er zijn verschillende netwerkmodi beschikbaar zijn voor Podman.

We hebben volgende al besproken:

- Bridge  
Standaard voor rootful containers.
- Pasta  
Standaard voor rootless containers.

Daarnaast heb je nog volgende modi:

- Host (rootful en rootless)  
Apps die alle netwerken van de host moeten zien, debugging.
- None (rootful en rootless)  
Maximale isolatie, geen netwerk nodig.
- Macvlan (enkel rootful)  
Containers zichtbaar als aparte hosts op fysiek LAN.

- Ipvlan (enkel rootful)
 

Zelfde als macvlan, maar zonder extra MAC's in LAN.

De modus Bridge is voldoende voor de meeste situaties en hebben we reeds besproken maar we bespreken ook de andere modi.

## 5.7 Host-netwerk

Verwijder eerst alle containers van student en root.

```
student@serverXX:~$ podman rm --all -f; sudo podman rm --all -f
...
```

Verwijder ook alle niet-standaard netwerken van student en root.

```
student@serverXX:~$ podman network prune; sudo podman network prune
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N] y
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N] y
```

Start een webserver met het host-netwerk en test de toegang.

```
student@serverXX:~$ sudo podman run -d --name myhttpd --network host ubi10/ubi
sh -c "dnf install -y httpd && httpd -D FOREGROUND"
b22d49742ecddd2bcb59390f5b5c7d82eea76a2723ea77f1305b32bd7866128e
```

Hier is een container gestart met het host-netwerk. Httpd wordt ook gestart in de container.

We checken de IP van de container.

```
student@serverXX:~$ sudo podman exec -it myhttpd ip addr show
```

```
student@serverXX:~$ sudo podman exec -it myhttpd ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
  link/ether 00:0c:29:99:b5:3c brd ff:ff:ff:ff:ff:ff
    altname enp3s0
    altname enx000c2999b53c
    inet 192.168.112.100/24 brd 192.168.112.255 scope global noprefixroute ens160
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe99:b53c/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
student@serverXX:~$ |
```

Toegang testen vanuit ServerXX.

```
student@serverXX:~$ curl localhost

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

...
```

Probeer vanuit een andere container toegang te krijgen tot de host-webserver.

```
student@serverXX:~$ podman run --rm --network host ubi10/ubi curl
http://localhost

...
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>

    <title>Test Page for the HTTP Server on Red Hat Enterprise
Linux</title>

...
<body>
```

--network host: Met deze optie wordt deze container uitgevoerd in het netwerk van de host.

Als je een container hebt die het host-netwerk gebruikt, geldt het volgende:

- Zelfde IP-adres als de host  
De container gebruikt hetzelfde IP-adres als de host-machine, omdat hij geen virtueel netwerk krijgt toegewezen.
- Poortgebruik  
Omdat de container dezelfde netwerkstack deelt, moeten de poorten uniek zijn. Als je bijvoorbeeld Nginx op poort 80 draait in de container met een host-netwerk, moet poort 80 beschikbaar zijn op de host.

Rootless host-netwerk (--network host) werkt technisch gezien gewoon, maar het wordt weinig gebruikt: poorten <1024 blijven niet beschikbaar omdat dat een kernelbeperking is.

Stel dat je onderstaande uitvoert:

```
student@serverXX:~$ podman run -d --name myhttpd --network host ubi10/ubi sh -c
"dnf install -y httpd && httpd -D FOREGROUND"
```

Je zal merken dat de container na enkele seconden stopt omdat hij poort 80 probeert te binden op de host en dat mag niet. Daardoor stop httpd en eindigt de container. We bespreken dat verder niet!

## 5.8 Macvlan-netwerk

### 5.8.1 Inleiding

Een macvlan-netwerk geeft containers een eigen MAC-adres en IP-adres rechtstreeks in het fysieke LAN van de host. Macvlan werkt op layer 2. De container lijkt dus op een echte machine in het netwerk.

### 5.8.2 Macvlan netwerk aanmaken

We verwijderen eerst alle containers en netwerken.

```
student@serverXX:~$ podman rm --all -f; sudo podman rm --all -f
...
student@serverXX:~$ podman network prune; sudo podman network prune
WARNING! This will remove all networks not used by at least one container.

Are you sure you want to continue? [y/N] y
WARNING! This will remove all networks not used by at least one container.

Are you sure you want to continue? [y/N] y
```

We vragen info op over de netwerkkaarten op ServerXX.

```
student@serverXX:~$ nmcli connection show

NAME      UUID              TYPE      DEVICE
ens160    2318b781-c803-36d2-8642-bb3d5bb513f9  ethernet  ens160
lo        671960ae-dbe0-4275-a4c1-86fa77dc9a33  loopback  lo
```

We vragen IP, DNS-server en gateway op van apparaat ens160.

```
student@serverXX:~$ nmcli connection show ens160 | grep ipv4.addresses
ipv4.addresses:                         192.168.112.100/24
student@serverXX:~$ nmcli connection show ens160 | grep ipv4.dns
ipv4.dns:                                192.168.112.2
...
student@serverXX:~$ nmcli connection show ens160 | grep ipv4.gateway
ipv4.gateway:                            192.168.112.2
```

We maken nu een macvlan netwerk aan:

```
student@serverXX:~$ sudo podman network create -d macvlan --subnet
192.168.112.0/24 --gateway 192.168.112.2 -o parent=ens160 macvlan_netwerk
```

### macvlan\_nettwerk

Nadat we terug verbinding hebben gemaakt zien we dat er een netwerk is bijgekomen.

```
student@serverXX:~$ sudo podman network ls
```

NETWORK ID	NAME	DRIVER
f4edbfe1f57f	macvlan_nettwerk	macvlan
2f259bab93aa	podman	bridge

### 5.8.3 Container met macvlan netwerk

We gaan nu een container opstarten die gebruikt maakt van het macvlan netwerk.

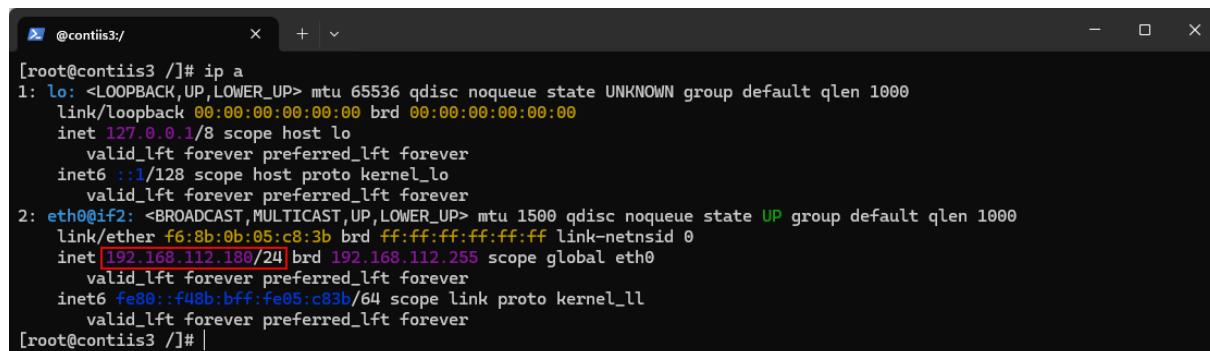
```
student@serverXX:~$ sudo podman run --detach -it --name contiis3 --hostname contiis3 --network macvlan_nettwerk --ip 192.168.112.180 ubi10/ubi  
5436fa022928ba4e4bafbe9d4c5411ce0e9d86bc19d7bfd7416501a294dbccdd
```

We gaan nu naar bash in de container.

```
student@serverXX:~$ sudo podman exec -it contiis3 bash
```

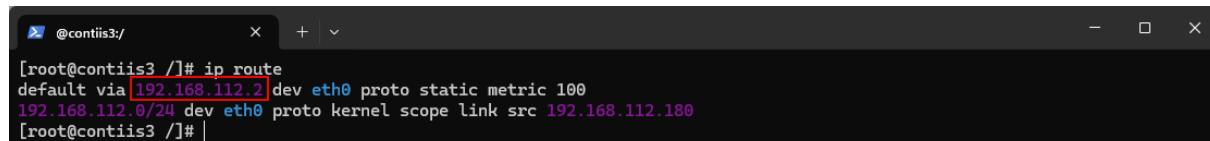
We checken nu IP, DNS en gateway.

```
[root@contiis3 /]# ip a
```



```
[root@contiis3 /]# ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/Loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host proto kernel_lo  
        valid_lft forever preferred_lft forever  
2: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000  
    link/ether f6:8b:0b:05:c8:3b brd ff:ff:ff:ff:ff:ff link-netnsid 0  
    inet 192.168.112.180/24 brd 192.168.112.255 scope global eth0  
        valid_lft forever preferred_lft forever  
    inet6 fe80::f48b:bff:fe05:c83b/64 scope link proto kernel_ll  
        valid_lft forever preferred_lft forever  
[root@contiis3 /]#
```

```
[root@contiis3 /]# ip route
```



```
[root@contiis3 /]# ip route  
default via 192.168.112.2 dev eth0 proto static metric 100  
192.168.112.0/24 dev eth0 proto kernel scope link src 192.168.112.180  
[root@contiis3 /]#
```

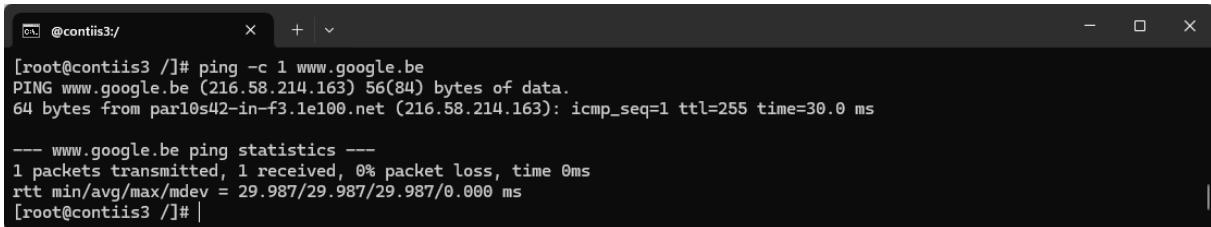
```
[root@contiis3 /]# cat /etc/resolv.conf
```



```
[root@contiiis3 ~]# cat /etc/resolv.conf
nameserver 169.254.1.1
nameserver 192.168.112.2
[root@contiiis3 ~]#
```

We voeren nu ping uit na installatie iputils om te checken of we verbinding hebben met internet.

```
[root@contiiis3 ~]# dnf install -y iputils
[root@contiiis3 ~]# ping -c 1 www.google.be
```



```
[root@contiiis3 ~]# ping -c 1 www.google.be
PING www.google.be (216.58.214.163) 56(84) bytes of data.
64 bytes from par10s42-in-f3.1e100.net (216.58.214.163): icmp_seq=1 ttl=255 time=30.0 ms
--- www.google.be ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 29.987/29.987/29.987/0.000 ms
[root@contiiis3 ~]#
```

We installeren in de container een webserver.

```
[root@contiiis3 ~]# dnf install -y httpd
```

We starten httpd op, checken of die draait en verlaten de container.

```
[root@contiiis3 ~]# httpd
AH00558: httpd: Could not reliably determine the server's fully qualified
domain name, using 192.168.112.180. Set the 'ServerName' directive globally to
suppress this message

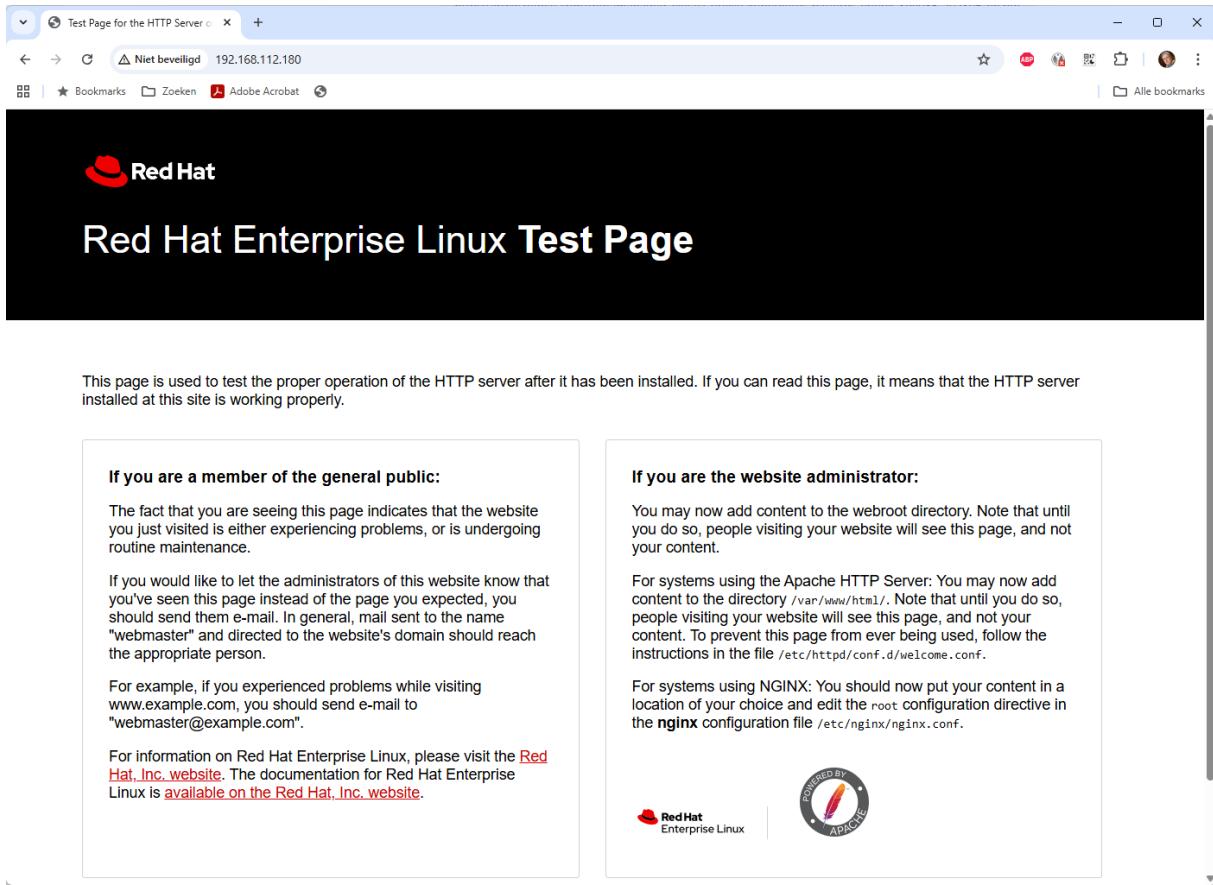
[root@contiiis3 ~]# curl 192.168.112.180

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...
[root@contiiis3 ~]# exit

exit
```

Ga naar je fysieke host (je Windows 10 of 11 computer dus) en je zal zien dat je de webpagina kan openen.



Ja kan ook vanaf een andere container verbinding maken met de container in het macvlan.

We starten hiervoor eerst een tweede container op.

```
student@serverXX:~$ sudo podman run --detach -it --name containerX --hostname
containerX --network macvlan_netwerk --ip 192.168.112.181 ubi10/ubi
```

```
2d3ca600b60819a3addf22d67d3fe350fd8b160c1c226899bc8a774fa695a72d
```

We gaan nu naar de tweede container.

```
student@serverXX:~$ sudo podman exec -it containerX bash
```

Installeer nu curl en vraag de webpagina op.

```
[root@containerX /]# dnf install -y curl
[root@containerX /]# curl http://192.168.112.180
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...

```

Verlaat nu de container.

```
[root@containerY /]# exit  
exit
```

#### 5.8.4 Scheiding met host

Bij macvlan (en ook ipvlan) krijgt de container een eigen IP op het fysieke netwerk, zonder dat er een virtuele bridge op de host nodig is. Het verkeer tussen host en container gaat echter niet automatisch via de host's eigen interface, omdat macvlan het verkeer rechtstreeks naar het fysieke netwerk stuurt. Daardoor "ziet" de host zijn eigen containers niet, tenzij je een extra interface toevoegt die in dat subnet kan praten.

Je kan dan ook niet pingen naar de container vanaf serverXX. Ook kan je de webpagina niet opvragen.

```
student@serverXX:~$ ping -c 1 192.168.112.180  
PING 192.168.112.180 (192.168.112.180) 56(84) bytes of data.  
From 192.168.112.100 icmp_seq=1 Destination Host Unreachable  
--- 192.168.112.180 ping statistics ---  
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms  
student@serverXX:~$ curl 192.168.112.180  
curl: (7) Failed to connect to 192.168.112.180 port 80 after 3107 ms: Could not  
connect to server
```

Je kan ervoor zorgen (ook al is dat een beveiligingsrisico) dat de containerhost (contiiis3) wel bereikbaar is vanaf de container met macvlan door een macvlan "Gateway" in te stellen.

Maak een nieuwe macvlan-subinterface op de host:

```
student@serverXX:~$ sudo ip link add macvlan_host link ens160 type macvlan mode  
bridge
```

Geef de nieuwe interface een IP-adres in hetzelfde subnet als het macvlan-netwerk:

```
student@serverXX:~$ sudo ip addr add 192.168.112.222/24 dev macvlan_host
```

Activeer nu de nieuwe interface.

```
student@serverXX:~$ sudo ip link set macvlan_host up
```

Je kan nu de webpagina opvragen.

```
student@serverXX:~$ curl http://192.168.112.180  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

```
...
```

Je kan ook pingen naar de container.

```
student@serverXX:~$ ping -c 1 192.168.112.180  
PING 192.168.112.180 (192.168.112.180) 56(84) bytes of data.  
64 bytes from 192.168.112.180: icmp_seq=1 ttl=64 time=0.048 ms  
--- 192.168.112.180 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 0.048/0.048/0.048/0.000 ms
```

We verwijderen nu de virtuele interface op de host omdat we die nu niet verder gaan gebruiken.

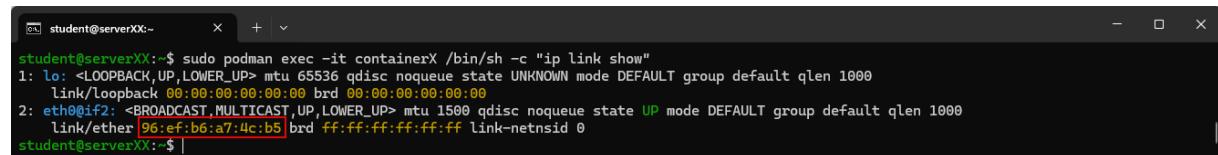
```
student@serverXX:~$ sudo ip link del macvlan_host
```

### 5.8.5 Controle macvlan

In macvlan krijgt elke container een eigen MAC-adres.

We checken het MAC-adres van contiis3.

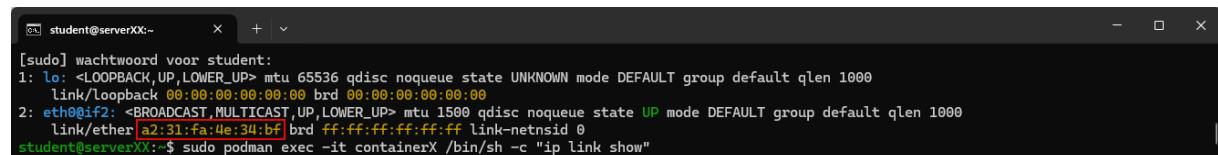
```
student@serverXX:~$ sudo podman exec -it contiis3 /bin/sh -c "ip link show"
```



```
student@serverXX:~$ sudo podman exec -it contiis3 /bin/sh -c "ip link show"  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000  
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
2: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default qlen 1000  
link/ether 96:ef:b6:a7:4c:b5 brd ff:ff:ff:ff:ff:ff link-netnsid 0  
student@serverXX:~$ |
```

We checken nu het MAC-adres van containerX.

```
student@serverXX:~$ sudo podman exec -it containerX /bin/sh -c "ip link show"
```



```
[sudo] wachtwoord voor student:  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000  
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
2: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default qlen 1000  
link/ether a2:31:fa:4e:b1 brd ff:ff:ff:ff:ff:ff link-netnsid 0  
student@serverXX:~$ sudo podman exec -it containerX /bin/sh -c "ip link show"
```

## 5.9 Ipvlan

### 5.9.1 Inleiding

Een ipvlan-netwerk geeft containers een eigen IP-adres, maar deelt hetzelfde MAC-adres als de host. Dit is een belangrijk verschil met een macvlan-netwerk, waarbij elke container zijn eigen MAC-adres krijgt.

Een ipvlan-netwerk functioneert op layer 3 (netwerklaag) en biedt containers een IP-adres direct op het fysieke netwerk van de host. Dit type netwerk deelt het MAC-adres van de host met de containers, wat het beheer vereenvoudigt en ARP-problemen op het netwerk voorkomt die soms optreden bij macvlan-netwerken.

### 5.9.2 Ipvlan-netwerk aanmaken

We verwijderen eerst alle containers en netwerken.

```
student@serverXX:~$ podman rm --all -f; sudo podman rm --all -f  
...  
student@serverXX:~$ podman network prune; sudo podman network prune  
WARNING! This will remove all networks not used by at least one container.  
Are you sure you want to continue? [y/N] y  
WARNING! This will remove all networks not used by at least one container.  
Are you sure you want to continue? [y/N] y
```

De handmatig aangemaakte macvlan-interface op ServerXX dien je ook nog te verwijderen.

```
student@serverXX:~$ sudo ip link delete macvlan_host
```

We maken nu een ipvlan netwerk aan:

```
student@serverXX:~$ sudo podman network create -d ipvlan --subnet  
192.168.112.0/24 --gateway 192.168.112.2 -o parent=ens160 ipvlan_netwerk  
ipvlan_netwerk
```

Opgelet: sudo is nodig! Anders kan je vanaf je host (Windows 10 of 11) geen verbinding maken met deze container (zie verder).

Nadat we terug verbinding hebben gemaakt zien we dat er een netwerk is bijgekomen.

```
student@serverXX:~$ sudo podman network ls  
NETWORK ID      NAME          DRIVER  
b78ca6671dbe  ipvlan_netwerk  ipvlan
```

```
2f259bab93aa podman bridge
```

### 5.9.3 Container met ipvlan netwerk

We gaan nu een container uitvoeren die gebruik maakt van het ipvlan netwerk.

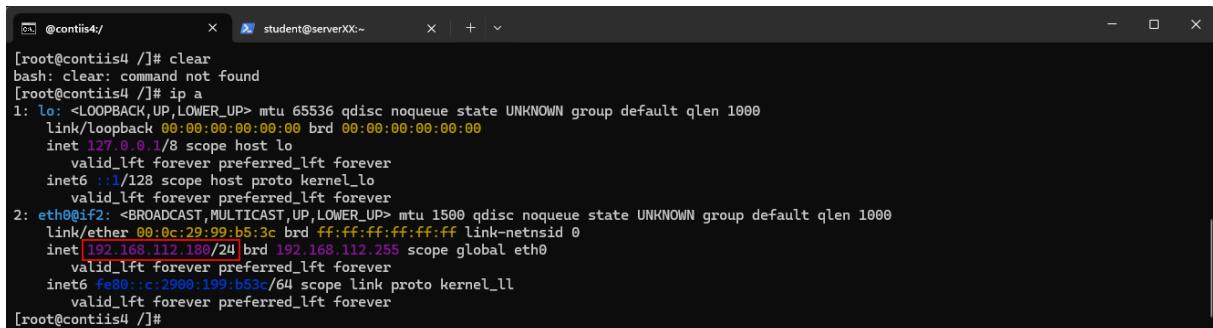
```
student@serverXX:~$ sudo podman run --detach -it --name contiis4 --hostname  
contiis4 --network ipvlan_netwerk --ip 192.168.112.180 ubi10/ubi  
  
5436fa022928ba4e4bafbe9d4c5411ce0e9d86bc19d7bfd7416501a294dbccdd
```

We gaan nu naar bash in de container.

```
student@serverXX:~$ sudo podman exec -it contiis4 bash
```

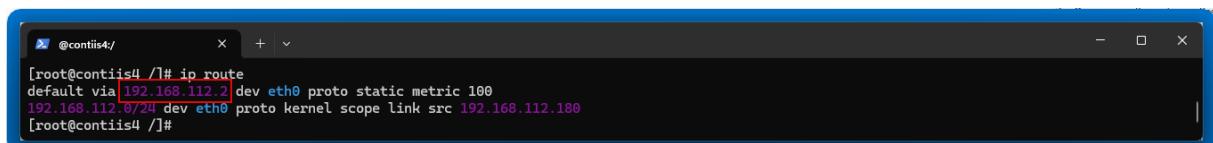
We checken nu IP, DNS en gateway.

```
[root@contiis4 /]# ip a
```



```
[root@contiis4 /]# clear  
bash: clear: command not found  
[root@contiis4 /]# ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo  
        valid_lft forever preferred_lft forever  
inet6 ::1/128 brd 00:00:00:00:00:00 scope host kernel_localhost  
    valid_lft forever preferred_lft forever  
2: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc qdisc state UNKNOWN group default qlen 1000  
    link/ether 00:0c:29:99:b5:3c brd ff:ff:ff:ff:ff:ff link-netnsid 0  
    inet 192.168.112.180/24 brd 192.168.112.255 scope global eth0  
        valid_lft forever preferred_lft forever  
    inet6 fe80::c290:199:b53c/64 scope link proto kernel ll  
        valid_lft forever preferred_lft forever  
[root@contiis4 /]#
```

```
[root@contiis4 /]# ip route
```



```
[root@contiis4 /]# ip route  
default via 192.168.112.2 dev eth0 proto static metric 100  
192.168.112.0/24 dev eth0 proto kernel scope link src 192.168.112.180  
[root@contiis4 /]#
```

```
[root@contiis4 /]# cat /etc/resolv.conf
```

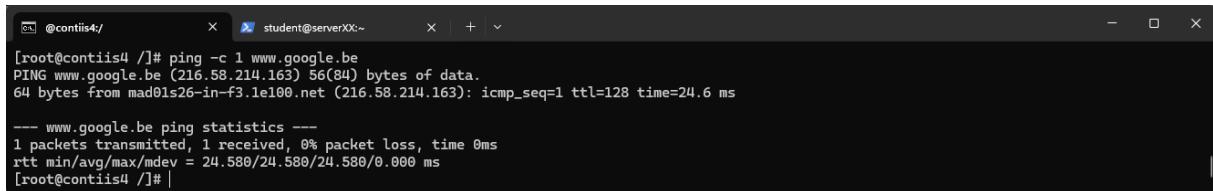


```
[root@contiis4 /]# cat /etc/resolv.conf  
nameserver 192.168.112.2  
[root@contiis4 /]#
```

We voeren nu ping uit na installatie iputils om te checken of we verbinding hebben met internet.

```
[root@contiis4 /]# dnf install -y iputils
```

```
[root@contiis4 /]# ping -c 1 www.google.be
```



```
[root@contiis4 /]# ping -c 1 www.google.be
PING www.google.be (216.58.214.163) 56(84) bytes of data.
64 bytes from mad01s26-in-f3.1e100.net (216.58.214.163): icmp_seq=1 ttl=128 time=24.6 ms

--- www.google.be ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 24.580/24.580/24.580/0.000 ms
[root@contiis4 /]# |
```

We installeren in de container een webserver.

```
[root@contiis4 /]# dnf install -y httpd
```

We starten httpd op, checken of die draait en verlaten de container.

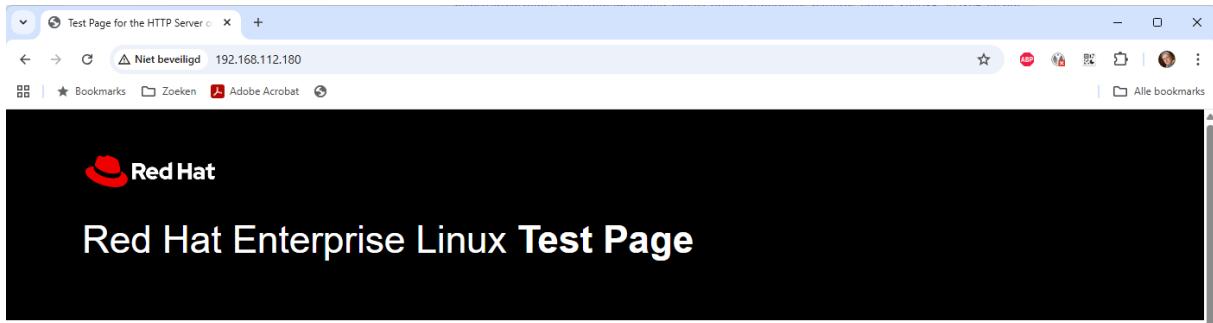
```
[root@contiis4 /]# httpd
AH00558: httpd: Could not reliably determine the server's fully qualified
domain name, using 192.168.112.180. Set the 'ServerName' directive globally to
suppress this message

[root@contiis4 /]# curl 192.168.112.180
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...
[root@contiis4 /]# exit

exit
```

Ga naar je fysieke host (je Windows 10 of 11 computer dus) en je zal zien dat je de webpagina kan openen.



This page is used to test the proper operation of the HTTP server after it has been installed. If you can read this page, it means that the HTTP server installed at this site is working properly.

#### If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting [www.example.com](http://www.example.com), you should send e-mail to "webmaster@example.com".

For information on Red Hat Enterprise Linux, please visit the [Red Hat, Inc. website](#). The documentation for Red Hat Enterprise Linux is [available on the Red Hat, Inc. website](#).

#### If you are the website administrator:

You may now add content to the webroot directory. Note that until you do so, people visiting your website will see this page, and not your content.

For systems using the Apache HTTP Server: You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page, and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

For systems using NGINX: You should now put your content in a location of your choice and edit the `root` configuration directive in the `nginx` configuration file `/etc/nginx/nginx.conf`.



Je kan ook vanaf een andere container verbinding maken met de container in het ipvlan.

We starten hiervoor eerst een tweede container op.

```
student@serverXX:~$ sudo podman run --detach -it --name containerY --hostname containerY --network ipvlan_netwerk --ip 192.168.112.181 ubi10/ubi
```

```
a2804a12387233cef67bfd1c2e3a5badfbfea7ef235bce4ca2bbc508ee796374
```

We gaan nu naar de tweede container.

```
student@serverXX:~$ sudo podman exec -it containerY bash
```

Installeer nu curl en vraag de webpagina op.

```
[root@containerY /]# dnf install -y curl
[root@containerY /]# curl http://192.168.112.180
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...

```

Verlaat nu de container.

```
[root@containerY /]# exit  
exit
```

#### 5.9.4 Scheiding met host

Ook nu kan je vanaf de host geen verbinding maken met de webserver in de container.

```
student@serverXX:~$ ping -c 1 192.168.112.180  
PING 192.168.112.180 (192.168.112.180) 56(84) bytes of data.  
From 192.168.112.100 icmp_seq=1 Destination Host Unreachable  
--- 192.168.112.180 ping statistics ---  
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms  
student@serverXX:~$ curl 192.168.112.180  
curl: (7) Failed to connect to 192.168.112.180 port 80 after 3107 ms: Could not  
connect to server
```

Je kan ervoor zorgen (ook al is dat een beveiligingsrisico) dat de containerhost (contiiis4) wel bereikbaar is vanaf de container met ipvlan door een ipvlan “Gateway” in te stellen.

Maak een nieuwe ipvlan-subinterface op de host:

```
student@serverXX:~$ sudo ip link add ipvlan_host link ens160 type ipvlan
```

Geef de nieuwe interface een IP-adres in hetzelfde subnet als het ipvlan-netwerk:

```
student@serverXX:~$ sudo ip addr add 192.168.112.222/24 dev ipvlan_host
```

Activeer nu de nieuwe interface.

```
student@serverXX:~$ sudo ip link set ipvlan_host up
```

De SSH-verbinding valt nu weg... Dit komt door het gedeeld MAC-adres. Dit is een nadeel van ipvlan I2. Je kan dit het eenvoudigst oplossen door gebruik te maken van macvlan I2. Standaard werken zowel ipvlan als macvlan in L2-mode. Over de andere mode gaan we het hier niet hebben.

Je kan nu de webpagina opvragen (in VM zelf).

```
student@serverXX:~$ curl http://192.168.112.180  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">  
...
```

```

student@serverXX:~$ curl http://192.168.112.180
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
    <head>
        <title>Test Page for the HTTP Server on Red Hat Enterprise Linux</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <style type="text/css">
            /<![CDATA[/*
                body {
                    background-color: #fff;
                    color: #000;
                    font-size: 1.1em;
                    font-family: "Red Hat Text", Helvetica, Tahoma,
                    sans-serif;
                    margin: 0;
                    padding: 0;
                    border-bottom: 30px solid black;
                    min-height: 100vh;
                    box-sizing: border-box;
                }
            */]]>
    </head>
    <body>

```

Je kan ook pingen naar de container.

```

student@serverXX:~$ ping -c 1 192.168.112.180
PING 192.168.112.180 (192.168.112.180) 56(84) bytes of data.
64 bytes from 192.168.112.180: icmp_seq=1 ttl=64 time=0.048 ms
--- 192.168.112.180 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.048/0.048/0.048/0.000 ms

```

## 5.10 Uitgebreide vergelijking netwerken podman

Eigenschap	macvlan L2	ipvlan L2	bridge rootful	pasta rootless
MAC-adres per container	Uniek	Gedeeld (zelfde als host)	Virtuele veth, niet zichtbaar op LAN	Virtuele veth, niet zichtbaar op LAN
IP-adres per container	LAN-subnet	LAN-subnet	Privé subnet (NAT)	Privé subnet (NAT, beperkt)
Zichtbaarheid op LAN	Ja, lijkt echte host	Ja, maar alle delen 1 MAC	Nee (via port forward)	Nee (via port forward)
Switch belasting	Hoog (veel MAC's)	Laag (1 MAC)	Geen invloed (alleen host zichtbaar)	Geen invloed (alleen host zichtbaar)
Beheer nodig	Root	Root	Root	Geen root

Eigenschap	macvlan L2	ipvlan L2	bridge rootful	pasta rootless
Toegang lage poorten (<1024)	Ja	Ja	Ja	Nee (tenzij workaround)
Zichtbaarheid vanuit host	Nee, extra macvlan-interface nodig	Nee, extra ipvlan-interface nodig	Ja, direct bereikbaar	Ja, direct bereikbaar (via container IP, te zoeken uit podman inspect)
Geschikt voor	Echte LAN-integratie per container	Schaalbare LAN-integratie zonder MAC-flood	Klassieke container-NAT setup (meest gebruikt)	Veilige gebruikersomgeving zonder root

## 5.11 Lokale volumes koppelen aan containers

### 5.11.1 Inleiding

Er zijn verschillende methoden om volumes aan podman toe te wijzen:

- Named volumes
  - Gecreëerd en beheerd door Docker of Podman.
  - Locaties kunnen variëren, afhankelijk van het besturingssysteem (en rootless of rootful).
  - Geschikt voor het opslaan van data die gedeeld moet worden door meerdere containers.
- Bind mounts:
  - Hiermee kun je een specifieke directory op de host koppelen aan een directory in de container.
  - De locatie op de host moet explicet worden opgegeven.
  - Gebruikt wanneer data op een specifieke locatie nodig is, zoals logbestanden.
- tmpfs volumes:
  - Data wordt alleen in het geheugen opgeslagen (niet op schijf).
  - Geschikt voor data die alleen tijdens de runtime nodig is en na het afsluiten van de container wordt gewist.
- Remote volumes
  - Podman ondersteunt direct mounts naar netwerk-filesystems.
  - CIFS en NFS worden in ieder geval ondersteund.
- iSCSI-integratie
 

Maak verbinding met een iSCSI-target op de host en koppel het als block device of filesystem. Vervolgens kan de aangekoppelde directory of device via een bind-mount aan de container worden aangeboden. iSCSI wordt mogelijk later besproken.

### 5.11.2 Stateless vs stateful

Het belangrijkste verschil tussen stateful en stateless applicaties is dat stateless applicaties geen gegevens "opslaan", terwijl stateful applicaties een vorm van opslag vereisen.

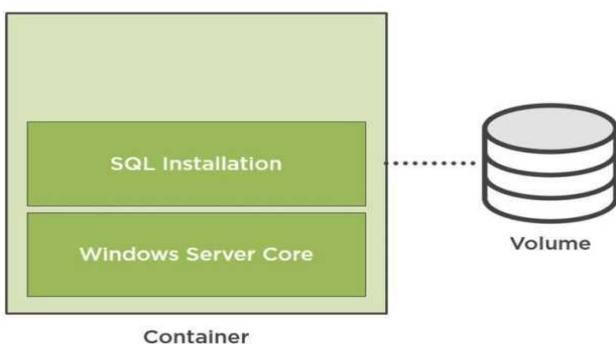
- Stateless
 

Een stateless applicatie kan worden gecreëerd en verwijderd zonder dat er zorgen zijn over

het behouden van gegevens. Je kunt de instantie eenvoudig opnieuw maken zonder dat dit gevolgen heeft voor de werking of configuratie.

- Stateful

Bij een stateful applicatie, zoals een SQL-installatie, is het behouden van de status cruciaal. Je wilt niet telkens de SQL-database opnieuw hoeven inrichten wanneer de container opnieuw wordt gestart. Dit is een situatie waarin het koppelen van een volume noodzakelijk is. Hiermee kunnen de gegevens van de SQL-installatie extern worden opgeslagen, zodat de container eenvoudig opnieuw kan worden gemaakt en opnieuw kan worden gekoppeld aan de bestaande data.



### 5.11.3 Named volume

We verwijderen eerst alle containers en netwerken.

```
student@serverXX:~$ podman rm --all -f; sudo podman rm --all -f  
...  
student@serverXX:~$ podman network prune; sudo podman network prune  
WARNING! This will remove all networks not used by at least one container.  
Are you sure you want to continue? [y/N] y  
WARNING! This will remove all networks not used by at least one container.  
Are you sure you want to continue? [y/N] y
```

De handmatig aangemaakte `ipvlan_host`-interface op ServerXX dien je ook nog te verwijderen.

```
student@serverXX:~$ sudo ip link delete ipvlan_host
```

Herstart nu sshd zodat sshd gebruik maakt van de nieuwe routeringsinstellingen.

```
student@serverXX:~$ sudo systemctl restart sshd
```

SSH werkt nu weer 😊.

Maak een volume aan genaamd `mijn\_data\_volume`:

```
student@serverXX:~$ podman volume create mijn_data_volume
```

```
mijn_data_volume
```

Deze stap is niet verplicht. Podman zal zelf het volume bij de volgende stap aanmaken als je deze stap niet hebt uitgevoerd.

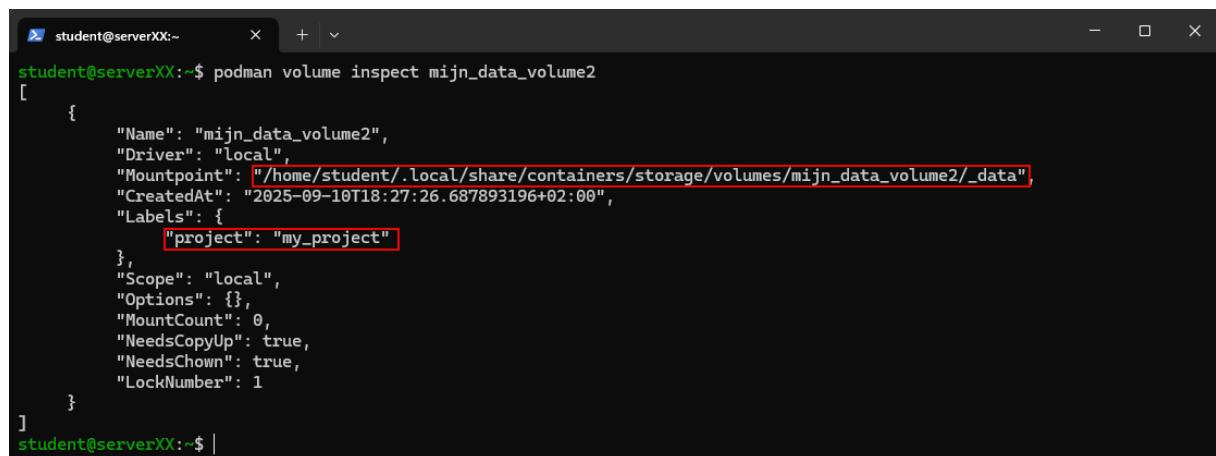
Je kan wel een label toevoegen als je zelf eerst het volume aanmaakt.

```
student@serverXX:~$ podman volume create mijn_data_volume2 --label  
"project=my_project"  
  
mijn_data_volume2
```

Labels kunnen handig zijn om volumes te onderscheiden.

Je kan via volume inspect de labels zien.

```
student@serverXX:~$ podman volume inspect mijn_data_volume2
```



```
student@serverXX:~$ podman volume inspect mijn_data_volume2  
[  
  {  
    "Name": "mijn_data_volume2",  
    "Driver": "local",  
    "Mountpoint": "/home/student/.local/share/containers/storage/volumes/mijn_data_volume2/_data",  
    "CreatedAt": "2025-09-10T18:27:26.687893196+02:00",  
    "Labels": {  
      "project": "my_project"  
    },  
    "Scope": "local",  
    "Options": {},  
    "MountCount": 0,  
    "NeedsCopyUp": true,  
    "NeedsChown": true,  
    "LockNumber": 1  
  }  
]  
student@serverXX:~$ |
```

Daarnaast zie je ook waar de data bewaard wordt op schijf:

/home/student/.local/share/containers/storage/volumes/mijn\_data\_volume2/\_data.

We starten nu een container en koppelen het volume aan een map binnen de container.

```
student@serverXX:~$ podman run -d --name conta -v mijn_data_volume:/test  
ubi10/ubi-init
```

In dit voorbeeld wordt de directory “/test” in de ubi10/ubi-container gekoppeld aan het volume “mijn\_data\_volume”. Dit betekent dat alle data die hier wordt opgeslagen, behouden blijft, zelfs als de container opnieuw wordt gemaakt.

We gaan in de container en zetten een bestand in /test.

```
student@serverXX:~$ podman exec -it conta bash  
[root@3e43b082ac47 /]# echo "dit is een test" > /test/bestandtest.txt
```

We gaan nu uit de container en verwijderen de container.

```
[root@3e43b082ac47 /]# exit  
exit  
  
student@serverXX:~$ podman rm conta -f  
conta
```

De data staat nog op ServerXX.

```
student@serverXX:~$ ls -l  
/home/student/.local/share/containers/storage/volumes/mijn_data_volume/_data  
totaal 4  
-rw-r--r--. 1 student student 16 10 sep 18:43 bestandtest.txt
```

We starten een nieuwe container op en koppelen mijn\_data\_volume aan deze nieuwe container.

```
student@serverXX:~$ podman run -d --name contb -v mijn_data_volume:/test  
ubi10/ubi-init  
  
6804d22cb76a558784100970c39733b2d3c02d9ee078c5aa30477a0ff158f89f
```

We zullen nu vanuit deze nieuwe container de inhoud bekijken van /test/bestandtest.txt.

```
student@serverXX:~$ podman exec -it contb bash -c "cat /test/bestandtest.txt"  
dit is een test
```

Zoals je ziet is vanuit container contb de data beschikbaar die aangemaakt is door container conta.

Je kan al je volumes als volgt bekijken.

```
student@serverXX:~$ podman volume ls  
  
DRIVER      VOLUME NAME  
  
local      mijn_data_volume  
  
local      mijn_data_volume2
```

Je kan enkel de volumenaam als volgt tonen.

```
student@serverXX:~$ podman volume ls -q  
  
mijn_data_volume  
  
mijn_data_volume2
```

Je kan hiervan gebruik maken om via command substitution alle volumes die niet door een container gebruikt worden tegelijk te verwijderen.

We stoppen dus eerst alle conta en contb.

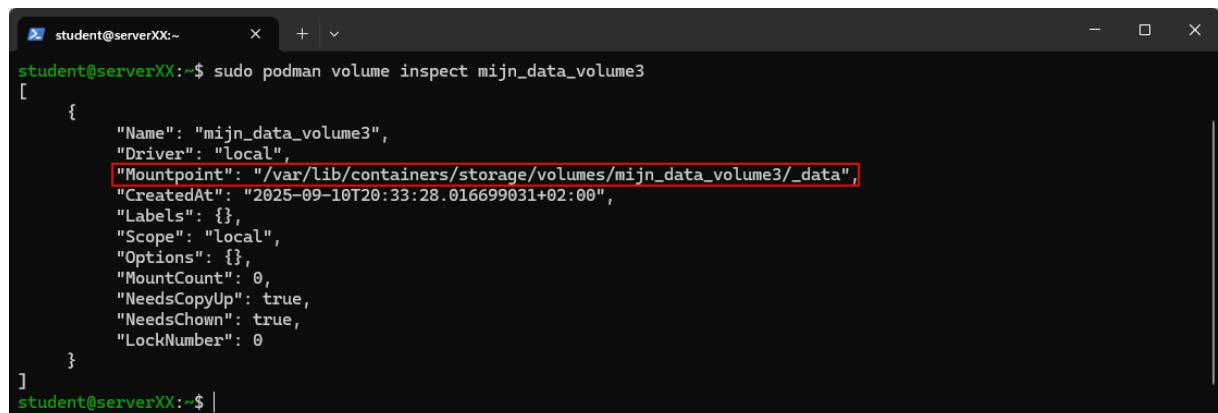
```
student@serverXX:~$ podman rm contb -f  
contb
```

We verwijderen nu alle volumes tegelijk.

```
student@serverXX:~$ podman volume rm $(podman volume ls -q)  
mijn_data_volume  
mijn_data_volume2
```

PS: Als je met sudo-rechten een volume creëert wordt de data op een andere locatie bewaard. Voor het overige is het principe hetzelfde. We verwijderen erna hier het volume.

```
student@serverXX:~$ sudo podman volume create mijn_data_volume3  
student@serverXX:~$ sudo podman volume inspect mijn_data_volume3
```



```
student@serverXX:~$ sudo podman volume inspect mijn_data_volume3  
[  
  {  
    "Name": "mijn_data_volume3",  
    "Driver": "local",  
    "Mountpoint": "/var/lib/containers/storage/volumes/mijn_data_volume3/_data",  
    "CreatedAt": "2025-09-10T20:33:28.016699031+02:00",  
    "Labels": {},  
    "Scope": "local",  
    "Options": {},  
    "MountCount": 0,  
    "NeedsCopyUp": true,  
    "NeedsChown": true,  
    "LockNumber": 0  
  }  
]  
student@serverXX:~$ |
```

```
student@serverXX:~$ sudo podman volume rm mijn_data_volume3  
mijn_data_volume3
```

#### 5.11.4 Bind mount

Bij een bind mount maak je eerst zelf een map aan op de container host en koppel je die bij het aanmaken van een container aan een map in de container.

We maken een map srv/contdata aan op ServerXX.

```
student@serverXX:~$ sudo mkdir -p /srv/contdata
```

We maken een tekstbestand data1.txt met als inhoud “Here is some data!” aan in de map /srv/contdata.

```
student@serverXX:~$ echo "Here is some data!" | sudo tee  
/srv/contdata/data1.txt
```

We maken nu een container aan met volume mapping.

```
student@serverXX:~$ sudo podman run -d --name cont1 --hostname cont1 -v /srv/contdata:/data:Z ubi10/ubi-init
```

-v /srv/contdata:/data Volume mapping die een map op de host (/srv/contdata) koppelt aan een map in de container (c:\data).

:Z heeft te maken met SELinux op RHEL/Fedora/CentOS.

:Z = exclusieve toegang (dit wil je meestal).

:z = gedeelde toegang (als meerdere containers tegelijk dezelfde map mounten).

We gaan nu naar de container en starten bash.

```
student@serverXX:~$ sudo podman exec -it cont1 bash
```

We kijken nu of we dat data via de mapping kunnen benaderen.

```
[root@cont1 /]# ls /data/data1.txt  
/data/data1.txt
```

We gaan nu content toevoegen aan data1.txt vanuit de container.

```
[root@cont1 /]# echo "Here is EXTRA data!" >> /data/data1.txt
```

We verlaten de container door exit in te typen.

```
[root@cont1 /]# exit  
exit
```

We gaan de container die op de achtergrond uitgevoerd wordt stoppen.

```
student@serverXX:~$ sudo podman stop cont1  
cont1
```

We gaan nu een nieuwe tweede container uitvoeren en zullen zien dat deze aan de data kan die juist is opgeslagen door cont1.

```
student@serverXX:~$ sudo podman run -d --name cont2 --hostname cont2 -v /srv/contdata:/data:z ubi10/ubi-init
```

We gaan nu via container cont2 de inhoud bekijken van /srv/contdata op ServerXX.

```
student@serverXX:~$ sudo podman exec -it cont2 /bin/sh -c "cat /data/data1.txt"  
Here is some data!  
Here is EXTRA data!
```

Eerst heeft cont1 inhoud toegevoegd aan /srv/contdata/data1.txt, daarna is cont1 afgesloten, is cont2 gestart en kan cont2 de inhoud bekijken van de data die opgeslagen is door cont1.

Cont2 kan uiteraard ook data toevoegen aan /srv/contdata/data1.txt op ServerXX.

```
student@serverXX:~$ sudo podman exec -it cont2 /bin/sh -c 'echo "Here is even more data!" >> /data/data1.txt'
```

We bekijken nu de data in het bestand /srv/contdata/data1.txt op ServerXX via cont1.

Start hiervoor eerst cont1 terug op.

```
student@serverXX:~$ sudo podman start cont1
```

Bekijk nu de inhoud van het bestand

```
student@serverXX:~$ sudo podman exec -it cont1 /bin/sh -c "cat /data/data1.txt"
Here is some data!
Here is EXTRA data!
Here is even more data!
```

Dit geeft je een idee van de mogelijkheden die deze verschillende containers hebben om met gedeelde gegevens te werken zodra deze is gekoppeld aan verschillende containers.

Het proces van het daadwerkelijk maken van die koppeling met het volume vindt plaats wanneer we de container aanmaken.

We moeten dit doen op het moment dat we de container starten, omdat dat de manier is waarop containers zijn ontworpen om te functioneren.

PS Je mag ook directories uit je eigen home-directory binden als je rootless podman wil gebruiken. Je kan geen directories binden waarvoor je geen permissies hebt. Je kan uiteraard chmod of ACL's gebruiken om een gewone gebruiker toegang te geven tot een bepaalde directory. Buiten de rechten is het exact hetzelfde om te gebruiken.

### 5.11.5 tmpfs volume

Een tmpfs volume heeft als eigenschap dat het niet beschikbaar is nadat een container is afgesloten.

Je voegt de --tmpfs-optie toe bij het aanmaken van een container om een tmpfs-volume te maken.

```
student@serverXX:~$ podman run --detach -i --name mijn_container --tmpfs /tijdelijk:rw,size=64M ubi10/ubi
948f2de74766e311d9e61fab968f11fdf69c466eb431c2264d1f3a700d61c576
```

```
--tmpfs /tijdelijk:rw,size=64M
```

Hiermee maak je een tijdelijk tmpfs-volume aan dat aan de map `/tijdelijk` in de container wordt gekoppeld. Dit werkt als volgt:

`/tijdelijk`: De directory in de container waar het tmpfs-volume wordt gemount.

`rw`: Geeft lees- en schrijfrechten in dit tmpfs-volume.

`size=64M`: Bepaalt de maximale grootte van het tmpfs-volume op 64 MB. Het tmpfs-volume gebruikt het RAM-geheugen van de host voor deze opslag.

We gaan nu in de container en checken de grootte van `/tijdelijk`.

```
student@serverXX:~$ podman exec -it mijn_container bash
[root@948f2de74766 /]# df -h /tijdelijk
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           64M    0    64M   0% /tijdelijk
```

We checken ook of we een bestand van 10MB kunnen aanmaken in `/tijdelijk`.

```
[root@948f2de74766 /]# dd if=/dev/zero of=/tijdelijk/testfile.img bs=10M
count=1
1+0 records in
1+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0.0168512 s, 622 MB/s
```

We checken nu nog even de grootte van `/tijdelijk`

```
[root@948f2de74766 /]# df -h /tijdelijk
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           64M   10M   54M  16% /tijdelijk
```

Zoals je ziet is nu 10MB opgebruikt.

We verlaten de container.

```
[root@948f2de74766 /]# exit
exit
```

De ruimte is nu terug vrijgegeven.

Je vraagt je waarschijnlijk af wat je daaraan hebt... Je kan tmpfs volumes o.a. gebruiken voor tijdelijke opslag in container (`/tmp`, `cache`, `lockfiles`) en voor veilige opslag (niets blijft over op schijf).

## 5.11.6 Remote volumes toevoegen aan containers

We gaan dit concreet uitvoeren:

- We maken een gedeelde map /srv/gedeeld met daarin een bestand data.txt aan op ClientXX.
- We koppelen (mounten) de gedeelde map op ServerXX.
- We maken een container cont3 aan en maken dat deze container toegang heeft tot de gedeelde share.

Maar we beginnen bij het begin... De gedeelde map aanmaken op ClientXX.

Download nfs-utils.

```
student@clientXX:~$ sudo dnf install -y nfs-utils
```

We maken nu de gedeelde map /srv/gedeeld aan.

```
student@clientXX:~$ sudo mkdir -p /srv/gedeeld
```

We zetten hierin een bestand data.txt.

```
student@clientXX:~$ echo 'Dit zijn gegevens' | sudo tee /srv/gedeeld/data.txt
Dit zijn gegevens
```

We zorgen er nu voor dat dit gedeeld wordt via NFS.

```
student@clientXX:~$ sudo nano /etc/exports
/srv/gedeeld 192.168.112.100(rw)
```

We starten nu de NFS server en zorgen er ook voor dat die gestart wordt na reboot.

```
student@clientXX:~$ sudo systemctl enable --now nfs-server
Created symlink '/etc/systemd/system/multi-user.target.wants/nfs-
server.service' → '/usr/lib/systemd/system/nfs-server.service'.
```

We checken voor de zekerheid nu of de NFS-server draait.

```
student@clientXX:~$ systemctl status nfs-server
...
Active: active (exited) since Wed 2025-09-03 20:41:42 CEST; 34s ago
...

```

Deel nu de map met 192.168.112.100.

```
student@clientXX:~$ sudo exportfs -rav
exporting 192.168.112.100:/srv/gedeeld
```

Stel de firewall correct in voor NFS.

```
student@clientXX:~$ sudo firewall-cmd --zone=public --permanent --add-service=nfs

student@clientXX:~$ sudo firewall-cmd --zone=public --permanent --add-service=mountd

student@clientXX:~$ sudo firewall-cmd --zone=public --permanent --add-service=rpc-bind

student@clientXX:~$ sudo systemctl restart firewalld
```

Op ServerXX mounten we nu de NFS-share.

We installeren hiervoor eerst weer nfs-utils.

```
student@serverXX:~$ sudo dnf install -y nfs-utils
```

We maken een gedeelde map aan /nfs/gedeeld.

```
student@serverXX:~$ sudo mkdir -p /nfs/gedeeld
```

We zullen nu de share mounten.

```
student@serverXX:~$ sudo mount -t nfs 192.168.112.200:/srv/gedeeld /nfs/gedeeld
```

We starten nu een container op ServerXX om de toegang tot de share te bekijken.

```
student@serverXX:~$ sudo podman run -d --name contV --hostname contV -v /nfs/gedeeld:/contdata ubi10/ubi-init
```

We bekijken nu de gegevens.

```
student@serverXX:~$ sudo podman exec -it contV bash -c 'cat /contdata/data.txt'
```

Dit zijn gegevens

-l bij -lc zorgt ervoor dat een login shell wordt gestart (configuratiebestanden Bash worden zo geladen).

Zoals je ziet werkt dit.

Je kan nu wel geen gegevens wegschrijven in de container omdat de root standaard naar de gebruiker nobody wordt omgeleid bij NFS. Dit hebben jullie gezien bij de OLOD Linux Advanced.